Classification-based Phrase Structure Grammar: an Extended Revised Version of HPSG

Richard Paul Cooper

Ph.D.

University of Edinburgh

1990

Declaration

I declare that this thesis has been composed by myself and that the research reported herein has been conducted by myself unless otherwise indicated.

Edinburgh, 8th October 1990.

Richard Cooper

For

my parents

Acknowledgements

First and foremost I owe a debt of gratitude to my principle supervisor, Dr. Robin Cooper. This thesis owes as much to his guidance and ever-enduring patience as to anything, and without either the thesis would not be here now. Robin though has been more than just a supervisor, and on this score Elisabet Engdahl also deserves special mention. Her advice and encouragement have always been appreciated.

Intellectually, the material in this thesis, and my perspective on it, has been directly influenced by almost every member of the Centre, as well as many visitors. The most influential of these have been Guy Barry, Patrick Blackburn, Nick Braisby, Kathrin Cooper, Robin Cooper, Elisabet Engdahl, Brad Franks, Claire Gardent, Mark Gawron, Lex Holt, Ewan Klein, Ian Lewin, Glyn Morrill, Stanley Peters, Martin Pickering, Mike Reape, Catrin Rhys, Bill Rounds, Jerry Seligman, Mark Steedman, and Pete Whitelock. Such varied influences could be culled only from the outstanding research environment provided by the Centre for Cognitive Science.

I am also most grateful to the British Council and the Association of Commonwealth Universities for providing me with funds whilst pursuing this research, and especially Alison of the British Council for always being so helpful.

Lastly I thank my family and my friends, both from near and from far. At no stage did my lack of correspondence reflect my true wishes to communicate.

Abstract

This thesis is concerned with a presentation of *Classification-based Phrase Structure Grammar* (or CPSG), a grammatical theory that has grown out of extensive revisions of, and extensions to, HPSG. The fundamental difference between this theory and HPSG concerns the central role that classification plays in the grammar: the grammar *classifies* strings, according to their feature structure descriptions, as being of various types. Apart from the role of classification, the theory bears a close resemblance to HPSG, though it is by no means a direct translation, including numerous revisions and extensions. A central goal in the development of the theory has been its computational implementation, which is included in the thesis.

The presentation may be divided into four parts. In the first, chapters 1 and 2, we present the grammatical formalism within which the theory is stated. This consists of a development of the notion of a classificatory system (chapter 1), and the incorporation of hierarchality into that notion (chapter 2).

The second part concerns syntactic issues. Chapter 3 revises the HPSG treatment of specifiers, complements and adjuncts, incorporating ideas that specifiers and complements should be distinguished and presenting a treatment of adjuncts whereby the head is selected for by the adjunct. Chapter 4 presents several options for an account of unbounded dependencies. The accounts are based loosely on that of GPSG, and a reconstruction of GPSG's Foot Feature Principle is presented which does not involve a notion of default. Chapter 5 discusses coordination, employing an extension of Rounds-Kasper logic to allow a treatment of cross-categorial coordination.

In the third part, chapters 6, 7 and 8, we turn to semantic issues. We begin (Chapter 6) with a discussion of Situation Theory, the background semantic theory, attempting to establish a precise and coherent version of the theory within which to work. Chapter 7 presents the bulk of the treatment of semantics, and can be seen as an extensive revision of the HPSG treatment of semantics. The aim is to provide a semantic treatment which is faithful to the version of Situation Theory presented in Chapter 6. Chapter 8 deals with quantification, discussing the nature of quantification in Situation Theory before presenting a treatment of quantification in CPSG. Some residual questions about the

semantics of coordinated noun phrases are also addressed in this chapter.

The final part, Chapter 9, concerns the actual computational implementation of the theory. A parsing algorithm based on hierarchical classification is presented, along with four strategies that might be adopted given that algorithm. Also discussed are some implementation details. A concluding chapter summarises the arguments of the thesis and outlines some avenues for future research.

Contents

Chapter 1

Background

1 Introduction

1.1 Aims, Goals and Scope of the Thesis

In [Pollard & Sag 87], a theory of grammar, Head-driven Phrase Structure Grammar (HPSG), is presented which attempts to be both formally precise and computationally implementable. This thesis presents a number of extensions and revisions to HPSG, together with a computational implementation of the revised theory. In some sense, the computational implementation is the primary goal, but in progressing towards this goal many questions are raised. These questions present their own subgoals, which are in many ways more important. These subsidiary goals may be briefly summarised as follows:

- revise the formal machinery behind the theory,
- revise certain aspects of the syntactic coverage of [Pollard & Sag 87],
- extend the syntactic coverage,
- revise the semantics of the current coverage,
- give the semantics for the extended coverage, and
- construct a computational implementation of the theory.

The revision of the formal machinery and the use of sorted feature structures is in part motivated by computational considerations: difficulties arise in implementing a system

of sorted feature structures where the sorts are hierarchically structured. However, these difficulties point to a system whereby sorting or typing information is not part of a feature structure. It is thus beneficial to revise the formal machinery so as to develop a coherent system which respects the distinction between typing information and feature structures.

The major revisions with respect to the coverage of HPSG as presented in [Pollard & Sag 87] are all motivated by phenomena which HPSG finds difficult to treat adequately. Difficulties with bare plural noun phrases and non-predicative prepositional phrases lead to modifications in the treatment of subcategorisation, which in turn suggest a more elegant treatment of subject-auxiliary inversion. These modifications include insights from $\overline{X}$-theory, and the grammar rules of our revised system bear a close resemblance to those of a standard version of $\overline{X}$-theory. Further modifications are suggested by potential problems arising from the use of an obliqueness hierarchy to totally order arguments, and the modifications we employ allow such verbs as 'argue', which take two optional but equally oblique prepositional phrase complements, to be incorporated naturally. The treatment of adjuncts in HPSG is also problematic, and by taking adjuncts to select for the heads that they modify, we provide a treatment which overcomes many of these problems. Lastly, there are numerous questions which remain unanswered concerning the semantic component of an HPSG grammar, and we also attempt to address these questions.

There are two major areas in which the coverage of HPSG is extended. Firstly, we discuss in detail various options for the treatment of unbounded dependencies. Even before constraints on movement are considered, unbounded dependencies raise numerous questions which should be addressed before any proper treatment can be attempted. Recent unpublished work by Pollard and Sag ([Pollard & Sag ∞a]) has addressed some of these questions, but this thesis aims to examine closely the options available, highlighting their heritage and faults, without stipulating one approach over the others. The second extension in coverage concerns the treatment of coordination. Almost nothing is said in [Pollard & Sag 87] about coordination, and it is far from clear how any previous treatment can easily be incorporated into an HPSG-like formalism. Our treatment, which involves a motivated extension to the logic of feature structures, captures the

potentially problematic cases of cross-categorial coordination.

HPSG claims to provide an integrated treatment of syntax and semantics, and as such for completeness we need to complement the above syntactic aims by providing a treatment of semantics which is consistent with the revised treatment of syntax and which also covers the above proposed syntactic extensions.

The final aim, that of developing a faithful computational implementation of the theory, is that which drives each of the above subgoals. At various points throughout the thesis decisions are influenced by computational considerations. The interaction between the grammar design and the implementation works both ways, however. The implementation does not aim to be especially efficient. Rather, it aims to reflect as faithfully as possible the structure of the grammar, and hierarchical structuring, which we take to be of major importance within the grammar, is central to the parsing algorithm developed.

Amongst the positive aspects advertised of HPSG is that it in many ways represents a coming together of various schools of thought. Whilst HPSG is most closely derived from GPSG ([Gazdar *et al.* 85]), it also draws heavily upon other modern linguistic theories, including Government-Binding theory ([Chomsky 81], [Chomsky 82]), Categorial Grammar ([Lambek 58], [Geach 72], [Ades & Steedman 82]), Lexical Functional Grammar ([Bresnan 82a]), Unification Categorial Grammar ([Zeevat *et al.* 87]) and Categorial Unification Grammar ([Uszkoreit 86]). HPSG draws on each of these current theories for its treatment of a variety of phenomena, and in doing so many of the ideas of what have often been seen as competing theories are unified. Many of the revisions suggested in this thesis similarly are drawn from previous work in other theories, and as such the thesis may be seen as one more step in this unifying process.

Many questions are outside the scope of the thesis. In particular, questions regarding precise constraints on extraction and binding, and the interaction of anaphora and quantification. The nature of such constraints is far from clear — their assumed syntactic nature is only now beginning to be questioned — and the theory presented really only makes options available. Some of these issues are addressed, though, in [Pollard & Sag ∞a].

The thesis is also concerned almost exclusively with English. Insights from other researchers into other languages are employed, but no analyses are presented, and the question of language universals is only tentatively broached. Furthermore, whilst a large syntactic coverage with an associated semantics is achieved, the result is still only a fragment: there remains much that is considered grammatical English which is beyond the fragment and beyond the scope of the thesis.

Our extensions and modification hardly represent a theory in themselves. However, to distinguish the revised theory from the original, we refer to the revised theory as *Classification-based Phrase Structure Grammar*, or CPSG. The name emphasises the classificatory basis of the theory.

1.2 A Guide to the Thesis

The remainder of this chapter focuses on preliminary issues. We begin with a discussion of the use of typed or sorted feature structures within HPSG, and argue for a different relationship between linguistic tokens and feature structure descriptions than that employed by [Pollard & Sag 87]. In particular, we pursue a notion of token and type which is independent from the use of feature structure descriptions. This is formalised in terms of Seligman's notion of a classification ([Seligman 90a, 90b]). Our linguistic applications require that this system be extended to include structured tokens. After presenting this extension, we consider the use of feature structure descriptions to mediate the *is of type* relation. We term the resulting system of feature structure descriptions and types, which may be used to classify described tokens, a *classificatory system*. It is in this that we formalise CPSG. The chapter concludes with a discussion of feature structure logics, following that of [Kasper & Rounds 86], [Rounds & Kasper 86] and [Kasper & Rounds 90], the purpose of which is to make precise our usage of feature structures, given the abundance of feature structure formalisations in current circulation.

Chapter 2 focuses on hierarchically structured classificatory systems, drawing on examples from the lexical hierarchy for motivation, though the principal concern of the chapter is the extrapolation of the hierarchical structuring of the lexicon to all constituents: both lexical and phrasal. The notion of a *constituent hierarchy* is then illustrated with examples from categorial grammar, HPSG as presented in [Pollard & Sag 87], and CPSG,

and it is suggested that such a hierarchy may be a language universal.

Having established the formal structure of CPSG in terms of a hierarchy of types with associated feature structure descriptions, we move on to syntactic issues, and Chapter 3 discusses the treatment of specifiers, complements and adjuncts. There are many very significant difference between the treatment of arguments in HPSG and CPSG, including the separation of subcategorised for arguments into specifiers and complements (following [Borsley 87]) and the use of sets, rather than lists, to describe subcategorisation requirements (with the associated use of an attribute to encode the grammatical function of a constituent). The treatment of adjuncts in CPSG is also different to that in HPSG, with adjuncts selecting for heads, rather than heads selecting for adjuncts.

Chapters 4 and 5 go beyond the original coverage of HPSG as presented in [Pollard & Sag 87], examining the treatment of unbounded dependencies and coordinate phrases respectively. In Chapter 4 an approach to unbounded dependencies similar to that of GPSG is advocated, with three stages: slash introduction, slash percolation and slash termination. Slash introduction is treated by a rule licensing head/filler phrases, which are compared with head/specifier phrases — the constituent hierarchy of CPSG includes a node dominating both. In the discussion of slash percolation, a reconstruction of GPSG's Foot Feature Principle which does not employ defaults is presented, and possible constraints on slash percolation are considered. We then consider the options available for slash termination, with an approach employing lexical rules being favoured.

The principal innovation of Chapter 5 concerns an augmentation to the logic of feature structures presented in Chapter 1, allowing composite feature structures to be formed. Such feature structures are employed in the CPSG treatment of cross-categorial coordination, which captures the intuition that in cross-categorial coordination the coordinate phrase bears a composite syntactic category: a category which is something of a hybrid of the categories of each of the conjuncts.

Chapters 6, 7 and 8 centre on semantic issues. Chapter 6 discusses in detail situation theory, which underlies the treatment of semantics in both HPSG and CPSG. The presentation consists of more than just a summary of previous work in situation theory, attempting to present in one place a coherent theory and including some comments on

the mathematical modelling of various situation theoretic objects. Chapter 7 focuses on the possible values of the SEMANTICS attribute, discussing the treatment of semantics for the syntactic fragment of chapters 3, 4 and 5. In Chapter 8 the issue of quantification and quantifier scoping is addressed. It is argued that quantification should be treated within situation theory as "structurally determined", and that as such should be treated in terms of situation theoretic propositions, rather than infons. Also discussed in this chapter is the treatment of distributive readings for coordinated noun phrases.

As one of the goals of the thesis, much of CPSG has been implemented, and Chapter 9 summarises the major points of this implementation, with a discussion of the use of a constituent hierarchy in parsing, a discussion of the implementation of the lexicon in CPSG, and a discussion of the novel programming techniques employed to improve efficiency.

The concluding chapter, Chapter 10, presents the results and summaries the arguments of the thesis, before pointing to some further research questions which the thesis raises. This is followed by two appendices. The first is a detailed fragment of CPSG, and the second contains a code listing of the Prolog implementation.

2 The Organisation of HPSG and CPSG

Central to HPSG is the notion of modelling linguistic information. Following much recent work in formal and computational linguistics ([Kay 79], [Karttunen 86], [Uszkoreit 86], [Shieber 86]), this modelling is achieved through a system of *feature structures* which (partially) describe linguistic tokens. In describing such tokens, HPSG employs a number of augmentations to the basic feature structure formalism. These augmentations include the use of logical operators, "functional dependencies", list and set valued attributes, and the typing, or sorting, of feature structures. We discuss most of these argumentations (the use of logical operators, functional dependencies and list and set valued attributes) in section 5. Here we are concerned only with the organisation of the domain of linguistic tokens and the relationship between these and the feature structures used to describe them. We take the sorting of features structures to be of central importance to this relationship.

It should be noted that in what follows we draw a sharp distinction between *types* and *sorts*. We refer to the feature structures of HPSG as sorted, and restrict the word 'type' to the field of classification. This distinction is clarified throughout this section.

2.1 The Nature of Linguistic Theory (According to HPSG)

[Pollard 88] and [Pollard & Sag ∞a] argue that a linguistic theory should be no different in its nature and structure than any other scientific theory, and as such a linguistic theory should consist of statements about a model, rather than about the empirical domain, with the predictive power of the theory being due to correspondences between the model and the empirical domain. Three domains are involved in this picture: the empirical domain, whose elements might be termed "linguistic events", the model-theoretic domain, and the domain of statements about the model.

In making statments about the model, HPSG employs a description language consisting of *sorted feature structures*, which we represent orthographically in terms of attribute value matrices (AVMs). The model-theoretic domain consists of finite state automata, which may or may not *satisfy* the feature structure descriptions. The general picture then is as in Figure 1.1.

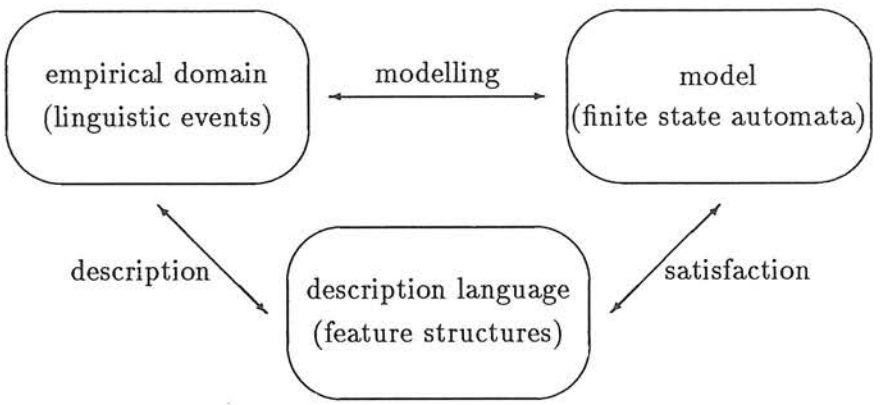


Figure 1.1: The Nature of Linguistic Theory

To illustrate these three domains and their relationships, consider an utterance of the determiner 'the', an element of the empirical domain. In HPSG it (like all utterances of 'the') is partially described by:

$$\textit{lexical sign}\left[\begin{array}{ll}\text{PHON} & \textit{the} \\ \text{SYN} & \left[\text{LOC}\left[\begin{array}{ll}\text{HEAD} & \left[\text{MAJ}\ \ D\right] \\ \text{SUBCAT} & \langle\,\rangle \\ \text{LEX} & +\end{array}\right]\right]\end{array}\right]$$

This sorted feature structure, an element of the description language domain, is satisfied by a class of automata, those having one arc from their root node labelled 'PHON' leading to the final state '*the*' and a second arc from their root node labelled 'SYN' leading to a non-final state which in turn has an arc labelled 'LOC' leading to another non-final state which has arcs labelled 'HEAD', 'SUBCAT' and 'LEX', and so on. Diagrammatically, the feature structure is satisfied by any automaton with root node q_0 having the automaton in Figure 1.2 as a sub-automaton.

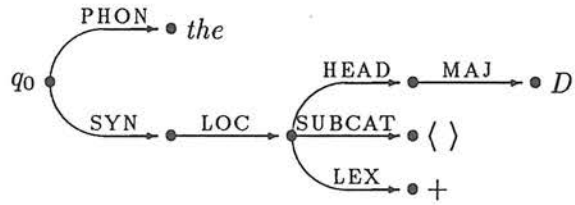


Figure 1.2: The Minimal Automaton Modelling 'the'

Within this section we shall mostly be concerned with the description relation — the relationship between linguistic events and their feature structure descriptions. This is where the principal difference between the underlying frameworks of HPSG and CPSG lies.

2.2 Sorted Feature Structures

Following similar work in unification-based grammar formalisms, [Pollard & Sag 87] introduce the notion of *type*, or *sort*, into the domain of feature structures to capture the fact that "different attributes make sense for different kinds of objects" ([Pollard & Sag 87, p. 39]). To motivate the use of sorted feature structures, they consider a feature structure which describes a syntactic category. For such a feature structure, the

attribute PHONOLOGY does not make sense — syntactic categories are described by feature structures which are of a sort for which the attribute PHONOLOGY is inappropriate. The "description" relation in HPSG can, therefore, be depicted more accurately as in Figure 1.3, where the grid on the description language is intended to represent sorting of the domain.

Figure 1.3: The Description Relation in HPSG

2.2.1 Appropriate Values

The notion of sort outlined above involves the notion of appropriate attributes. A notion of an *appropriate value* for an attribute may also be incorporated into a system of feature structures to capture the fact that different values are appropriate for different attributes. A syntactic category, for example, is an inappropriate value for the attribute PHONOLOGY. Although such a notion may be introduced independently from the notion of sort, HPSG treats appropriate values in terms of sorts — a sort is associated with each attribute and only feature structures (or atoms) of the sort associated with an attribute are appropriate values for that attribute. Such an approach requires atomic values, as well as feature structures (including feature structures which are embedded in other feature structures as values of attributes) to be sorted.

The appropriate values for an attribute may be intrinsic to that attribute, or determined by the sort of feature structure in which the attribute occurs. Subtle interactions stemming from the sort hierarchy (discussed below) suggests that it is the second of these notions of appropriateness which is required by HPSG, though this is by no means clear.

2.2.2 Closed and Open Sorts

Two distinct notions of 'sort' occur in the literature: *closed sorts* and *open sorts*. In a system with closed sorts, the sort of a feature structure determines exactly which attributes are appropriate for that feature structure. In an open sorted system, the sorts specify which attributes are required by feature structures, but individual feature structures may involve attributes over and above those specified by their sort. Thus if the sort *sign* requires the attributes PHONOLOGY, SYNTAX and SEMANTICS, then in a closed sorted system a feature structure of sort *sign* must be defined on all and only those three attributes, but in an open sorted system a feature structure of sort *sign* may be defined on additional attributes, including perhaps a DAUGHTERS attribute. In a system with closed sorts, the sorts allow attributes that are inappropriate for a feature structure to be distinguished from those which are appropriate but undefined.

Within HPSG, all strings admitted by a grammar are described by feature structures of sort *sign*. This includes both words and phrases. For words, the attributes PHONOLOGY, SYNTAX, and SEMANTICS are appropriate. The additional attribute DAUGHTERS is used in describing phrasal constituents. HPSG thus requires an open sorted system, with the appropriate attributes for the sort *sign* being PHONOLOGY, SYNTAX, and SEMANTICS. This, however, does not capture the fact that the attribute DAUGHTERS is inappropriate for a feature structure describing a lexical constituent, nor the fact that the attribute PHONOLOGY is inappropriate for a feature structure of the sort which describes a syntactic category. Arguments based on the inappropriateness of features suggest that the sorts should be closed: open sorted systems introduce a notion of appropriateness for attributes, but not a notion of inappropriateness. As the use of sorts in HPSG is motivated by considerations of inappropriateness, the precise nature of the notion of sort employed by HPSG is unclear.

If the notion of appropriate value were independent of that of sort, this problem could be circumvented by the addition of a further distinguished value. If, for example, the appropriate values for the DAUGHTERS attribute were extended to include the distinguished value *null*, then that attribute could be considered appropriate for descriptions of both lexical and phrasal items and a system of closed sorts could be employed. In such a system descriptions of lexical items would always extend [DAUGHTERS *null*].

This solution is not available in a system where appropriate values are determined by sorts, as in such a system *null* can only be appropriate for atomic valued attributes yet phrasal constituents require the value of DAUGHTERS to be complex.

2.2.3 Sorts and Subsorts

HPSG also employs an ordering on the set of sorts. Lexical items are described by signs of sort *lexical sign* and phrasal items are described by signs of sort *phrasal sign*, both of which are subsorts of *sign*. Subsorts of *lexical sign* and *phrasal sign* are also employed, as are subsorts of those sorts. This leads to a *hierarchy* of sorts, but raises questions about the role of sorting information. It is reasonable to require subsorts to partition sorts in the sense that if the set of subsorts of x is y (and y is non-empty), then if a feature structure is of sort x it must be of one and only one sort $x' \in y$. Given this, it is possible to reflect much of the lattice structure which exists on the domain of feature structures in terms of the sort hierarchy, in which case the sort of a feature structure abbreviates much of the information conveyed by the attribute-value pairs of the feature structure. Sorts in HPSG appear to be used primarily (if not totally) for this purpose, as abbreviations.

2.2.4 Atomic Values and Sorts

Atomic values are themselves sorts. This is best illustrated in terms of directed acyclic graphs (DAGs), where the use of sorts can be seen in terms of labelling the non-terminal nodes of the graph. When feature structures are represented as DAGs, the edges of the DAGs are labelled with attributes and the terminal nodes are labelled with atomic values. Extrapolating the labelling of non-terminal nodes with sorts to labelling terminal nodes suggests that atomic values are just sorts with no appropriate attributes. Subsorts play a role here as well. The sort *boolean*, for example, has no appropriate attributes and two subsorts, + and −.

It should be pointed out that in an open sorted system, the notion of atomic value is somewhat blurred, for if an atomic value is just a sort with no appropriate attributes, there is no reason why, in an open sorted system, that sort cannot have subsorts with appropriate values.

2.2.5 Subsorts and Appropriate Values

Subsorts play a role in determining appropriate values for attributes. If a feature structure or value is of sort x, and x is a subsort of y, then that feature structure or value must also be of sort y. So if subsorts partition sorts, requiring an attribute such as HEAD to take values of sort *head*, whose subsorts are *nhead*, *vhead*, *ahead* and *phead* (corresponding to nouns, verbs, adjective and prepositions, respectively), amounts to an exhaustive disjunctive requirement on the HEAD attribute. Conversely, the appropriate values of an attribute can be seen as playing a significant role in determining the subsort relation and motivating various sorts.

As mentioned above, the notion of appropriateness required by HPSG is one where appropriate values for attributes are determined by the sort of the feature structure containing the attribute, rather than the attribute itself. The sort *head*, for example, requires the attribute MAJ to take values of sort *major*, whereas the sort *nhead* requires the attribute MAJ to take values of sort *N* (see [Pollard & Sag 87, p. 200]). Thus, the sort of the value of an attribute *is* dependent on the sort of the feature structure in which that attribute occurs.

2.3 Tokens, Types and Feature Structures

In CPSG, we see a slightly different relationship between the objects of the empirical domain and their descriptions. Basically, we adopt a token/type distinction, associating (unsorted) feature structures with both tokens and types. The notion of *sort* in HPSG is effectively replaced by a notion of *type*, in the token/type sense, in CPSG. The use of feature structures in CPSG is thus as depicted in Figure 1.4.

Note that the typing of tokens is prior to the use of feature structures to model the tokens. Nevertheless, feature structure descriptions may be engineered in such a way as to mediate the *is of type* relation. This is the approach we adopt in section 4, where a token is an instance of a type if and only if the description of that token extends the description of the type. The claim is that whilst sorted feature structures may be used to describe linguistic tokens, this is only as a consequence of the typing of tokens, which is prior to any notion of feature structure description. All of this is independent of the

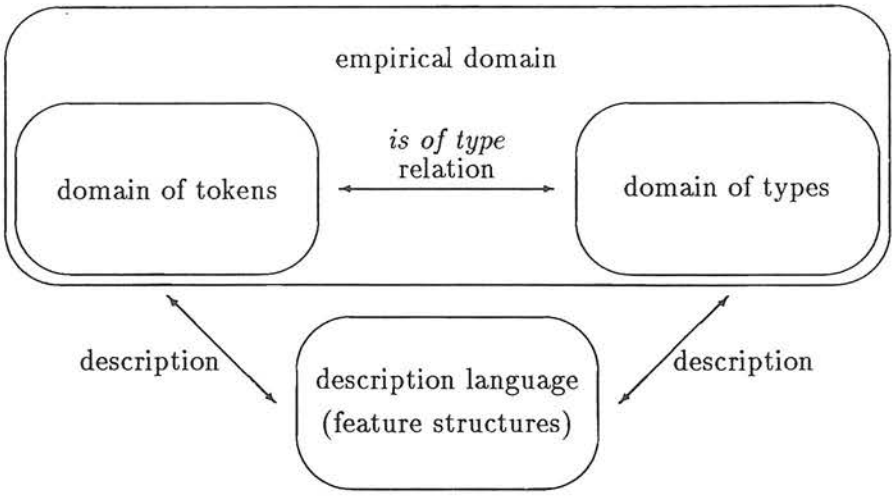


Figure 1.4: The Token/Type Distinction in CPSG

modelling and description relations, which relate the empirical and description domains to the model. The CPSG modelling domain, which is not depicted in Figure 1.4, is very similar to that of HPSG, being based on a domain of finite state automata.

In practical terms, sorted feature structures and typed tokens lead to a very similar formulation. Given a feature structure that describes a token, that token will be of a various types, which may be more or less equated with the sorts of the feature structure. The one major difference concerns embedded feature structures. In the sorted system of HPSG, such feature structures are also sorted. In a system based on tokens and types, embedded feature structures do not describe tokens of the empirical domain, and hence cannot have a type associated with them in the same way as other feature structures. The correspondence thus breaks down. In this way, the formulation of CPSG is slightly weaker than that of HPSG: sortal restrictions cannot be made on the values of various attributes. This restriction does not interfere with what is required by the formal machinery, as, as stated above, sorts in HPSG are mostly employed for abbreviatory purposes. Furthermore, the use of relational dependencies by CPSG (which are really no more powerful than the functional dependencies of HPSG) allows all the power of HPSG's sort system.

The use of an unsorted system of feature structures means that, in principle, any feature structure may be defined on any attributes: the necessary and sufficient subsumption

relationship that holds between the description of a type t and the description of a token s for any tokens s of type t will be satisfied provided the description of s is *any* extension of the description of t. In practice, however, only certain attributes will occur within certain feature structures. Though this is not essential to our analyses, we use this to impose a weak notion of sort on the domain of feature structures. This notion is in many ways independent of our notion of type, and may be seen as an implementation issue. The precise notion of sort employed is one where with each sort there is associated a disjoint set of attributes. No other conditions on sorts are required. In particular, the possible values of attributes are not restricted. Thus, a given attribute identifies the sort of feature structure it belongs to, but may take as its value any feature structure (or atomic value, or set, or list) in the domain. This notion of sort leads to numerous computational simplifications, including the use of term unification rather than graph unification, and the use of paths to create "frames" for the feature structures that they refer to. These computational simplifications are discussed in full in section 3 of Chapter 9.

The above is mentioned here because it leads essentially to a system based on "closed" sorts. In particular, every feature structure defined on the attribute PHONOLOGY, must also be defined on the attributes SYNTAX, SEMANTICS and DAUGHTERS. Similarly, every feature structure defined on the attribute SYNTAX must also be defined on the attributes PHONOLOGY, SEMANTICS and DAUGHTERS, and so on. To allow for tokens which should be described by feature structures defined on, for example, SYNTAX but which have no daughters, we employ the distinguished value *null*, which essentially amounts to stating that there is no valid value that this attribute can take. The value *null* does not correspond to some notion of "being undefined", rather it might be read as "is undefinable". Thus a lexical constituent will be described by a feature structure containing the specification [DAUGHTERS *null*], whereas in the case of a phrasal constituent the value of the DAUGHTERS attribute will be a (complex) feature structure.

3 Classification and Law-Like Dependencies

Central to linguistics is the notion of grouping together into classes words or phrases that share common properties. Classes such as noun phrase, verb phrase and sentence,

for example, which may be discriminated by distributional and semantic properties, are prominent in most, if not all, linguistic theorising. Also central are the relationships between members of such classes, relationships such as "if n is a proper noun then n is also a noun phrase", or "if n is a noun phrase and v a verb phrase then the string formed by concatenating n and v is a sentence".

In this section, we develop a formalism based on the distinctions of the previous section between tokens, types and feature structures, adopting as our point of departure Seligman's notion of an *abstract classification* ([Seligman 90a]). In applying Seligman's abstract classifications to linguistic domains, we augment his framework by considering domains of *structured tokens*, and employ feature structure descriptions to mediate the relationship between tokens and types. Having developed the formal apparatus, we then consider in more detail its application to linguistic domains, with two examples.

3.1 Abstract Classification

In developing a model of information transfer, [Seligman 90a] begins with the notion of an *abstract classification.* In an abstract classification, *tokens* are classified as being instances of various *types* according to a binary *is of type* relation. Correlations may hold between various tokens of various types due to law-like dependencies which the classification respects, and it is these law-like dependencies which allow inference: if an agent knows that the token n is a noun phrase and the token v is a verb phrase, and that agent is attuned to the relevant law-like dependency, then the agent may infer that the composite token (string) comprised of n followed by v is a sentence.

Following [Seligman 90a], an *abstract classification* is a triple $\langle S, T, : \rangle$, where

- S is a set of atoms known as tokens,
- T is a set of atoms known as types, and
- $:$ is a relation on $[S \times T]$.

A token s is of type t if and only if $s : t$.

Within an abstract classification, the *is of type* relation is primitive: it does not reflect any structure intrinsic to the domain of tokens. As far as the classification is concerned,

the only structure on the domain of tokens is that arising from the *is of type* relation itself.

3.2 Perspectives

[Seligman 90a, 90b] employs the notion of abstract classification in the development of *perspectives*. A perspective is intended to model the flow of information: how information 'about' one 'thing' can lead to information 'about' another 'thing'. Seligman does this by supplementing an abstract classification with two primitive binary relations between types, *involves* (written $\Rightarrow$) and *precludes* (written $\perp$), the idea being that if the type t_1 involves the type t_2, and there is a token s of type t_1, then there exists a token s' of type t_2, and if there is a token s of type t_1 and t_1 precludes t_2, then s is not of type t_2. In terms of the above, the tokens of the classification are the 'things' and the *is of type* relation expresses information about those things. Consequently the binary relations on types allow information about one thing to lead to information about another. The involves relation is taken to be transitive and the precludes relation is taken to be symmetric. The structure of a perspective is deliberately minimal.

A perspective is thus a tuple $\langle S, T, :, \Rightarrow, \perp \rangle$ where $\langle S, T, : \rangle$ is an abstract classification and $\Rightarrow$ and $\perp$ are binary relations on T such that for all $s \in S$ and $t \in T$,

- if $s : t$ and $t \Rightarrow t'$ then $\exists s' \in S$ such that $s' : t'$
- if $t \Rightarrow t'$ and $t' \Rightarrow t''$ then $t \Rightarrow t''$
- if $s : t$ and $t \perp t'$ then it is not the case that $s : t'$
- if $t \perp t'$ then $t' \perp t$

Seligman considers various special perspectives which satisfy further requirements. In particular, we are interested in those perspectives which satisfy *strong facticity*:

- if $s : t$ and $t \Rightarrow t'$ then $s : t'$

3.3 Structured Tokens

One way of looking at the relations *involves* and *precludes* is that they capture law-like dependencies within the classification: given the types noun and pronoun in a perspective

satisfying strong facticity, "pronoun *involves* noun" captures the law-like dependency that all pronouns are nouns, which any linguistic classification should respect. These primitive relations between types do not, however, allow us to capture relationships such as those between a constituent and its subconstituents. For this, we need to appeal to independent structure on the domain of tokens and the fact that the *is of type* relation in an abstract classification is primitive does not preclude such structure.

Given a domain of tokens where each token is a sequence of words, there are clearly many relationships between tokens. For example, one token may be a sub-sequence of another, or one token may be the concatenation of two other tokens. This structure may be relevant to law-like dependencies. In a linguistic classification, if one token s_1 is a (token of type) noun phrase (with agreement features x) and a token s_2 is a (token of type) verb phrase (also with agreement features x) then, provided their concatenation is also a token, it will be a (token of type) sentence. This kind of dependency cannot be captured solely in terms of *involves* and *precludes* because such relations are relations between types, and do not involve structured tokens. The two law-like dependencies which we have mentioned above might be stated as:

$$v : \text{proper noun} \Rightarrow v : \text{noun phrase}$$

$$v_1 : \text{noun phrase} \wedge v_2 : \text{verb phrase} \Rightarrow \mathit{concat}(v_1, v_2) : \text{sentence}$$

All variables in such statements range over the set S of tokens. In section 3.5 a full syntax and semantics is given for a language in which such law-like dependencies may be expressed.

3.4 Two Linguistic Classifications

As a first linguistic example of a classification consider the triple $\langle S, T, : \rangle$, where

- $S = \{$"Tigger", "Fido", "miaows", "chases Tom", "Tigger miaows", "Fido miaows", "Tigger chases Tom", "Fido chases Tom"$\}$,
- $T = \{$noun phrase, verb phrase, sentence$\}$,
- the *is of type* relation holds only of the following pairs:

"Tigger" : noun phrase	"Fido" : noun phrase
"miaows" : verb phrase	"chases Tom" : verb phrase
"Tigger miaows" : sentence	"Tigger chases Tom" : sentence
"Fido miaows" : sentence	"Fido chases Tom" : sentence

The second of the law-like dependencies mentioned above holds within this classification. In particular, the grammar rule

$$\textsf{sentence} \rightarrow \textsf{noun phrase} \quad \textsf{verb phrase}$$

corresponds to this dependency.

A different sort of law-like dependency is illustrated by the classification $\langle S', T', :\rangle$, where

- $S' = \{$"chase", "miaow", "give", "chasing", "miaowing", "giving"$\}$,
- $T' = \{$base, present participle, intransitive, transitive, ditransitive$\}$,
- the *is of type* relation holds only of the following pairs:

"chase" : base	"chasing" : present participle
"miaow" : base	"miaowing" : present participle
"give" : base	"giving" : present participle
"chase" : intransitive	"chasing" : intransitive
"miaow" : transitive	"miaowing" : transitive
"give" : ditransitive	"giving" : ditransitive

Two interesting law-like dependencies hold of the types of this classification. Firstly, if s is a token of type base, then $prp(s)$ is a token of type present participle, and *vice versa*, where prp maps (instances of) base form verbs to (instances of) their present participle form, and secondly, if s is a token of type intransitive/transitive/ditransitive, then so is $prp(s)$, and *vice versa*. Assuming that the function prp has an inverse, we may state these dependencies as:

$$v : \textsf{base} \Rightarrow prp(v) : \textsf{present participle}$$

$$v : \textsf{present participle} \Rightarrow prp^{-1}(v) : \textsf{base}$$

$$v : \textsf{intransitive} \Rightarrow prp(v) : \textsf{intransitive}$$

$$v : \textsf{intransitive} \Rightarrow prp^{-1}(v) : \textsf{intransitive}$$

$$v : \text{transitive} \Rightarrow prp(v) : \text{transitive}$$

$$v : \text{transitive} \Rightarrow prp^{-1}(v) : \text{transitive}$$

$$v : \text{ditransitive} \Rightarrow prp(v) : \text{ditransitive}$$

$$v : \text{ditransitive} \Rightarrow prp^{-1}(v) : \text{ditransitive}$$

What is of interest here is that what would normally be termed a lexical rule corresponds to a law-like dependency within the classification.

3.5 Law-Like Dependencies

As mentioned above, [Seligman 90a, 90b] treats law-like dependencies in terms of his framework of *perspectives*, which only allows for dependencies based on *involvement* and *preclusion*. Such dependencies are insufficient when dealing with structured tokens, as in linguistic domains, and consequently the framework of *perspectives* is too restrictive for our purposes. We thus present here a syntax and semantics for law-like dependencies which is sufficient for our linguistic domains. We do not present the inference rules necessary to transform this language into a full logic.

Note that within a classification, correlations between tokens of various types over and above law-like dependencies may exist. Seligman distinguishes law-like dependencies from "accidental correlations". We say more about this intensional distinction when we consider *classificatory systems* in section 4.

3.5.1 Syntax

Given a set V of atoms known as variables, a set F of sets F_n of function symbols for each $n \in N$, and a set T of types, we define two sets, $\textsc{Prop}(V,T)$ and $\textsc{FnProp}(V,F,T)$.

$\textsc{Prop}(V,T)$ is the smallest set such that:

if $v \in V$, $t \in T$ then $v : t \in \textsc{Prop}(V,T)$,

if $\phi_1, \phi_2 \in \textsc{Prop}(V,T)$ then $(\phi_1 \wedge \phi_2) \in \textsc{Prop}(V,T)$,

if $\phi_1, \phi_2 \in \textsc{Prop}(V,T)$ then $(\phi_1 \vee \phi_2) \in \textsc{Prop}(V,T)$, and

if $\phi \in \textsc{Prop}(V,T)$ then $\neg\phi \in \textsc{Prop}(V,T)$.

$\textsc{FnProp}(V,F,T)$ is the smallest set such that:

if $f \in F_n$, $v_1,\ldots,v_n \in V$, and $t \in T$, then $f(v_1,\ldots,v_n) : t \in \textsc{FnProp}(V,F,T)$,

if $\psi_1,\psi_2 \in \textsc{FnProp}(V,F,T)$ then $(\psi_1 \wedge \psi_2) \in \textsc{FnProp}(V,F,T)$,

if $\psi_1,\psi_2 \in \textsc{FnProp}(V,F,T)$ then $(\psi_1 \vee \psi_2) \in \textsc{FnProp}(V,F,T)$, and

if $\psi \in \textsc{FnProp}(V,F,T)$ then $\neg\psi \in \textsc{FnProp}(V,F,T)$.

Given such sets, the set $\mathrm{LLD}(V,F,T)$ of *possible law-like dependencies* is the set of all statements of the form:

$$\phi \Rightarrow \psi$$

where $\phi \in \textsc{Prop}(V,T)$ and $\psi \in \textsc{FnProp}(V,F,T)$

3.5.2 Semantics

An interpretation consists of a total assignment of partial functions in $[S^n \rightarrow S]$ to each of the function symbols in each set F_n. Given an interpretation h, a law-like dependency $\phi \Rightarrow \psi \in \mathrm{LLD}(V,F,T)$ holds of the classification $\langle S,T,:\rangle$ only if for all assignments $g : [V \rightarrow S]$ of tokens to variables such that $[\![\phi]\!]_g = 1$, it is also the case that $[\![\psi]\!]_g = 1$, where $[\![\cdot]\!]$. is such that:

$$[\![v : t]\!]_g = \begin{cases} 1 & \text{if } g(v) : t \text{ in } \langle S,T,:\rangle \\ 0 & \text{otherwise} \end{cases}$$

$$[\![f(v_1,\ldots,v_n) : t]\!]_g = \begin{cases} 1 & \text{if } h(f)(g(v_1),\ldots,g(v_n)) : t \text{ in } \langle S,T,:\rangle \\ 1 & \text{if } h(f)(g(v_1),\ldots,g(v_n)) \text{ is undefined} \\ 0 & \text{otherwise} \end{cases}$$

$$[\![(\chi_1 \wedge \chi_2)]\!]_g = \begin{cases} 1 & \text{if } [\![\chi_1]\!]_g = 1 \text{ and } [\![\chi_2]\!]_g = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$[\![(\chi_1 \vee \chi_2)]\!]_g = \begin{cases} 1 & \text{if } [\![\chi_1]\!]_g = 1 \text{ or } [\![\chi_2]\!]_g = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$[\![\neg\chi]\!]_g = \begin{cases} 1 & \text{if } [\![\chi]\!]_g = 0 \\ 0 & \text{otherwise} \end{cases}$$

Notice that a law-like dependency may involve a partial function in its consequent, in which case assignments of tokens to variables for which the function is undefined do not interfere with the classification's respect of the law-like dependency.

Note also that in place of involves, which is a binary relation between types, we use a binary relation between propositions. By including tokens within these propositions, we can capture preclusion in terms of negation: $v : t_1 \Rightarrow \neg v : t_2$ is equivalent to $t_1 \perp t_2$. Furthermore, in the case of involvement we can be more specific, formulating a dependency which says not just that if there is a token of type t_1 and t_1 involves t_2 then there is *some* token of type t_2, but that if v is a token of type t_1 then $f(v)$, for some known function f, is of type t_2. That is, we can identify the relevant token in the consequent of the dependency. The function may be the identity function, in which case we have the equivalent of Seligman's strong facticity.

4 Classificatory Systems

Since their introduction by [Kay 79], feature structures have been used prominently by many formal/computational linguistic theories. In this section, we follow this general use and augment Seligman's notion of an abstract classification with feature structure descriptions of tokens and types. The resulting systems, which we term *classificatory systems*, capture the distinction between accidental correlations and law-like dependencies and allow the classification of tokens to proceed on the basis of their descriptions. Essentially this is achieved by associating feature structure descriptions with tokens and types and allowing these descriptions to mediate the *is of type* relation.

4.1 Tokens and Descriptions

Given a domain of tokens and a set of feature structures, the tokens may be *described* by the feature structures. For example, in a classification of cats, the cat (token) named Tom might be described by:

$$\begin{bmatrix} \text{NAME} & \textit{Tom} \\ \text{COLOUR} & \textit{black} \\ \text{AGE} & \textit{4 years} \\ \text{SEX} & \textit{male} \\ \text{HEIGHT} & \textit{35cm} \\ \text{WEIGHT} & \textit{3.9kg} \end{bmatrix}$$

Intuitively, given fairly standard assumptions about the set of feature structures (that the elements are partially ordered by a subsumption relation, that an operation of unification corresponding to the least upper bound of the partial order is defined on subsets of the set, and that the set is closed under this operation of unification), there is a unique "most complete" description for each token: if f_1 and f_2 (partially) describe some token, then their unification also describes that token, and furthermore that description is at least as complete as each of f_1 and f_2. Here, the notion of relative "completeness" is given by the subsumption ordering on the feature structures. If f_1 subsumes f_2 then f_2 is more complete than f_1.

That we require a unique most complete description for each token means that the description relation may be treated as functional. A classification may thus be augmented by an assignment δ of feature structures to its tokens: if s is a token then $\delta(s)$ is a description of s. Any feature structure which subsumes $\delta(s)$ will be said to *partially* describe s.

This notion of description does not require the tokens to be classified. We may thus formalise the above independently of a classification in terms of a *domain of described tokens*, $\langle S, \mathcal{F}, \delta \rangle$, where

- S is a set of atoms known as tokens,
- $\mathcal{F}$ is a set of feature structures, and
- $\delta : [S \rightarrow \mathcal{F}]$.

Note that because the domain of feature structure descriptions is structured (by the subsumption ordering), structure is indirectly imposed on the domain of tokens. According to Seligman's view of a classification any such structure is derivative — law-like dependencies impose structure on the domain of types which is reflected in the domain of tokens via the *is of type* relation. In associating feature structure descriptions with tokens we neither accept nor reject this view.

There is more that we can say about the description relation than merely that it is functional. The relation may be many-to-one: within a system of feature structures, if the system does not include any attributes which distinguish some tokens, those tokens

will be described by identical feature structures. Consequently, a description of a token need not uniquely identify that token. This may be used to advantage as, for example, it is not normally necessary to distinguish each instance of a string — only strings with different phonological and syntactic properties need be distinguished. Because of the system of attributes and values employed by HPSG, for example, that theory assigns identical descriptions to strings with the same phonological and syntactic properties but different physical properties (such as space-time location, or pitch of utterance), but different descriptions to strings with different phonological or syntactic properties. In Bromberger's terms ([Bromberger 88]), feature structure descriptions of tokens are in one-to-one correspondence with *archetypes* — abstractions over tokens which share all features relevant to the description. The operation of feature structure unification is a test for archetype identity, not token identity.

4.2 Types and Descriptions

The types of a classification are normally motivated by their participation in law-like dependencies. The types noun phrase, verb phrase and sentence from above, for example, are motivated by the existence of the law-like dependency between them. Typically, all tokens of a type share common properties as a consequence of the law-like dependencies that the type participates in. Again in the example above, all tokens of type verb phrase share the property of combining with a token of type noun phrase to yield a token of type sentence. These common properties may be captured by an appropriate system of attributes and values. Within HPSG, for example, all tokens of type verb phrase may be (partially) described by the attribute-value pair [SUBCAT ⟨NP⟩]. The use of feature structure descriptions may thus be extended to types.

Paralleling the use of a domain of described tokens in the previous section, we thus introduce the notion of a *domain of described types*. A domain of described types is a tuple $\langle T, \mathcal{F}, \Delta \rangle$, where

- T is a set of atoms known as types,
- $\mathcal{F}$ is a set of feature structures, and
- $\Delta : [T \rightarrow \mathcal{F}]$.

It should be clear from the above that if we have a domain of described tokens $\langle S, \mathcal{F}, \delta \rangle$, a domain of describe types $\langle T, \mathcal{F}, \Delta \rangle$, and an abstract classification $\langle S, T, : \rangle$, then for each type t of the classification, $\Delta(t)$ must be such that if s is of type t then $\Delta(t)$ is a partial description of s, and so subsumes $\delta(s)$. That is, if a token is of a type then it must have all features associated with that type:

$$\text{if } s : t \text{ then } \Delta(t) \sqsubseteq \delta(s)$$

Given what we have said about classifications, however, the converse need not hold: if a token has all the features associated with a type it does not follow that the token is an instance of that type.

If a token is of several types, the descriptions of those types must unify, and the unification of those descriptions will also be a description of the token. One might argue that *all* features of a token are a consequence of the token being of types which have those features. That is, it might be argued that all of a token's features are inherited from the types of which it is an instance. We do not require this to be so: the description of a token might include features over and above those inherited from the types of which it is an instance.

Given a classification $\langle S, T, : \rangle$ and a domain of described tokens $\langle S, \mathcal{F}, \delta \rangle$, we might derive a domain of described types $\langle T, \mathcal{F}, \Delta \rangle$, by defining Δ in purely extensional terms as a greatest lower bound:

$$\Delta(t) \stackrel{\text{def}}{=} \bigvee_{\{s \mid s:t\}} \delta(s)$$

Such a Δ will clearly satisfy:

$$\text{if } s : t \text{ then } \Delta(t) \sqsubseteq \delta(s)$$

However, this definition does not capture the intensional distinction between "essential" properties and "accidental" properties of types — a distinction parallel to that between law-like dependencies and accidental correlations. Within some linguistic classification, for example, the following feature structures might describe the only two tokens of type verb phrase:

$$\begin{bmatrix} \text{PHONOLOGY} & \textit{likes Tom} \\ \text{SUBCAT} & \langle \text{NP} \rangle \\ \text{DTRS} & \begin{bmatrix} \text{HEAD} & \textit{likes} \\ \text{COMP} & \textit{Tom} \end{bmatrix} \end{bmatrix} \qquad \begin{bmatrix} \text{PHONOLOGY} & \textit{chases Tom} \\ \text{SUBCAT} & \langle \text{NP} \rangle \\ \text{DTRS} & \begin{bmatrix} \text{HEAD} & \textit{chases} \\ \text{COMP} & \textit{Tom} \end{bmatrix} \end{bmatrix}$$

We would probably not wish to say that the greatest lower bound of these feature structures describes the type **verb phrase**, as such a feature structure will include the feature [DTRS|COMP *Tom*]. A preferable description would be:

$$\begin{bmatrix} \text{SUBCAT} & \langle \text{NP} \rangle \end{bmatrix}$$

Seligman's presentation of classifications also argues against the above extensional definition of Δ. If Δ is to allow us to distinguish between law-like dependencies and accidental correlations (as we intend it to), then it cannot be defined in purely extensional terms. Furthermore, it would seem that according to [Seligman 90a], Δ is prior to δ: tokens may only be seen to have properties/features by being instances of types, and so if anything, δ should be defined in terms of Δ. Seligman might thus prefer to start with a classification $\langle S, T, : \rangle$ and a domain of described types $\langle T, \mathcal{F}, \Delta \rangle$ and then derive a domain of described tokens $\langle S, \mathcal{F}, \delta \rangle$ where

$$\delta(s) \stackrel{\text{def}}{=} \bigwedge_{\{t \mid s:t\}} \Delta(t)$$

[Seligman 90b] also introduces feature structures but avoids this question of priority by treating the features of feature structures as elements of another domain of types. Tokens are then cross-classified according to two different sets of types. He then considers relations between the resultant classifications.

We also avoid making a commitment to any notion of priority and simply require that in such a complex of systems

$$\forall t \in T, \quad \Delta(t) \sqsubseteq \bigvee_{\{s \mid s:t\}} \delta(s)$$

or equivalently

$$\forall s \in S, \quad \delta(s) \sqsupseteq \bigwedge_{\{t \mid s:t\}} \Delta(t)$$

These are the minimum requirements such that if $s : t$ then $\Delta(t) \sqsubseteq \delta(s)$.

The utility of the partial descriptions afforded by types may be illustrated by an HPSG-like theory which classifies tokens such as "chases Tom" as being of type **headed phrase**. Within this theory, the description of the type **headed phrase** might be given by:

$$\Delta(\textbf{headed phrase}) = \begin{bmatrix} \text{SYN|LOC|HEAD} & \boxed{1} \\ \text{DTRS|HEAD-DTR|SYN|LOC|HEAD} & \boxed{1} \end{bmatrix}$$

From this, and the fact that "chases Tom" : **headed phrase**, we may deduce that "chases Tom" is partially described by:

$$\begin{bmatrix} \text{SYN|LOC|HEAD} & \boxed{1} \\ \text{DTRS|HEAD-DTR|SYN|LOC|HEAD} & \boxed{1} \end{bmatrix}$$

This is essentially a restatement of HPSG's Head Feature Principle within a classification.

4.3 Classificatory Systems

As mentioned above, we cannot in general infer from that fact that a token has all the features associated with a type that that token is an instance of that type. That is, we cannot infer from $\Delta(t) \sqsubseteq \delta(s)$ that $s : t$. However, it is worth considering the restricted complexes of domains of described tokens, domains of described types and abstract classifications in which this inference does hold. In such cases, all tokens whose descriptions are subsumed by the description of a type will be instances of that type, and inferences can be made based on feature structure descriptions of tokens: knowing how a token is described allows us to correctly classify it as being of various types. This leads to the notion of a *classificatory system*, whereby tokens may be classified as being of various types according to their feature structure descriptions.

A domain of described types $\langle T, \mathcal{F}, \Delta \rangle$ classifies the tokens in any domain of described tokens $\langle S, \mathcal{F}, \delta \rangle$ with ':' defined as:

$$s : t \text{ iff } \Delta(t) \sqsubseteq \delta(s)$$

When a domain of described types is viewed in this light we refer to it as a *classificatory system*. A classificatory system classifies the tokens in any domain of described types which is based on the same set $\mathcal{F}$ of feature structure descriptions. Whilst a classificatory system classifies tokens, its definition is independent of those tokens. For a token to be classified by a classificatory system it must be described by a feature structure from the set on which the system is based. In contrast to the abstract classifications of section 3.1,

where the *is of type* relation is primitive, a classificatory system leads to a classification where the *is of type* relation is derivative: it is derived from descriptions of tokens and types (i.e., from the functions δ and Δ).

Clearly within a classificatory system Δ cannot be given an extensional definition in terms of descriptions of tokens as there is no predefined set of tokens which the system classifies. Furthermore, accidental correlations within a classificatory system cannot arise — any correlation that arises when a classificatory system classifies some domain of described tokens $\langle S, \mathcal{F}, \delta \rangle$ which is not due to a law-like dependency may be violated by some alternate domain of described tokens $\langle S', \mathcal{F}, \delta' \rangle$. Treating classifications in terms of classificatory systems and domains of described tokens thus allows the intensional distinction between law-like dependencies and accidental correlations to be captured.

4.4 Law-Like Dependencies

A classificatory system is said to respect a law-like dependency if and only if all classifications which it may give rise to (in the above sense) respect that law-like dependency. Recall that the functions involved in law-like dependencies need not be total. Partial functions allow exceptions to such dependencies to be admitted. Thus, although law-like dependencies were introduced as involving functions on the domain of tokens, there is no problem in associating law-like dependencies with classificatory systems where there is no explicit domain of tokens. The classification that arises from a classificatory system associated with an instance of a domain of described tokens can still be said to respect a law-like dependency provided all tuples for which the relevant functions are defined satisfy the dependency.

Now, given a dependency of the form:

$$v_1 : t_1 \ \wedge \ \ldots \ \wedge \ v_n : t_n \ \Rightarrow\!\!\!\triangleright \ f_t(v_1, \ldots, v_n) : t$$

we may infer, for suitable $\langle s_1 \ldots s_n \rangle$, that $f_t(s_1, \ldots, s_n) : t$, and thus that $f_t(s_1, \ldots, s_n)$ is partially described by $\Delta(t)$. That is,

$$\delta(f_t(s_1, \ldots, s_n)) \sqsupseteq \Delta(t)$$

Whilst this allows something to be deduced about the description of $f_t(s_1, \ldots, s_n)$, what can be deduced is fairly minimal. In particular, this description is not dependent on

the descriptions of the tokens $\langle s_1 \ldots s_n \rangle$. We may remedy this by associating with each function f_t mapping n-tuples of tokens to single tokens, a function f_{fs} mapping n-tuples of feature-structure descriptions to single feature structure descriptions such that

$$\delta(f_t(s_1, \ldots, s_n)) \sqsupseteq f_{fs}(\delta(s_1), \ldots, \delta(s_n))$$

This is always trivially possible, as illustrated by the constant functions mapping all tuples from $\mathcal{F}^n$ to $[\ ]$.

To demonstrate the utility of such functions in a non-trivial case, consider the classificatory system $\langle T, \mathcal{F}, \Delta \rangle$ where

- T is the set {noun phrase, verb phrase, sentence},
- $\mathcal{F}$ is the set of feature structures with attributes from {CAT, SUBJ, PRED, PHON} and atomic values from {*NP*, *VP*, *S*, *Tigger*, *miaows*, *Tigger miaows*, ...}.
- Δ is defined as:

$$\Delta(\text{noun phrase}) = \begin{bmatrix} \text{CAT} & NP \end{bmatrix}$$

$$\Delta(\text{verb phrase}) = \begin{bmatrix} \text{CAT} & VP \end{bmatrix}$$

$$\Delta(\text{sentence}) = \begin{bmatrix} \text{CAT} & S \\ \text{SUBJ} & \begin{bmatrix} \text{CAT} & NP \end{bmatrix} \\ \text{PRED} & \begin{bmatrix} \text{CAT} & VP \end{bmatrix} \end{bmatrix}$$

The classificatory system respects the law-like dependency

$$v_1 : \text{np} \ \wedge \ v_2 : \text{vp} \ \Rightarrow \ subj/pred_t(v_1, v_2) : \text{sentence}$$

where the function $subj/pred_t$ effectively concatenates its arguments. Correspondingly we have the function

$$subj/pred_{fs}(\boxed{1}, \boxed{2}) = \begin{bmatrix} \text{SUBJ} & \boxed{1} \\ \text{PRED} & \boxed{2} \end{bmatrix}$$

which maps ordered pairs of feature structures to feature structures such that

$$\delta(subj/pred_t(s_1, s_2)) \sqsupseteq subj/pred_{fs}(\delta(s_1), \delta(s_2))$$

Thus in stating law-like dependencies which a classificatory system respects, we also state the functions on the domain of token descriptions which correspond to those functions on the domain of tokens employed by the dependencies.

A classificatory system may necessarily respect certain law-like dependencies in virtue of the descriptions of the types involved. If, for example, a classificatory system includes two types t_1 and t_2 such that $\Delta(t_1) \sqsupseteq \Delta(t_2)$ then any token of type t_1 will necessarily also be of type t_2, so the classificatory system will respect the dependency:

$$v : t_1 \;\Rightarrow\; v : t_2$$

More generally, suppose that for some types $t, t_1, \ldots, t_n$ and some function $f_{fs} : [\mathcal{F}^n \rightharpoonup \mathcal{F}]$,

$$f_{fs}(\theta_1, \ldots, \theta_n) \sqsupseteq \Delta(t)$$

for all $\theta_1 \ldots \theta_n$ such that

$$\theta_1 \sqsupseteq \Delta(t_1)$$

$$\vdots$$

$$\theta_n \sqsupseteq \Delta(t_n)$$

Then for any tokens $s_1, \ldots, s_n$ such that $s_1 : t_1, \ldots, s_n : t_n$, it will necessarily be the case that

$$f_{fs}(\delta(s_1), \ldots, \delta(s_n)) \sqsupseteq \Delta(t)$$

Thus if $f_{fs}(\delta(s_1), \ldots, \delta(s_n))$ is a partial description of some token s, then $s : t$.

Therefore if there is some corresponding function f_t mapping sequences of n tokens to another token such that

$$\delta(f_t(s_1, \ldots, s_n)) \sqsupseteq f_{fs}(\delta(s_1), \ldots, \delta(s_n))$$

then the classificatory system will necessarily respect the following dependency:

$$v_1 : t_1 \;\wedge\; \ldots \;\wedge\; v_n : t_n \;\Rightarrow\; f_t(v_1, \ldots, v_n) : t$$

5 Feature Structures

Many subtly different formulations of feature structures co-exist in the current literature. In order to be precise, it is thus important to specify exactly what we take feature structures to be: it is not sufficient to rely on the intuitive and common core of the many formulations in circulation.

We adopt a logical approach, treating feature-structures as *descriptions*, rather than *representations*. Thus several distinct feature structures may partially describe the same object. Partiality is effectively put at the level of syntax, rather than at the level of semantics: the semantic domain consists of objects which are complete in their own right, but which may be *partially described* by syntactic objects. [Halvorsen 87] discusses in detail this issue of description versus representation.

Feature structures themselves may be represented in various different ways, the two most common being as attribute-value matrices (AVMs) and as directed acyclic graphs (DAGs). For formatting ease, we employ the AVM representation throughout this thesis.

We distinguish between *attributes* and *features*. An attribute is a dimension along which the objects that we are describing may vary. Given a particular object and an attribute, that object may, or may not, have a particular value for that attribute. If the object does have a value for that attribute, the attribute is said to be *appropriate* for the object. A feature, on the other hand, is an atomic entity which an object either does or does not have: features do not take on various values. A feature is equivalent to an attribute-value pair.

5.1 Rounds-Kasper Logic

In a series of papers ([Rounds & Kasper 86], [Kasper & Rounds 86], [Moshier & Rounds 87], [Kasper & Rounds 90]), Rounds and Kasper and Moshier have developed an equational logic for feature structures. The syntactic domain of this logic consists of equations expressing relationships between attributes and values, with the semantic domain consisting of a class of deterministic finite state automata. Following this work, we present a similar equational system here, which we extend throughout this section. We do not present the accompanying calculus of formulae which Rounds and Kasper also develop. In all instances we work with respect to a set L of atoms known as *labels* and a set A of atoms known as *atomic values*.

5.1.1 The Syntactic Domain

Define the set $A' = A \cup \{\top, \bot\}$. The domain $\mathcal{LF}$ of logical formulae is the smallest set such that:

1. $a \in \mathcal{LF}$ if $a \in A'$,

2. $[l : \phi] \in \mathcal{LF}$ if $l \in L$ and $\phi \in \mathcal{LF}$,

3. $(\phi \vee \psi) \in \mathcal{LF}$ if $\phi, \psi \in \mathcal{LF}$,

4. $(\phi \wedge \psi) \in \mathcal{LF}$ if $\phi, \psi \in \mathcal{LF}$, and

5. $[\![p_1, \ldots, p_n]\!] \in \mathcal{LF}$ if each $p_i \in L^*$.

Clause 1 ensures that all atoms, including the distinguished atoms $\top$ and $\bot$, are in $\mathcal{LF}$. Clause 2 ensures that pairs consisting of a single element long path, which is interpreted as an attribute, and a logical formulae, which is interpreted as the value of the attribute, are in $\mathcal{LF}$. The inductive nature of the definition ensures that values may be associated with paths of any finite length. Clauses 3 and 4 ensure that finite disjunctions and conjunctions of formulae are in $\mathcal{LF}$. Clause 5 allows path equivalences to be stated: expressions of the form in 5 are interpreted as requiring that all paths within the expression are equivalent. Equivalent paths are often said to *share* the value to which they lead.

5.1.2 Equational Logic and Attribute Value Matrices

There is a fairly direct correspondence between the syntax of this equational logic and the attribute value matrix representation employed throughout this thesis. This is perhaps best illustrated by example. The following attribute value matrix:

$$\left[\begin{array}{ll} \textsc{phon} & \textit{kitten} \\ \textsc{syn} & \left[\textsc{loc}\ \bot\right] \vee \left[\textsc{bind}\ \bot\right] \end{array}\right] \vee \left[\begin{array}{ll} \textsc{syn} & \left[\textsc{loc}\ \left[\textsc{head}\ \boxed{1}\right]\right] \\ \textsc{head} & \boxed{1} \end{array}\right]$$

corresponds to the following formula of equational logic:

$$(([\textsc{phon}{:}\textit{kitten}] \wedge [\textsc{syn}{:}([\textsc{loc}{:}\bot] \vee [\textsc{bind}{:}\bot])]) \vee [\![\langle \textsc{syn}, \textsc{loc}, \textsc{head}\rangle, \langle \textsc{head}\rangle]\!])$$

This correspondence between AVMs and formulae might be formally defined.

Note that path equivalences are indicated in AVM terms via "tags". The equivalence $[\![p_1, \ldots, p_n]\!]$ is represented in AVM terms as if by the equation

$$p_1 : \boxed{i} \wedge \ldots \wedge p_n : \boxed{i}$$

where $\boxed{i}$ is an otherwise unused tag

5.1.3 The Semantic Domain

A deterministic finite state automaton (DFA) may be given formally in terms of a tuple $\langle Q, q_0, L, \delta, A, \pi \rangle$, where

- Q is a finite set of atoms known as states,
- $q_0 \in Q$ is a distinguished state known as the start state,
- L is a finite set of atoms known as labels or attributes,
- $\delta : [Q \times L \rightharpoonup Q]$ is a state transition function,
- A is a set of atoms known as atomic values, and
- $\pi : [Q \rightharpoonup A]$ is a partial function assigning atomic values to terminal states. A terminal state is a state $q \in Q$ for which for all $l \in L$, $\delta(q, l)$ is undefined.

We also require that the DFA be connected, i.e., that for each $q \in Q$, there is some path $\langle l_0, \ldots, l_n \rangle \in L^*$ such that $\delta^*(q_0, \langle l_0, \ldots, l_n \rangle) = q$, where

$$\delta^*(q, \langle\ \rangle) = q, \text{ and}$$

$$\delta^*(q, \langle l_0, \ldots, l_n \rangle) = \delta^*(\delta(q, l_0), \langle l_1, \ldots, l_n \rangle).$$

5.1.4 Satisfiability

Satisfiability is defined inductively paralleling the clauses for syntactic formation of formulae. Given a DFA $\mathcal{A} = \langle Q, q_0, L, \delta, A, \pi \rangle$:

1. - $\mathcal{A} \models \top$ never.
   - $\mathcal{A} \models \bot$ always.
   - $\mathcal{A} \models a$ where $a \in A$ iff $Q = \{q_0\}$, π is defined on q_0 (so $\delta(q_0, l)$ is undefined for each $l \in L$), and $\pi(q_0) = a$.
2. $\mathcal{A} \models [l : \phi]$ iff $\mathcal{A}/l$ is defined and $\mathcal{A}/l \models \phi$, where $\mathcal{A}/l$ is defined iff $\delta(l, q_0)$ is defined, in which case $\mathcal{A}/l = \langle Q', \delta(q_0, l), L, \delta, A, \pi \rangle$ where Q' is formed from Q by removing any states which are unreachable from $\delta(q_0, l)$.
3. $\mathcal{A} \models (\phi \vee \psi)$ iff $\mathcal{A} \models \phi$ or $\mathcal{A} \models \psi$.

4. $\mathcal{A} \models (\phi \wedge \psi)$ iff $\mathcal{A} \models \phi$ and $\mathcal{A} \models \psi$.

5. $\mathcal{A} \models [\![p_1, \ldots, p_n]\!]$ iff for each $p_i \in \{p_1, \ldots, p_n\}$, $\delta^*(q_0, p_i)$ is defined and constant (i.e., all paths in the equivalence lead to the same state $q = \delta^*(q_0, p_i)$).

5.2 Subsumption and Unification

A partial order may be defined on the domain of logical formulae in terms of the models which satisfy the formulae. The subsumption relation, $\sqsubseteq$, may be defined as:

$$f_1 \sqsubseteq f_2 \text{ iff } \{\mathcal{M} \mid \mathcal{M} \models f_1\} \supseteq \{\mathcal{M} \mid \mathcal{M} \models f_2\}$$

That is, one formula subsumes another if and only if every model which satisfies the second also satisfies the first, i.e., the first picks out more models, or is less specific, than the second.

The subsumption ordering is reflexive (if $f_1 \sqsubseteq f_2$ then $f_2 \sqsubseteq f_1$), transitive (if $f_1 \sqsubseteq f_2$ and $f_2 \sqsubseteq f_3$ then $f_1 \sqsubseteq f_3$), and antisymmetric (if $f_1 \sqsubseteq f_2$ and $f_2 \sqsubseteq f_1$ then $f_1 = f_2$, where two formulae are equal if and only if they are satisfied by exactly the same models). It is, therefore, a partial order.

The unification of a set of formulae is defined to be the least upper bound of that set according to the subsumption ordering. Such a formula will be satisfied by (at most) only those models which satisfy every formula in the set. If there are no models which satisfy every formula, then the formulae are said to be incompatible and their unification is $\top$.

Unification in the equational, or feature structure, domain is not a procedural operation, and it must not be confused with Prolog unification, as employed in the implementation discussed in Chapter 9.

5.3 Closed Sorts

We have criticised the use of (open) sorts employed in HPSG as lacking clarity: it is not always clear precisely what sorts are required to achieve within HPSG. CPSG employs a system of closed sorts, primarily for computational efficiency, and here we consider the use of closed sorts within Rounds-Kasper logic. Informally, the notion of closed sort

which we require centres around co-occurring sets of attributes: the set of all attributes is partitioned into subsets of co-occurring attributes, and attributes from distinct sets of the partition must not co-occur.

We thus begin by partitioning the set of labels: let $\mathcal{L} = \{L_i \mid L_i \subseteq L\}$ be a partition of L. Each element of the partition is termed a *sort*. The syntax of the sorted system is as before except that there is an extra value, *null*, that an attribute may take. This distinguished value marks attributes within a sort which, in a particular feature structure, are not defined.

The semantics is also pretty much as before except we restrict attention to those DFAs for which for all $q \in Q$, the set $\{l \mid \delta(q,l) \text{ is defined}\}$ is either empty or a subset of some sort. That is, those DFAs such that if a transition is defined from some state $q \in Q$ and label $l \in L_i \in \mathcal{L}$, then transitions are defined from q for at most some subset of those attributes in the sort L_i, and in particular, transitions are not defined from the same node for attributes from distinct sorts.

For satisfiability we have one extra clause for the case $[l : null]$:

2′. $\mathcal{A} \models [l : null]$ iff $\delta(q_0, l)$ is undefined.

This may be compared with the clause for $\bot$:

$\mathcal{A} \models \bot$ always.

There is thus a significant difference between an equation of the form $[l : \bot]$ and one of the form $[l : null]$. The first is satisfied by any DFA with a root node having a transition arc labelled l. The second is only satisfied by DFAs which consist of a root node *not* having a transition arc labelled l. The value *null* thus means "is not definable", rather than "is undefined".

5.4 List and Set Valued Attributes

The above presentation of Rounds-Kasper logic is not sufficiently expressive for either HPSG or CPSG: list and/or set valued attributes are required to describe, among other things, subcategorisation requirements. In this section we thus consider how the logic might be extended to admit such values. We concentrate initially on the semantic side,

before considering the trivial augmentation to the syntax of equational logic which is required to admit such values.

Lists may be simply introduced to the sorted system by adding one further sort and one further atomic value, the atomic value corresponding to the empty list and the sort corresponding to a non-empty list. This sort contains two labels, HEAD and TAIL (or some similar otherwise unused labels). Our system allows no way of applying the further constraint that this method requires: that the value of the TAIL should either be $\perp$, the empty-list, or a complex value of sort {HEAD, TAIL}. Furthermore, although this method does allow us to model lists without altering the logic, it does not easily generalise to sets, which may be viewed as unordered lists.

[Rounds 88] proposes that set-valued attributes may be introduced by dropping the deterministic constraint on automata which we have been assuming, i.e., by allowing non-deterministic automata. He does this by replacing the transition function with a transition *relation*: given an initial state q and a label l, the automaton may change to any one of several states, with the state changed to being chosen non-deterministically from the set of states related to l and q by the transition relation. More formally, a non-deterministic finite state automaton is a tuple $\langle Q, q_0, L, \delta, A, \pi \rangle$, where Q, q_0, L, A, and π are as before, and δ is a relation on $Q \times Q \times L$.

This treatment of set values has the advantage of being easily generalisable to list values: the possible transitions from a state according to a given label may be weighted, and hence ordered, allowing sets, posets, and lists to be accommodated in a unified manner. However, for the solution to yield values with many of the properties normally associated with sets, we must distinguish between transitions leading to the elements of a set and other transitions. Without this, there is, for example, no distinction between attributes whose values are singleton sets and other values, as each is modelled by a finite state machine with a transition leading to a single state. Similarly there is no distinction between the empty set and final states. [Rounds 88] effects this distinction by introducing "ϵ transitions", which are a new variety of transition leading exclusively to the elements of a set. This still doesn't help with empty sets, but in a sorted system they can be treated via a distinguished atom.

An alternative to Rounds' approach is to augment the conception of a finite state machine to include transitions to sets of nodes, as well as individual nodes. This approach allows determinism to be maintained, as well as distinguishing between normal transitions and singleton sets, and empty sets and terminal nodes. This is the solution which we adopt in providing a semantics for set valued attributes. We similarly allow transitions to lists of nodes, thus providing a semantics for list valued attributes.

Syntactically, sets are as in 6 and lists as in 7:

6. $\{\phi_i \mid i \in I\} \in \mathcal{LF}$ for some index set I if each $\phi_i \in \mathcal{LF}$

7. $\langle \phi_1, \phi_2, \ldots, \phi_n \rangle \in \mathcal{LF}$ if each $\phi_i \in \mathcal{LF}$

The satisfiability clauses require the redefinition of our model theoretic elements. We generalise the notion of a finite state automata to a *generalised deterministic finite state automaton* (GDFA), which is given by a tuple $\langle Q, q_0, L, \delta, A, \pi \rangle$, where

- Q is a set of atoms known as states,
- $Q' = Q \cup Pow(Q) \cup Q^*$,
- $q_0 \in Q'$ is a distinguished state known as the start state,
- L is a set of atoms known as labels,
- δ is a partial function from $Q \times L$ to Q',
- A is a set of atoms, and
- π is a partial assignment of atoms to final states.

Satisfiability may now be defined:

6. $\mathcal{A} \models \{\phi_i \mid i \in I\}$ iff the start state of $\mathcal{A}$ is $q_0 \in Pow(Q)$ and there exists a one-one onto mapping Φ from $\{\phi_i \mid i \in I\}$ to q_0 such that $\mathcal{A}_i \models \phi_i$ for each $\phi_i \in \{\phi_i \mid i \in I\}$, where $\mathcal{A}_i$ is the sub-GDFA of $\mathcal{A}$ with start state $\Phi(\phi_i)$.

7. $\mathcal{A} \models \langle \phi_1, \phi_2, \ldots, \phi_n \rangle$ iff the start state of $\mathcal{A}$ is $q_0 = \langle q_1, q_2, \ldots, q_n \rangle \in Q^*$ and $\mathcal{A}_i \models \phi_i$ for each $\phi_i \in \{\phi_i \mid i \in I\}$, where $\mathcal{A}_i$ is the sub-GDFA of $\mathcal{A}$ with start state q_i.

This definition allows sets but not sets of sets: Q' includes all subsets of Q, but no sets of subsets of Q. This is sufficient for our purposes.

Note that the interpretation of sets here is different to that employed in HPSG. In HPSG, the mapping between the elements in a set description and the elements in the set described is many to one, so that, for example, a set of ten descriptions may describe a set containing only one element. The use made of sets in CPSG, to indicate subcategorisation requirements, requires the one-one mapping between descriptions and elements of the model that we have employed above.

5.5 Poset Values and Poset Unification

Given list and set values, and the view of a list as a totally ordered set, we might extend the formalism to deal with arbitrary posets. A poset is a pair consisting of a set together with a binary ordering relation on that set that is reflexive, anti-symmetric, and transitive. We may model a set in the empirical domain by a poset in the model theoretic domain with the null ordering on the elements of the model theoretic set. A list in the empirical domain may be modelled by a poset with a total ordering on the elements of the model theoretic domain. On this view, lists and sets are extremes on a continuum. As suggested in the previous section, the modelling may be achieved by following the approach of [Rounds 88] to sets (using non-deterministic finite state automata), together with an ordering on the non-deterministic state changes.

As usual, unification may be defined in terms of the least upper bound of the subsumption relation, but to do this we must first define the subsumption relation between posets. It is fairly clear how we want list unification to behave: the unification of two lists should succeed iff the lists are of equal length (i.e., the posets are of equal cardinality) and the first elements of the lists unify, and the second elements of the list unify, and so on. In the case of arbitrary posets the position is not so clear. If we take the posets to each represent partial information about some model theoretic partially order set, then we might take the partial ordering on each poset as indicating partial information about the ordering of that poset, or we might take the partial order as indicating total information about the ordering of the poset modelled. If we assume the former, then we would be required to allow a set and a list to unify, with the result being a list.

This is contrary to our requirements: sets and lists should not unify. Given this, two posets can only unify if they are isomorphic, i.e., if there exists a one-one onto function from one poset to the other which preserves the partial order. Given a particular isomorphism, poset unification becomes deterministic (i.e., yields one non-disjunctive solution). The unification of the posets $\langle P, \leq_p \rangle$ and $\langle Q, \leq_q \rangle$, given the isomorphism $\phi : [P \rightarrow Q]$ is thus the poset whose elements are $p \sqcup \phi(p))$ for each $p \in P$ and ordered according to $(p_1 \sqcup \phi(q_1)) \leq (p_2 \sqcup \phi(p_2))$ iff $p_1 \leq_p p_2$ (or equivalently $\phi(p_1) \leq_q \phi(p_2)$). Poset unification without such an isomorphism is non-deterministic: there may be one distinct solution for each possible isomorphism between the posets. In the case of set unification, where for n element sets there are $n(n-1)/2$ isomorphisms, this leads to a possible $n(n-1)/2$ solutions. List unification, on the other hand is deterministic because there is only one isomorphism between the elements of a list.

5.6 Functional and Relational Dependencies

HPSG also makes use of *functionally dependent* values: values which are related to other values by some function. In the AVM below, for example, the value of the AREA attribute is functionally dependent on the values of the HEIGHT and LENGTH attributes.

$$\begin{bmatrix} \text{HEIGHT} & \boxed{1} \\ \text{LENGTH} & \boxed{2} \\ \text{AREA} & \textit{product}(\boxed{1}, \boxed{2}) \end{bmatrix}$$

[Pollard & Sag 87] distinguish between such functional dependencies and *relational dependencies*, claiming that the former are more computationally tractable. Given that any relation between n arguments can be expressed as a function of n arguments to $\{\mathrm{T}, \mathrm{F}\}$, this claim is clearly not true, and when feature structures are viewed as a complex of constraints on some semantic object (whether the semantic object is considered to be a some automaton or some other form of semantic object), relational dependencies are perhaps more intuitive than functional dependencies: with relational dependencies, values may be constrained without isolating a particular element, that which might otherwise be the value of the functional dependency, and without the requirement that the constraint yield a unique value for that element, i.e., relational dependencies do not impose the asymmetry between values that function dependencies impose.

The use of functional and relational dependencies is clearly a further source of complexity in the feature structure logic. The satisfiability problem, the problem of determining whether a given semantic object satisfies a given syntactic description, for example, is clearly dependent on the complexity of the functions or relations involved. If we allow arbitrary functions and relations, this problem need not be even decidable. Given that we allow disjunction in out language, the satisfiability problem for formulae of the language is already NP-complete (see [Kasper & Rounds 86]), so we don't see the introduction of relations whose satisfiability is also NP-complete as a problem.

Given the above, we do not distinguish between relational and functional dependencies, normally treating functions in terms of relations. The use of relational dependencies does, however, demand further notation. In AVM notation, we express such dependencies as side conditions on the AVM. Thus we have, for example:

$$\begin{bmatrix} \text{HEIGHT} & \boxed{1} \\ \text{LENGTH} & \boxed{2} \\ \text{AREA} & \boxed{3} \end{bmatrix}$$

where $\boxed{3} = \boxed{1} \times \boxed{2}$

Clearly we must also formally specify the relations employed in the side conditions: in the above case we must say what $\boxed{1}$, $\boxed{2}$, and $\boxed{3}$ are such that they stand in the relation specified.

Relational dependencies can be thought to "percolate" to the top-most feature structure in which they occur. That is, a feature structure of the form:

$$\left[\alpha|\beta|\gamma \quad \boxed{1}_{\Gamma}\right]$$

where $\boxed{1}$ is an embedded feature structure constrained by various associated relations Γ, is equivalent to

$$\left[\alpha|\beta|\gamma \quad \boxed{1}\right]_{\Gamma}$$

Formally, relational dependencies are just further constraints which the semantic object which a feature structure describes must satisfy. To introduce such constraints into the

language $\mathcal{LF}$ we entertain a class of n-place symbols $\mathcal{R}_n$ for each non-negative integer n, and augment the definition of syntax with the clause:

8. $R_n(\langle p_1, \ldots, p_n \rangle) \in \mathcal{LF}$ iff $R_n \in \mathcal{R}_n$ and $p_i \in L^*$ for each p_i.

We can side-step the issue of complexity by assuming an interpretation for each R_n: an interpretation H is a total function from $\mathcal{R}_n$ to $Pow(Q^n)$ for each non-negative integer n. An interpretation thus map each relation symbol of arity n to the set of n-tuples which that relation holds of. Given an interpretation H,

8. $\mathcal{A} \models R_n(\langle p_1, \ldots, p_n \rangle)$ iff $\langle \delta^*(q_0, p_1), \ldots, \delta^*(q_0, p_n) \rangle \in H(R_n)$.

6 Summary

The primary purpose of this chapter has been to introduce and provide background on a number of areas which are central to this thesis. After a presentation of the scope and goals of the thesis, the organisation of HPSG was discussed and compared with that of CPSG. Essentially, HPSG uses sorted feature structures to describe linguistic tokens, whereas CPSG takes the typing of tokens to be prior to the use of feature structure descriptions, and bases the use of descriptions on this typing via a classificatory system. CPSG does not employ sorted feature structures. The organisation of CPSG was then made more concrete with the presentation of a theory of classification following that of [Seligman 90a]. This presentation emphasised the linguistic applications of such a theory. The theory presented extends Seligman's work by considering structured tokens, as required by most linguistic theories, and lays the groundwork for the introduction of classificatory systems, which employ feature structure descriptions to mediate the *is of type* relation within a classification. This merger of feature structures with the notion of classification is motivated by the use of feature structures in many modern computationally oriented theories of linguistics. The chapter concludes with an extensive discussion of feature structures from the perspective of an equational logic, and the extensions to the basic formalism of [Kasper & Rounds 86] required by both HPSG and CPSG. These extensions include the use of sorts, sets, lists and functional and relational dependencies.

Chapter 2

Constituent Hierarchies

Having developed the formal framework of classificatory systems, we now consider in detail their application to linguistic theory. In section 1 we consider how a classificatory system may be applied to the lexicon, with special emphasis on the hierarchical structure on the domain of lexical types. This structure involves *subtype* relationships, where all instances of one type are necessarily instances of another type, and the partitioning of types by subtypes, where all instances of one type may be partitioned into instances of subtypes. This application thus leads us to consider a restricted class of classificatory systems, *hierarchical classificatory systems*, which are introduced in section 2. In section 3 we extrapolate the structure on the lexical domain to the domain of all constituents, both lexical and phrasal, and suggest that constituents may be classified according to a "constituent hierarchy", of which the lexical hierarchy forms but one branch. Section 4 illustrates how advantage may be taken of subtype relationships within a classificatory system to reduce the redundant specification of features for types via feature inheritance. In the final section we illustrate how grammar rules, lexical rules and phrasal rules may be phrased as law-like dependencies which linguistic classifications respect.

1 Classifying Words: Lexical Hierarchies

Recall that lexical items that share common features are commonly treated as members of a class — classes such as **noun** and **verb**, for example, are central to syntactic theorising. We may treat the members of such classes as the instances of types. Each

class corresponds to a type within a classification of words. It is also common to further subdivide the classes (according to other common features) into, for example, proper noun, common noun, pronoun, and intransitive verb, transitive verb, ditransitive verb, control verb. These classes may be seen as further types in a classification (or classificatory system), related to the preceding types via law-like dependencies, where the tokens are strings of words. The classes and subclasses form the basis of a lexical hierarchy: a *hierarchical* classification of lexical items.

1.1 Control Verbs: An Illustrative Paradigm

To illustrate a part of the lexical hierarchy and demonstrate how it may be treated in terms of a classificatory system, consider the paradigm of control verbs. We may classify control verbs according to their subcategorisation requirements. All members of the class control verb subcategorise for a subject noun phrase and a verb phrase complement. The class control verb can be partitioned into two subclasses: intransitive control (members of which require no further complements) and transitive control (members of which require a direct object noun phrase). 'Try' and 'persuade' are, respectively, examples of each of these classes:

(1) a. $[\text{Tom}]_{subj}$ tries $[\text{to miaow}]_{comp}$

b. $[\text{Tigger}]_{subj}$ persuades $[\text{Tom}]_{obj}$ $[\text{to miaow}]_{comp}$

We may encode subcategorisation requirements in terms of three binary attributes, SUBJ, DOBJ, and COMP. All tokens of type control verb are [SUBJ +] and [COMP +]. Tokens of type transitive control are also [DOBJ +], whereas tokens of type intransitive control are [DOBJ −].

The class intransitive control is commonly subdivided into the classes intransitive equi and intransitive raising. 'Try' is a member of the former, whereas 'appear' is a member of the latter. One distinguishing characteristic of these classes is semantic: intransitive equi verbs have an agent argument role ('Tom' is the agent of 'try') whereas intransitive raising verbs do not ('Tom' is not the agent of 'appear'). This is reflected in possible paraphrasings: 'Tom tries to miaow' may be paraphrased as 'Tom tries for Tom to miaow' but not as 'It tries that Tom miaows'; 'Tom appears to miaow' may

be paraphrased as 'It appears that Tom miaows' but not as 'Tom appears for Tom to miaow'.

Similarly, transitive control may be divided into transitive equi (whose members include 'promise' and 'persuade') and transitive raising (whose members include 'believe' and 'expect'). Again one distinction between these classes is semantic: transitive equi verbs have an argument role corresponding to their direct object (the act of promising involves someone that promises, something that is promised, and someone that is promised that thing), whereas transitive raising verbs do not (an instance of believing involves only a believer and a belief).

Transitive equi may be subdivided into subject equi (where it is the subject of the verb which notionally fills the agent argument role of the embedded verb phrase, as in 'promise'), and object equi (where it is the direct object of the verb which notionally fills the agent argument role of the embedded verb phrase, as in 'persuade'). Cf. 'Tom promises Tigger to miaow' (where 'Tom' is the agent of 'miaow') and 'Tom persuades Tigger to miaow' (where 'Tigger' is the agent of 'miaow').

Transitive raising cannot be subdivided in a corresponding manner. In the case of such verbs it is always the direct object that fills the subject/agent role of the embedded clause. There is no other argument role for the direct object to be assigned to (as there is in the case of transitive equi verbs), and so if the subject of the verb were to fill the agent argument role of the embedded clause (as in the case of intransitive raising verbs and subject equi verbs), the direct object would be redundant.

This hierarchy of types of control verbs may be represented diagrammatically as in Figure 2.1.

The paradigm is captured by the classificatory system $\langle T, \mathcal{F}, \Delta \rangle$, where

- T is the set {control verb, intransitive control, transitive control, intransitive equi, intransitive raising, transitive equi, transitive raising, object equi, subject equi},
- $\mathcal{F}$ is the set of feature structures defined in terms of the attributes {SUBJ, DOBJ, COMP, AGENT, PATIENT, THEME, PRED} and atomic values {+, −, *subj*, *obj*, *xcomp*, *none*}, and

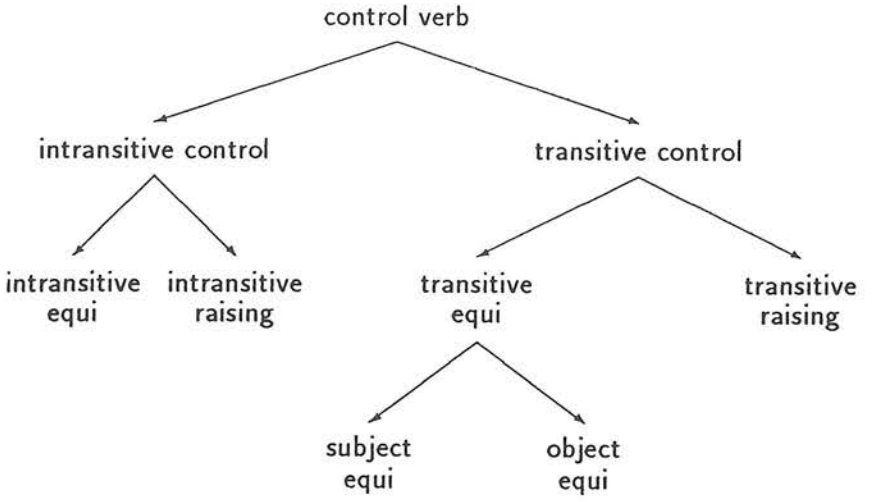


Figure 2.1: A Hierarchy of Control Verbs

- $\Delta : [T \rightarrow \mathcal{F}]$ is defined as:

$$\Delta(\text{control verb}) = \begin{bmatrix} \text{SUBJ} & + \\ \text{COMP} & + \\ \text{THEME} & \begin{bmatrix} \text{PRED} & \textit{xcomp} \end{bmatrix} \end{bmatrix}$$

$$\Delta(\text{intransitive control}) = \begin{bmatrix} \text{SUBJ} & + \\ \text{COMP} & + \\ \text{DOBJ} & - \\ \text{THEME} & \begin{bmatrix} \text{AGENT} & \textit{subj} \\ \text{PRED} & \textit{xcomp} \end{bmatrix} \end{bmatrix}$$

$$\Delta(\text{transitive control}) = \begin{bmatrix} \text{SUBJ} & + \\ \text{COMP} & + \\ \text{DOBJ} & + \\ \text{AGENT} & \textit{subj} \\ \text{THEME} & \begin{bmatrix} \text{PRED} & \textit{xcomp} \end{bmatrix} \end{bmatrix}$$

$$\Delta(\text{intransitive raising}) = \begin{bmatrix} \text{SUBJ} & + \\ \text{COMP} & + \\ \text{DOBJ} & - \\ \text{AGENT} & \textit{none} \\ \text{THEME} & \begin{bmatrix} \text{AGENT} & \textit{subj} \\ \text{PRED} & \textit{xcomp} \end{bmatrix} \end{bmatrix}$$

$$\Delta(\text{intransitive equi}) = \begin{bmatrix} \text{SUBJ} & + \\ \text{COMP} & + \\ \text{DOBJ} & - \\ \text{AGENT} & subj \\ \text{THEME} & \begin{bmatrix} \text{AGENT} & subj \\ \text{PRED} & xcomp \end{bmatrix} \end{bmatrix}$$

$$\Delta(\text{transitive raising}) = \begin{bmatrix} \text{SUBJ} & + \\ \text{COMP} & + \\ \text{DOBJ} & + \\ \text{AGENT} & subj \\ \text{PATIENT} & none \\ \text{THEME} & \begin{bmatrix} \text{AGENT} & obj \\ \text{PRED} & xcomp \end{bmatrix} \end{bmatrix}$$

$$\Delta(\text{transitive equi}) = \begin{bmatrix} \text{SUBJ} & + \\ \text{COMP} & + \\ \text{DOBJ} & + \\ \text{AGENT} & subj \\ \text{PATIENT} & obj \\ \text{THEME} & \begin{bmatrix} \text{PRED} & xcomp \end{bmatrix} \end{bmatrix}$$

$$\Delta(\text{subject equi}) = \begin{bmatrix} \text{SUBJ} & + \\ \text{COMP} & + \\ \text{DOBJ} & + \\ \text{AGENT} & subj \\ \text{PATIENT} & obj \\ \text{THEME} & \begin{bmatrix} \text{AGENT} & subj \\ \text{PRED} & xcomp \end{bmatrix} \end{bmatrix}$$

$$\Delta(\text{object equi}) = \begin{bmatrix} \text{SUBJ} & + \\ \text{COMP} & + \\ \text{DOBJ} & + \\ \text{AGENT} & subj \\ \text{PATIENT} & obj \\ \text{THEME} & \begin{bmatrix} \text{AGENT} & obj \\ \text{PRED} & xcomp \end{bmatrix} \end{bmatrix}$$

The classificatory system respects several law-like dependencies, including:

$$v : \text{intransitive control} \Rightarrow v : \text{control verb}$$

$$v : \text{intransitive equi} \Rightarrow v : \text{intransitive control}$$

$$v : \text{intransitive raising} \Rightarrow v : \text{intransitive control}$$

$$v : \text{transitive control} \Rightarrow v : \text{control verb}$$

$$v : \text{transitive equi} \Rightarrow v : \text{transitive control}$$

$$v : \text{subject equi} \Rightarrow v : \text{transitive equi}$$

$$v : \text{object equi} \Rightarrow v : \text{transitive equi}$$

$$v : \text{transitive raising} \Rightarrow v : \text{transitive control}$$

[Flickinger 87] and [Pollard & Sag 87] give detailed accounts of more complete lexical hierarchies. We concentrate here more on relevant properties of classifications of lexical constituents.

1.2 Alternate Classifications

In the hierarchy of (types of) control verbs above, the verbs are partitioned into intransitive control verbs and transitive control verbs. Alternately we may partition control verbs into equi verbs and raising verbs — just as every control verb is either intransitive or transitive, every control verb is either equi or raising — and then partition these classes according to transitivity. The resulting hierarchy is given in Figure 2.2.

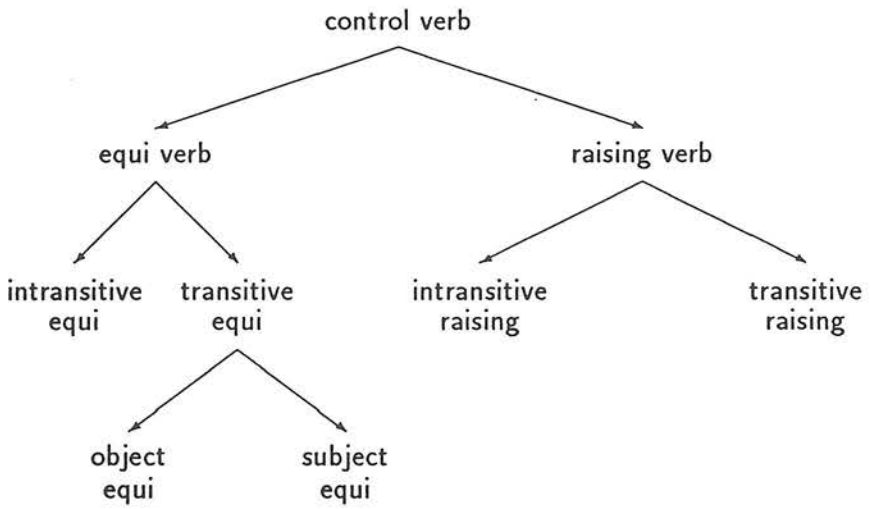


Figure 2.2: An Alternate Hierarchy of Control Verbs

Yet another alternative is to take the types intransitive raising, intransitive equi, transitive raising and transitive equi as immediate subtypes of control verb. Cases such as these, where more than one possible partitioning of the tokens of a type is apparent, challenge the use of a single hierarchy in structuring the lexicon. It appears that the relationships

between types might best be characterised via a *network*, rather than a hierarchy, as in Figure 2.3.

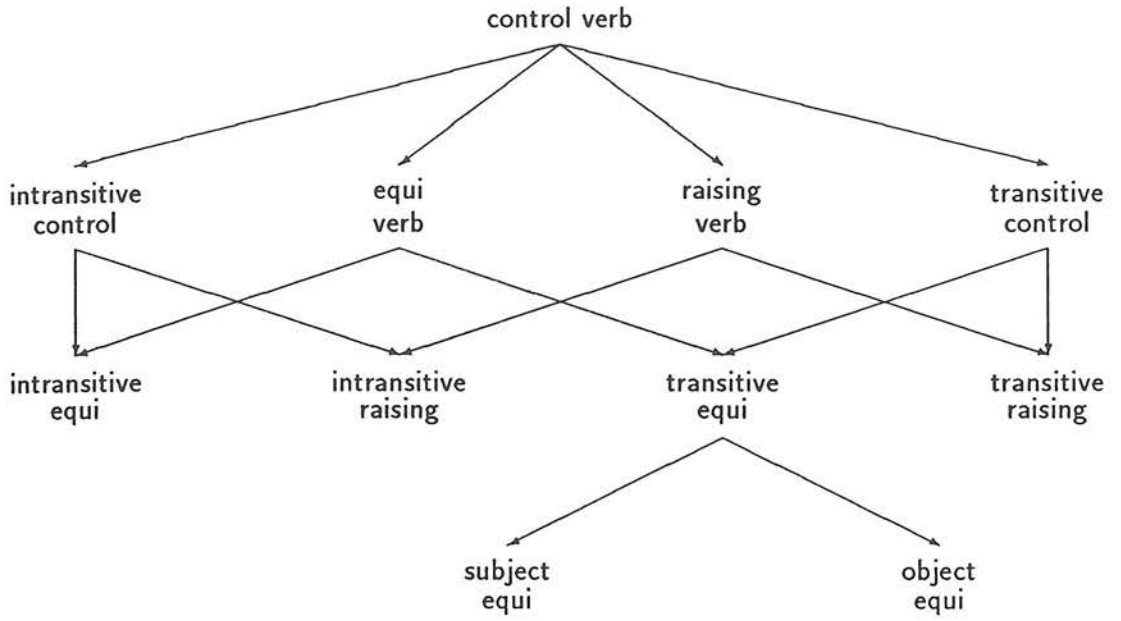


Figure 2.3: A Network of Control Verbs

Clearly more motivation is required if the lexicon is to be considered as being structured purely hierarchically. It is difficult to give concrete motivation, except that hierarchical structuring is clearly more restrictive than the general network structuring and has computational advantageous. If hierarchical structuring is sufficient, then we may capitalise on the computational advantages that it affords us.

Some motivation for hierarchical structuring may be given at a local level by motivating the individual types corresponding to internal nodes. There are two (related) principal sources of motivation for such types. Firstly, different features are relevant to the subdivision of various classes. The feature CASE, for example, is relevant to the subdivision of nouns but not verbs, whereas the feature AUX is relevant to the subdivision of verbs but not nouns. This suggests that somewhere in the classification of lexical items there should be types corresponding to the classes of nouns and verbs, and that the members of those classes should be disjoint. Secondly, generalisations over classes of tokens exist and these can best be captured by treating tokens as instances of a type, and stating the generalisations in terms of types. Nouns, verbs, prepositions, adjectives, adverbs and

determiners, for example, are all projectable categories — they have phrasal projections — whereas conjunctions and complementisers are non-projectable. This motivates the partitioning within HPSG of lexical sign into major lexical sign and minor lexical sign[1]. Other important generalisations concern the participation by tokens of various types in law-like dependencies which the classifications respect. We discuss such dependencies in section 5. It is not clear that the types required to capture such generalisations require a network — a hierarchy seems sufficient.

1.3 Cross-Classification

An alternative to a network of types that retains many of the computational advantages of hierarchical structuring is *cross-classification*, whereby tokens are cross-classified according to several hierarchies. Control verbs might be cross-classified according to the two hierarchies of Figure 2.4, the first capturing the equi/raising distinction and the second capturing the intransitive/transitive distinction.

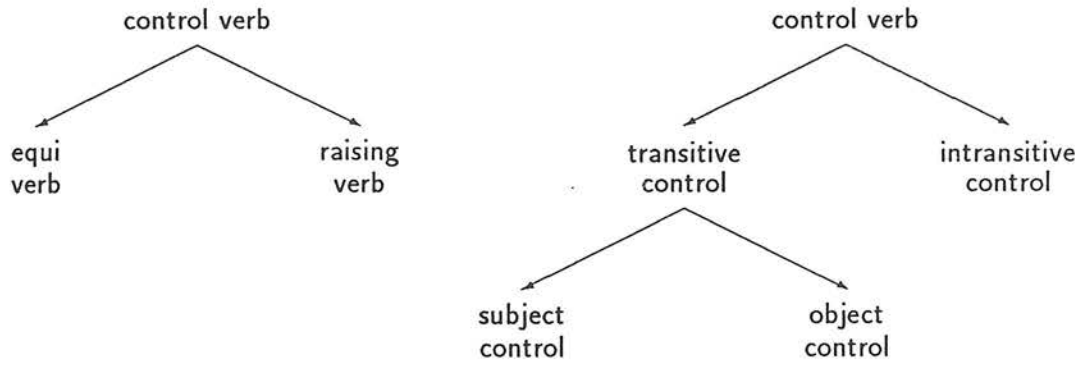


Figure 2.4: Hierarchies To Cross-Classify Control Verbs By

It is important to note that these hierarchies are not independent. If a verb is classified according to the left hierarchy as being of type raising verb, then it cannot be classified according to the right hierarchy as being of type subject control, as transitive raising verbs are necessarily of type object control (as explained in section 1.1).

The need for cross-classification becomes most apparent when classifying inflected forms of words. The classifications of control verbs we have considered are based on subcat-

[1]Note that HPSG takes determiners to be minor lexical items. That determiners are projectable justifies CPSG's treatment of such lexical items as major.

egorisation characteristics of control verbs. Control verbs (and verbs in general) could also, however, be classified according to their form (*finite*, *base*, *past participle*, *present participle*, *passive*, *infinite*, or *gerund*), or according to agreement features (the number/person of their subject: *first*, *second*, *third*, *singular*, *plural*). Classification by form cuts across that by subcategorisation in that members of the same subcategorisation class may belong to different form classes and vice versa (though again the classifications are not independent: intransitive verbs do not have a passive form, for example.) A similar situation holds for classification by agreement features. The applicability of cross-classification is also not limited to verbs — all classes whose members exhibit inflected forms may be cross-classified. However, if we are concerned only with classifying base forms, it is not clear that cross-classification is required. The lexicon of CPSG is based on a single hierarchical classification of base forms. Inflected forms are derived from these base forms via the application of lexical rules, which allow the feature structure descriptions of inflected forms to be derived from those of base forms. For a more detailed discussion of the lexicon, see section 4 of Chapter 9, where a computational implementation of the lexicon is presented.

2 Hierarchical Classificatory Systems

The *subtype* relation, which is a binary relation on the set of types, may be used to capture the hierarchical structure present in the lexical hierarchy when it is treated as a classificatory system. This relation arises from a class of law-like dependencies that the classificatory system respects.

2.1 The Subtype Relation

Given a classification $\langle S, T, : \rangle$, t' is a *subtype* of t, written $t' \leq t$, if the classification respects a law-like dependency requiring that all tokens of type t' are also of type t. i.e., if $t, t' \in T$, then $t' \leq t$ (with respect to the classification $\langle S, T, : \rangle$) if the class of law-like dependencies which $\langle S, T, : \rangle$ respects includes

$$v : t' \Rightarrow v : t$$

Within a classification, there is the temptation to give an extensional definition of

subtype:

$$t' \leq t \text{ iff } \{s \mid s : t'\} \subseteq \{s \mid s : t\}$$

As in the case of the proposed extensional definition of Δ in section 4.2 of Chapter 1, this does not capture the distinction between law-like dependencies and accidental correlations.

Within a classificatory system, the subtype relationship may be captured in terms of feature structure descriptions of types. For any classificatory system $\langle T, \mathcal{F}, \Delta \rangle$,

$$\Delta(t') \sqsupseteq \Delta(t) \text{ iff } t' \leq t$$

The subtype relation has some important properties. Firstly, it is reflexive: every classification respects

$$v : t \Rightarrow v : t$$

for each type t of the classification, and so every type is a subtype of itself. The subtype relation is also transitive: if a classification respects

$$v : t' \Rightarrow v : t$$

and

$$v : t'' \Rightarrow v : t'$$

then it must also respect

$$v : t'' \Rightarrow v : t$$

so if $t'' \leq t'$ and $t' \leq t$ then $t'' \leq t$. The relation is, however, not antisymmetric (and hence not a partial order). If $t_1 \leq t_2$ and $t_2 \leq t_1$, then t_1 and t_2 are extensionally equivalent, but not necessarily the same type.

2.2 The Covers Relation

From the subtype relation we may derive the *covers* relation. For t, t' distinct types, t covers t', written $t \succ t'$, if $t' \leq t$, $t \not\leq t'$ and whenever $t' \leq t'' \leq t$, either $t' = t''$ or $t'' = t$. If t covers t', then t' is an *immediate subtype* of t, written $t' \prec t$. "$\leq$" is the reflexive transitive closure of "$\prec$".

Several dependencies between types relevant to the classification of linguistic tokens may be cast in terms of the covers relation. In the classification of control verbs in Figure 2.1, for example, we have:

$$\text{control verb} \succ \text{intransitive control}$$

$$\text{control verb} \succ \text{transitive control}$$

$$\text{intransitive control} \succ \text{intransitive equi}$$

$$\text{intransitive control} \succ \text{intransitive raising}$$

$$\text{transitive control} \succ \text{transitive equi}$$

$$\text{transitive control} \succ \text{transitive raising}$$

$$\text{transitive equi} \succ \text{subject equi}$$

$$\text{transitive equi} \succ \text{object equi}$$

The covers relation may be represented graphically by taking the set of types as the nodes of the graph and joining the nodes t and t' with a directed arc from t to t' if $t \succ t'$. The graph corresponding to the above relation is the hierarchy of Figure 2.1. Such graphs may be treated in terms of a set of "local trees". The local tree rooted at the node t is the subgraph consisting of t, each arc from t, and each node that those arcs lead to. The local tree rooted at transitive control, for example, is given in Figure 2.5.

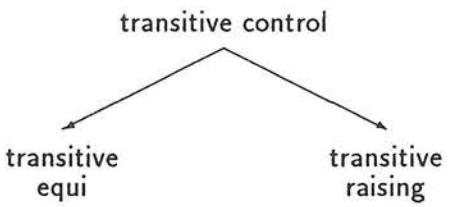
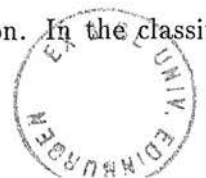


Figure 2.5: A Local Tree Rooted At Transitive Control

2.3 Partitioned Types

Law-like dependencies may interact to ensure that a subset of the set of immediate subtypes of a type necessarily partition the tokens of that type. Linguistic examples of such interaction are also common. In the classification of control verbs, for example,



the types intransitive control and transitive control partition the tokens of type control verb: each control verb is necessarily either an intransitive control verb or a transitive control verb, and no control verb can be both. Similarly the types intransitive equi and intransitive raising partition the tokens of type intransitive control.

We introduce a special notation to abbreviate the interacting dependencies:

$$\textsf{control verb} \twoheadrightarrow \{\textsf{intransitive control, transitive control}\}$$

Such statments require that the set of types on the right of the " $\twoheadrightarrow$ " symbol necessarily partition the tokens of the type on the left. In long-hand this may be written as:

$$v : \textsf{control verb} \Rightarrow v : \textsf{intransitive control} \vee v : \textsf{transitive control}$$

$$v : \textsf{intransitive control} \Rightarrow \neg\, v : \textsf{transitive control}$$

$$v : \textsf{transitive control} \Rightarrow \neg\, v : \textsf{intransitive control}$$

We shall talk of such interacting dependencies as a single dependency, using the above notation.

Again, such dependencies cannot be defined within a classification in extensional terms. Nor, however, can they necessarily be defined within a classificatory system in terms of Δ. The reason for this is that, due to subtle dependencies (often referred to as *feature co-occurrence restrictions*) between features, not all elements of $\mathcal{F}$ are descriptions of tokens, and so partitioning the tokens of a type need not involve partitioning the feature structures which extend the description of that type.

To illustrate this point, consider GPSG, which contains feature co-occurrence restrictions such as [PFORM] $\supset$ [−V,−N]. This requires that all syntactic categories for which the PFORM feature is specified are also specified to be [−V,−N]. Consequently, feature structures which extend

$$\begin{bmatrix} \text{PFORM} & to \\ \text{V} & + \\ \text{N} & + \end{bmatrix}$$

are not valid descriptions of any syntactic category. Thus a type whose description subsumes this feature structure may be partitioned by subtypes even if none of the descriptions of the subtypes subsumes this feature structure.

2.4 Hierarchical Classificatory Systems

A hierarchical classificatory system is a classificatory system in which there is a *root* type, of which all tokens are instances, and which respects a number of law-like dependencies requiring that for each type, the set of all immediate subtypes is either empty or partitions the tokens of that type.

2.4.1 An Example: Control Verbs Revisited

Given that the initial classificatory system for control verbs respects the following law-like dependencies:

control verb $\twoheadrightarrow$ {intransitive control, transitive control}

intransitive control $\twoheadrightarrow$ {intransitive raising, intransitive equi}

transitive control $\twoheadrightarrow$ {transitive raising, transitive equi}

transitive equi $\twoheadrightarrow$ {subject equi, object equi}

and that intransitive raising, intransitive equi, transitive raising, subject equi, and object equi all have no immediate subtypes, that classification of control verbs is a hierarchical classificatory system with root type control verb.

Some important points are illustrated by this example:

- Every token that the system classifies must be of type control verb, and so

$$\begin{bmatrix} \text{SUBJ} & + \\ \text{COMP} & + \\ \text{THEME} & \begin{bmatrix} \text{PRED} & \textit{xcomp} \end{bmatrix} \end{bmatrix}$$

  partially describes every token. In general, every hierarchical classificatory system contains a least descriptive type of which all tokens are instances. This may be the totally vacuous description: [ ].

- Each token is an instance of exactly one immediate subtype of control verb (i.e., intransitive control or transitive control), and exactly one immediate subtype of that

type, etc.. The set of types with no subtypes (the *leaf* types) thus partitions the tokens — every token is an instance of one and only one leaf type — and there is a unique path from the root node to each leaf node. That is, the graph of the covers relation on the set of types is a tree.

- A full description of a token may therefore be given in terms of a type corresponding to a leaf along with any idiosyncratic feature-value pairs that describe the token. This is how [Pollard & Sag 87] state the lexicon of HPSG (though Pollard & Sag employ a different notion of *type* than that employed here, and the use of cross-classification leads to multiple inheritance).

2.4.2 Two Views of Hierarchical Classificatory Systems

[Pollard & Sag 87] employ a hierarchically structured lexicon with cross-classification in their presentation of HPSG to efficiently encode lexical information and capture generalisations over classes of lexical items. A lexical entry of HPSG includes a feature structure specifying any idiosyncratic features of the word (such as its phonology) and a set of classes that the word belongs to. For example, the lexical entry for 'promise' includes a feature structure specifying the phonology of 'promise' and the information that the word belongs to the classes subject equi and base. A lexical entry is compiled into a more complete feature structure by unifying the feature structures corresponding to each class that the word belongs to with the feature structure specifying the idiosyncratic features.[2] The classification of lexical items is thus used by HPSG to determine the featural description of a word based on the classes that the word belongs to.

The presentation of the hierarchy of control verbs as a classificatory system suggests an alternate way of viewing the lexical hierarchy: as a system where given a word with a known description, that word may be classified according to the hierarchy as being an instance of certain types (or belonging to certain classes). For example, knowing the description of 'seem' (as in 'Tom seems to be miaowing') allows us to classify it as being of type intransitive raising in our classificatory system for control verbs.

There are thus two ways to view the lexical hierarchy, and these two views arise from

[2] HPSG also employs feature inheritance (see section 4) in the lexical hierarchy to reduce the redundant specification of features.

the treatment of the hierarchy as a classificatory system — any classificatory system can be viewed in either way. In abstract terms, the classificatory system can be seen as specifying a ternary relation between words/tokens, classes/types and feature structure descriptions. The first view of the lexical hierarchy corresponds to using a token and the types it is an instance of to determine the feature structure description of the token, whereas the second corresponds to using the token and its feature structure description to determine the types of which it is an instance. For completeness, we state here an obvious and efficient recursive algorithm for the classification as required by the second view: beginning from the root type of the hierarchy, a word s is of type t if and only if its description is subsumed by $\Delta(t)$ and either t has no subtypes or there is some subtype t' of t of which s is an instance.

The second view of the lexical hierarchy, where words are classified based on their descriptions, is not normally employed in linguistic systems. The feature structure description of a word is not normally used to classify that word, reflecting the fact that the feature structure description of a word is not normally known *a priori*. Normally the classification is used to determine the feature structure description of a word, which is then used to construct the feature structure descriptions of phrases involving that word. The usual use of the lexical hierarchy is as a convenient, space efficient way of listing descriptions of lexical items. However, given a hierarchical classificatory system, an approach similar to the second might be used to place new words on the hierarchy as they are learned: partial descriptions of an unknown word may be inferred from the context in which the word appears and these partial descriptions may be used to place the word appropriately on the lexical hierarchy. As argued below, however, the second view is also tailor made for the classification of phrases, whose feature structure descriptions may be inferred from the descriptions of their constituent words.

3 Classifying Constituents: Constituent Hierarchies

In grammatical theories such as HPSG and various versions of CG words and phrases are treated uniformly within a single framework: in HPSG both words and phrases are modelled by sorted feature structures and in CG the assignment of syntactic categories to phrases is an extension of the assignment of syntactic categories to words. This reflects

a theoretical presumption that words and phrases are elements of a single domain, the only difference between them being that words are atomic. Applying this view to our treatment of the lexicon as a classificatory system suggests that all grammatical strings should be considered as tokens within a single classificatory system. Furthermore, it raises the question of whether the hierarchical structure on the lexicon may be extended to all grammatical strings, both lexical and phrasal.

3.1 Preliminaries

Many linguistic theories subscribe to the view that strings of words are licenced as phrases via grammar rules. Phrase structure grammars, for example, admit strings as phrases via phrase structure rules. Such grammars may be viewed as classificatory systems with a type corresponding to each rule and with the admissible strings as tokens: if two strings are admitted by the same rule then they are of the same type, the type corresponding to that rule. As every admissible string of a language is admitted by a rule of that language, within such a system a string is admissible if and only if it may be classified as being of some type. Furthermore, in a disambiguated language, every admissible string is admitted by exactly one rule, so the types corresponding to the rules partition the admissible phrases.

Within most grammatical theories, the domain of grammar rules is often not unstructured. Any structure on the domain of grammar rules may be reflected in the domain of types corresponding to those rules. In a phrase structure grammar, for example, there are often several rules which licence strings of the same syntactic category: strings of category VP might be licenced by rules involving intransitive verbs, transitive verbs, ditransitive verbs and various sorts of control verbs. These rules may all be seen as licensing different types of VP, and the types corresponding to VPs headed by intransitive verbs, transitive verbs, etc., will all be subtypes of a type corresponding to VP. Furthermore, taking syntactic categories as corresponding to types of constituents allows phrase structure rules to be phrased as law-like dependencies: the phrase structure rule S $\rightarrow$ NP VP, for example, corresponds to "if v_1 is a token of type noun phrase and v_2 is a token of type verb phrase then $concat(v_1, v_2)$ is a token of type sentence".

If the classification is to be extended to a classificatory system, we must give feature

structure descriptions of the tokens and types. Within the domain of phrasal constituents we assume a form of compositionality: the features of a phrasal constituent are dependent on the features of its immediate subconstituents and their mode of combination. Rephrasing this, we have: the feature structure description of a phrase is dependent on the feature structure descriptions of its immediate subconstituents and the grammar rule which licences the phrase.

In this section we have so far been concerned with phrasal constituents: phrases admitted to the grammar via grammar rules. Any classification of phrasal constituents may be extended to also include lexical items (or lexical constituents). The constituents of a grammar may be partitioned into lexical constituents and phrasal constituents: every constituent is either lexical or phrasal and no constituent is both. Thus we may view a grammar as a classificatory system having types constituent, lexical constituent and phrasal constituent (among others) in which all classifiable tokens are of type constituent and:

$$\textsf{constituent} \twoheadrightarrow \{\textsf{lexical constituent}, \textsf{phrasal constituent}\}$$

We may take the type lexical constituent as the root of a lexical hierarchy, with all lexical types being subtypes of lexical constituent. Hierarchical structure on the domain of phrasal types may also be motivated in individual cases.

3.2 A Categorial Grammar Constituent Hierarchy

Categorial grammars, with their relatively simple rule schemas, provide simple examples of grammars which may be viewed as hierarchical classificatory systems. Consider a grammar whose lexicon consists of a set of words and an assignment of syntactic categories to those words and whose well-formed formulae of syntactic category X are

- strings consisting of a single word from the lexicon whose syntactic category is X,
- strings formed by postfixing a well-formed formula of category Y to one of category X/Y (i.e., "forward" application), and
- strings formed by prefixing a well-formed formula of category Y to one of category $Y\backslash X$ (i.e., "backward" application).

In this grammar all lexical well-formed formulae share the property of consisting of a single word, and all non-lexical well-formed formulae share the property of consisting of two concatenated well-formed formulae. This suggests types corresponding to lexical well-formed formulae and phrasal well-formed formulae, as in the schematic constituent hierarchy of Figure 2.6. In this hierarchy X is a variable ranging over syntactic categories.

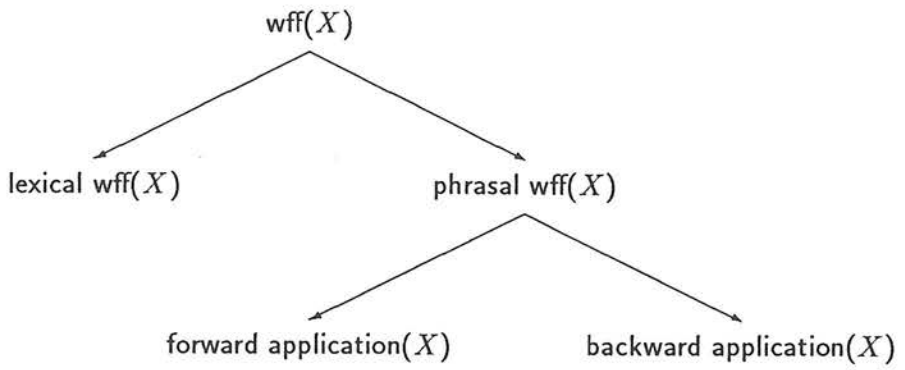


Figure 2.6: A Categorial Grammar Constituent Hierarchy

This may be formalised as a classificatory system $\langle T(X), \mathcal{F}, \Delta \rangle$ (for each syntactic category X) where

- $T(X) = \{\text{wff}(X), \text{lexical wff}(X), \text{phrasal wff}(X), \text{forward application}(X), \text{backward application}(X)\}$,
- $\mathcal{F}$ is the set of feature structures defined in terms of the attributes {CAT, HEAD, COMP, FUNCTOR, ARG, DIR} and atomic values $\{none, left, right, S, NP\}$,
- $\Delta : [T(X) \rightarrow \mathcal{F}]$ is defined as:

$$\Delta(\text{wff}(X)) = \begin{bmatrix} \text{CAT} & fs(X) \end{bmatrix}$$

$$\Delta(\text{lexical wff}(X)) = \begin{bmatrix} \text{CAT} & fs(X) \\ \text{HEAD} & none \\ \text{COMP} & none \end{bmatrix}$$

$$\Delta(\text{phrasal wff}(X)) = \begin{bmatrix} \text{CAT} & \boxed{1}\, fs(X) \\ \text{HEAD} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{FUNCTOR} & \boxed{1} \\ \text{ARG} & \boxed{2} \end{bmatrix} \end{bmatrix} \\ \text{COMP} & \begin{bmatrix} \text{CAT} & \boxed{2} \end{bmatrix} \end{bmatrix}$$

$$\Delta(\text{forward application}(X)) = \begin{bmatrix} \text{CAT} & \boxed{1} fs(X) \\ \text{HEAD} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{FUNCTOR} & \boxed{1} \\ \text{DIR} & right \\ \text{ARG} & \boxed{2} \end{bmatrix} \end{bmatrix} \\ \text{COMP} & \begin{bmatrix} \text{CAT} & \boxed{2} \end{bmatrix} \end{bmatrix}$$

$$\Delta(\text{backward application}(X)) = \begin{bmatrix} \text{CAT} & \boxed{1} fs(X) \\ \text{HEAD} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{FUNCTOR} & \boxed{1} \\ \text{DIR} & left \\ \text{ARG} & \boxed{2} \end{bmatrix} \end{bmatrix} \\ \text{COMP} & \begin{bmatrix} \text{CAT} & \boxed{2} \end{bmatrix} \end{bmatrix}$$

(where *fs* maps syntactic categories to their feature structure descriptions) which respects the following law-like dependencies:

$$\text{wff}(X) \twoheadrightarrow \{\text{lexical wff}(X), \text{phrasal wff}(X)\}$$

$$\text{phrasal wff}(X) \twoheadrightarrow \{\text{forward application}(X), \text{backward application}(X)\}$$

$$v_1 : \text{wff}(X/Y) \wedge v_2 : \text{wff}(Y) \Rightarrow postfix_t(v_2, v_1) : \text{forward application}(X)$$

$$v_1 : \text{wff}(Y) \wedge v_2 : \text{wff}(Y\backslash X) \Rightarrow prefix_t(v_1, v_2) : \text{backward application}(X)$$

The function $postfix_{fs}$, which maps pairs of feature structures to a third feature structure and which corresponds to the function $postfix_t$ on the domain of tokens is given by:

$$postfix_{fs}(\boxed{1},\boxed{2}) = \begin{bmatrix} \text{HEAD} & \boxed{2}\begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{DIR} & right \end{bmatrix} \end{bmatrix} \\ \text{COMP} & \boxed{1} \end{bmatrix}$$

Similarly, the function $prefix_{fs}$, corresponding to $prefix_t$ is given by:

$$prefix_{fs}(\boxed{1},\boxed{2}) = \begin{bmatrix} \text{HEAD} & \boxed{2}\begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{DIR} & left \end{bmatrix} \end{bmatrix} \\ \text{COMP} & \boxed{1} \end{bmatrix}$$

3.3 The Constituent Hierarchy of HPSG

The grammar rules of HPSG are both more numerous and more complex than the rule schemas of the above version of CG. Classificatory systems based on these rules are more

complex than that of the previous subsection and allow more interesting hierarchical structuring to be motivated on the domain of phrasal constituents

All HPSG words and phrases are modelled by sorted feature structures of sort *sign*. This sort has two subsorts: *lexical sign* and *phrasal sign*. Feature structures of these sorts model words and phrases respectively. These sorts may be taken as corresponding to nodes in a constituent hierarchy, with the lexical hierarchy fitting in below the node corresponding to *lexical sign*. With respect to phrases in HPSG, [Pollard & Sag 87] are mostly concerned with *headed phrases*, though some comments are also made about *coordinate phrases*. These thus correspond to two subtypes of the type corresponding to *phrasal sign*. The headed phrases discussed include *head/complement phrases*, *head/adjunct phrases*, and *head/filler phrases*. We may take these to correspond to three subtypes of headed phrase. Grammar rules 1, 2, and 3 admit different sorts of head/complement phrases. Grammar Rule 4 admits head/adjunct phrases. The three different sorts of head/complement phrases could each be taken to correspond to subtypes of head/comp phrase. However, there is a useful generalisation regarding word order that can be made for phrases admitted by grammar rules 2 and 3: each of these rules requires that the phrase be lexically headed. The (English) linear precedence rules of HPSG thus require this head to precede any complements. Grammar rule 1, on the other hand, only admits phrases with non-lexical heads, and so requires that the head should follow any complements. This thus results in the hierarchy of Figure 2.7. We omit the formal details of the accompanying hierarchical classificatory system.

An important issue in the development of this, and any, hierarchical classification concerns the choice of non-terminal nodes. Some motivation for some of the above non-terminal nodes comes from Pollard and Sag's use of implication in their logic of sorted feature structures to capture various "principles" of grammar ([Pollard & Sag 87]). Under the view which Pollard and Sag present, the Grammar Principles of a language apply to *every* sign of that language. This universal applicability is, however, achieved in terms of a conditional which effectively restricts the applicability of the principle. The Head Feature Principle, for example, may be stated as:

$$\left[\text{DTRS} \quad \textit{headed-structure}\left[\ \right]\right] \Rightarrow \left[\begin{array}{ll} \text{SYN|LOC|HEAD} & \boxed{1} \\ \text{DTRS|HEAD-DTR|SYN|LOC|HEAD} & \boxed{1} \end{array}\right]$$

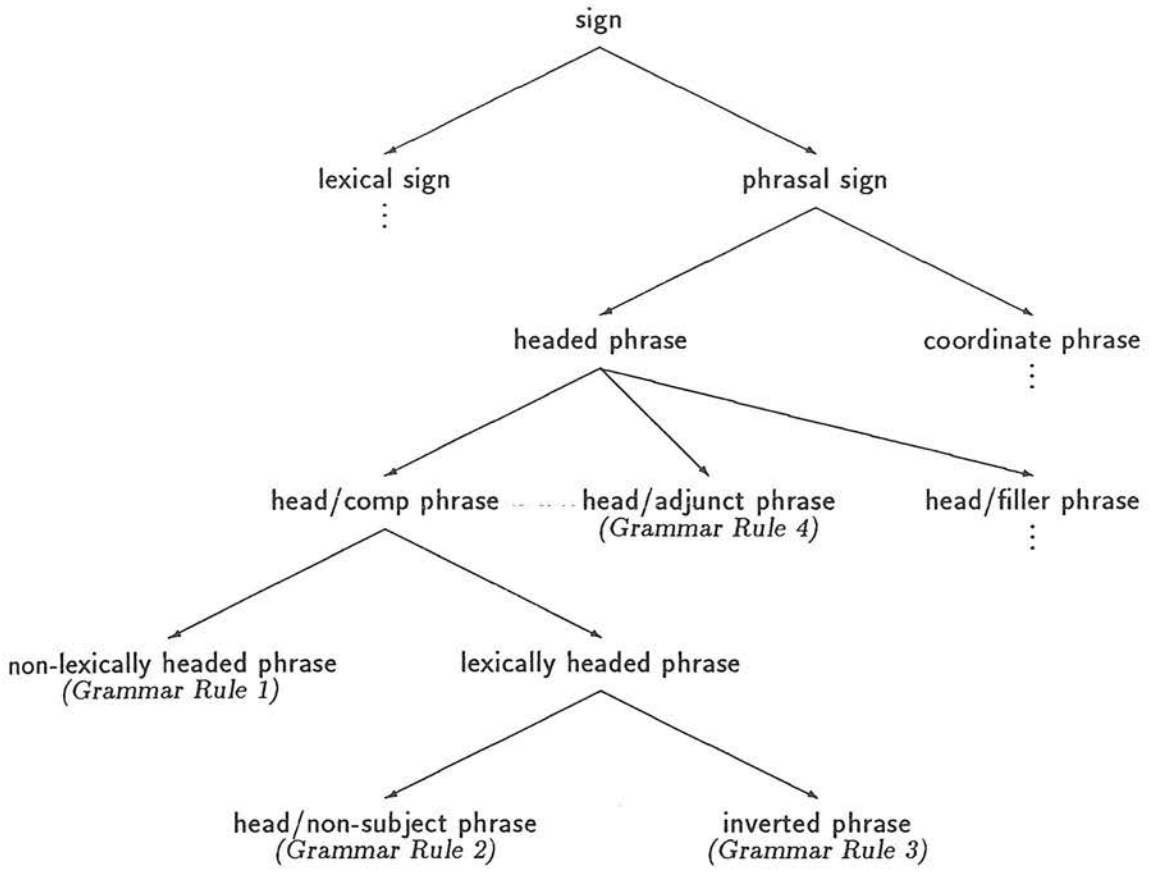


Figure 2.7: The Constituent Hierarchy of HPSG

The conditional prevents this principle from having any force on constituents which are not headed (such as coordinate phrases and lexical items). If a classificatory system contains a type "headed phrase", the conditional may be eliminated by requiring that the description of that type extend the consequent of this principle. That is, with the above hierarchy, we may capture the Head Feature Principle by requiring that:

$$\Delta(\text{headed phrase}) \sqsupseteq \begin{bmatrix} \text{SYN|LOC|HEAD} & \boxed{1} \\ \text{DTRS|HEAD-DTR|SYN|LOC|HEAD} & \boxed{1} \end{bmatrix}$$

Other instances of implication may be treated in a similar fashion: the English Constituent Ordering Principle (which would require implication), for example, motivates the types lexically headed phrase and non-lexically headed phrase. (The above hierarchy is clearly language specific). In this way implication may be absorbed into the classificatory system (motivating the types) and eliminated from the logic of sorted feature structures (simplifying the logic). Adopting this view, we can see that the various instances of implication in Pollard and Sag's formulation of HPSG provide a potential

challenge to a constituent hierarchy. It is possible that various instances of implication could motivate types that do not support a hierarchical structure. That no such types are motivated by their uses of implication lends some (albeit weak) support to this hierarchical view of constituents.

Note that this phrasal hierarchy is relatively small in comparison to the lexical hierarchy of HPSG — far fewer types are involved. This relative smallness, due to the lexical nature of HPSG, has the consequence that the need for cross-classification in the phrasal hierarchy is not apparent. In the lexical hierarchy we have seen cross-classification as being of most use in classifying inflected forms. The distinction between uninflected and inflected forms in the lexical domain is not paralleled in the phrasal domain of HPSG: all such structure is captured in the lexicon by lexical rules, obviating the motivation for cross-classification in the phrasal hierarchy.

3.4 The Constituent Hierarchy of CPSG

The phrasal branch of the constituent hierarchy of CPSG is depicted in Figure 2.8. As in HPSG, phrasal constituents are partitioned into headed phrases and coordinate phrases. Headed phrases are further partitioned into head/argument phrases, head/conjunction phrases and head/adjunct phrases. Head/argument phrases are partitioned into head/topic phrases and head/complement phrases. Head/topic phrases are partitioned into head/specifier phrases and head/filler phrases. Coordinate phrases are partitioned into binary coordinate phrases and iterated coordinate phrases. Motivation for the partitioning of phrases into headed and coordinate phrases follows that in HPSG: headed phrases satisfy the Head Feature Principle, coordinate phrases do not. The partitioning of coordinate phrases into binary and iterated coordinate phrases follows GPSG. The partitioning of headed phrases is based on the role the non-head constituents play in the phrase. In head/conjunction phrases a conjunction, such as 'and', 'both', or 'nor', marks the head. In head/adjunct phrases an adjunct modifies the head. In head/argument phrases, arguments fill one or more of the head's argument roles. English word order is one of the factors motivating the distinction between head/complement phrases (where complement is understood as in GB terms) and head/topic phrases: complements follow their lexical heads and topics (either subjects, specifiers or fillers) precede their non-

lexical heads. However, the distinction is not specific to English. Further motivation for this hierarchy is presented throughout Chapters 3 to 8.

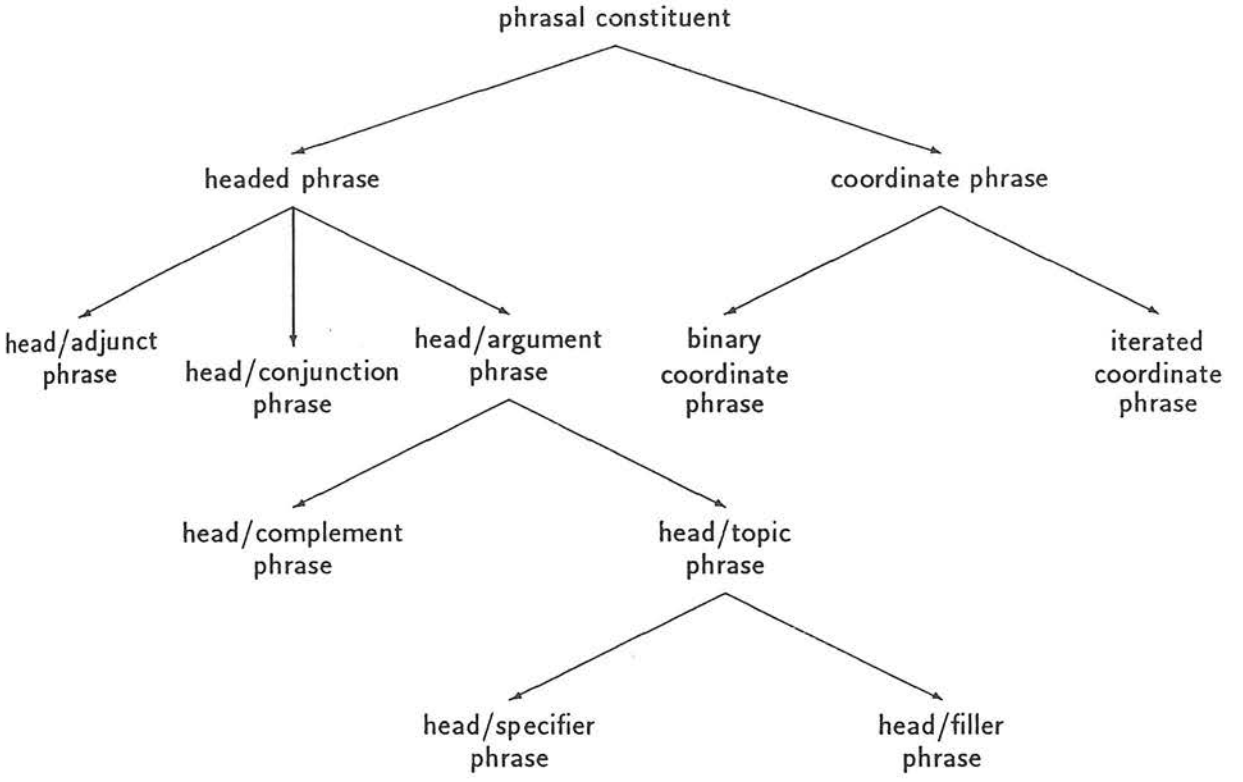


Figure 2.8: The Phrasal Hierarchy of CPSG

The principal differences between the hierarchies of Figure 2.7 and Figure 2.8 are motivated by a concern for potential linguistic universals: the types lexically headed phrase and non-lexically headed phrase from the HPSG hierarchy are motivated by language specific concerns, whereas the CPSG types head/complement phrase and head/topic phrase are intended to have force in all natural languages. The subdivision of head/topic phrase into head/specifier phrase and head/filler phrase is similarly intended to related not just to English. In CPSG, the constituent hierarchy is postulated as a linguistic universal.

4 Feature Inheritance

Within a classificatory system, subtype relationships between types may be inferred from the feature structure descriptions of those types: if $\Delta(t') \sqsupseteq \Delta(t)$ then $t' \leq t$. Stating law-like dependencies stemming from subtype relationships within a classificatory system

is thus redundant. Conversely, partial feature structure descriptions of types can be inferred from subtype relations: if $t' \leq t$, then $\Delta(t') \sqsupseteq \Delta(t)$; and the least upper bound of the descriptions of all types which cover a type (with respect to the subsumption ordering) is a partial description of that type. A complete description of the type is given by the unification of this partial description with a feature structure corresponding to the idiosyncratic features, the features which describe that type but do not describe any of the type's covering types, of the type. Thus we may avoid redundancy by stating only covers relations and idiosyncratic features, and requiring that types (and tokens of those types) *inherit* the descriptions of their covering types. For inheritance networks (as opposed to the strict hierarchies which CPSG employs) where *multiple inheritance* may occur, deriving the covers relation from Δ in this way ensures that no conflicts can arise if features are inherited from more than one type.

4.1 Reducing Redundancy

A classificatory system may be fully defined in compact terms by a network of nodes (where the nodes of the network are the types) and a total assignment Γ of feature structures to types (where $\Gamma(t)$ is the feature structure corresponding to the idiosyncratic features of the type t). Clearly any network augmented with an assignment of feature structures to the nodes does not necessarily define a classificatory system. Δ may be defined in terms of Γ as:

$$\Delta(t) = \Gamma(t) \wedge \left(\bigwedge_{t' \succ t} \Delta(t') \right)$$

For any classificatory system feature inheritance may be employed to reduce redundancy, but this will, in general, involve an inheritance *network* rather than an inheritance *hierarchy*. Only for hierarchical classificatory systems will the network be strictly hierarchical. Inheritance within a hierarchical classificatory system is simplified as in such a system each non-root type is covered by exactly one type. We thus have:

$$\Delta(t) = \begin{cases} \Gamma(t) & \text{if } t \text{ is the root type} \\ \Gamma(t) \wedge \Delta(t') \quad (\text{where } t' \succ t) & \text{otherwise} \end{cases}$$

Our previous hierarchical classificatory system for control verbs may be stated in more compact terms as:

Control verbs may be classified by a hierarchical classificatory system which respects the following law-like dependencies:

control verb $\gg$ {intransitive control, transitive control}

intransitive control $\gg$ {intransitive equi, intransitive raising}

transitive control $\gg$ {transitive equi, transitive raising}

transitive equi $\gg$ {subject equi, object equi}

and where Γ is given by:

$$\Gamma(\text{control verb}) = \left[\text{THEME } \left[\text{PRED } \textit{xcomp}\right]\right]$$

$$\Gamma(\text{intransitive control}) = \begin{bmatrix}\text{DOBJ} & - \\ \text{THEME} & \left[\text{AGENT } \textit{subj}\right]\end{bmatrix}$$

$$\Gamma(\text{transitive control}) = \begin{bmatrix}\text{DOBJ} & + \\ \text{AGENT} & \textit{subj}\end{bmatrix}$$

$$\Gamma(\text{intransitive raising}) = \left[\text{AGENT } \textit{none}\right]$$

$$\Gamma(\text{intransitive equi}) = \left[\text{AGENT } \textit{subj}\right]$$

$$\Gamma(\text{transitive raising}) = \begin{bmatrix}\text{PATIENT} & \textit{none} \\ \text{THEME} & \left[\text{AGENT } \textit{obj}\right]\end{bmatrix}$$

$$\Gamma(\text{transitive equi}) = \left[\text{PATIENT } \textit{obj}\right]$$

$$\Gamma(\text{subject equi}) = \left[\text{THEME } \left[\text{AGENT } \textit{subj}\right]\right]$$

$$\Gamma(\text{object equi}) = \left[\text{THEME } \left[\text{AGENT } \textit{obj}\right]\right]$$

Within this classificatory system we may deduce that, for example, subject equi is described by the description of transitive equi augmented with the idiosyncratic features of subject equi. Similarly we may deduce that transitive equi is described by the description of transitive control (which is in turn described by the description of control augmented with the idiosyncratic features of transitive control) augmented with the idiosyncratic features of transitive equi.

4.2 Default Inheritance

An abstract classification requires no structure on the domain of tokens. In particular, the tokens of a type are not required to "cohere" in any sense. In a classificatory system, on the other hand, coherence between tokens of a type is enforced by treating the *is of type* relation as derivative upon descriptions of tokens and types: for any type t, $\Delta(t)$ gives features common to all tokens of type t. A classificatory system does not allow exceptions: tokens whose descriptions agree on most, but not all, of a type's feature specifications are necessarily not of that type, and types which agree on most, but not all, of some other type's feature specifications are necessarily not subtypes of that type. It may be argued, however, that exceptions are common. Within the hierarchical lexicon, for example, it has proven useful to distinguish "normal" nouns from the expletives 'it' and 'there'. Following GPSG, HPSG achieves this through the (head) feature NFORM, which may take as values *norm*, *it*, or *there*. Most nouns are characterised as bearing the feature specification [NFORM *norm*], but the expletives, which we would still like to say are of type noun, are marked as [NFORM *it*] and [NFORM *there*]. As a second example, consider the class of verbs, whose members normally require a non-expletive nominative noun phrase as their subject. Some verbs, however, such as 'appear', allow expletive subjects (as in 'It appears that Tigger is miaowing'), whilst others, such as 'rain', require an expletive subject (as in 'It rains'), and others still, such as 'bother', allow sentential subjects (as in 'That Tom miaows bothers Tim'). It is unclear whether similar sorts of exceptions also occur within the phrasal hierarchy, though variations from default word order, due to, for example, heavy noun phrase shift, might be regarded as such.

Within inheritance networks the notion of *default inheritance* is often introduced to allow classes to admit exceptions. Nouns may be characterised by default as having the

feature specification [NFORM *norm*]. This default feature is overridden in the cases of the two expletives. Similarly, verbs may be characterised as subcategorising by default for a non-expletive nominative subject noun phrase. [Flickinger *et al.* 85] and [Flickinger 87] develop lexical hierarchies in which default inheritance plays a major role.

The overridding of defaults is a powerful concept. In principle, there is no reason why some token cannot be classified as being an instance of some type even though the token has none of the features normally associated with that type — each of the default features may be overridden. This suggests at the very least that default inheritance should be constrained. A lack of constraint is apparent in the work of [Flickinger 87], who gives an account of the class of auxiliary verbs in which the members are taken by default to subcategorise for a verb phrase complement in base form. This default behaviour is illustrated by the auxiliaries 'will' and 'to':

(2) a. Tom will [miaow]$_{VP[base]}$

b. Tim pretended to [have been miaowing]$_{VP[base]}$

Within the class of auxiliary verbs, however, there are many exceptions to this behaviour. In (2b) 'have' and 'been' are auxiliary verbs which subcategorise for verb phrase complements in other than base form, and the copula commonly occurs with non-verbal predicative complements (as in 'Tom is a cat'). Flickinger treats the classes of the perfective and the copula as exceptions to the default. [Pollard & Sag 87] achieve the same results in HPSG without the use of overridden defaults by subdividing the class of auxiliary verbs according to the form of verb phrase complement for which they subcategorise.

A second unwelcome aspect of default inheritance arises from the interaction of multiple inheritance with defaults. If a type is covered by more than one type, then the type inherits features from more than one source. If the feature structure descriptions inherited are only default descriptions, and not necessary descriptions of tokens of those types, then there is nothing to prevent the descriptions from conflicting. In such cases some form of ordering seems necessary to allow the features of one description to override those of other descriptions with which it conflicts. Note though that this problem does

not arise when the types are hierarchically stru… red as in such cases a type can only ever inherit from one source.

Default inheritance can nevertheless be accomm… ated within the broad framework of classification. Classificatory systems were intr… ced via descriptions of tokens and types, with $\Delta(t)$ being interpreted as giving tho… 'eat…res necessary for a token to be of type t. It is feasible to reinterpret the functi… \ as giving a default description: if $s : t$ then, by default, s will be partially descri… by $\Delta(t)$. Reinterpreting Δ in this way requires that the *is of type* relation be primi… (as in a classification) rather than derivative, and renders incorrect the algorithm … n to section 2.4.2 for determining of which types in a hierarchical classification a … en is an instance. That algorithm requires that each and every instance of a type … each and every feature associated with that type. If the features describing a type … be overridden by individual tokens of that type, then the algorithm will fail.

[Pollard & Sag 87], in their treatment of the hie… nical lexicon as it relates to HPSG, strictly avoid the use (and even the mention) of d… t inheritance. Feature inheritance in HPSG is defined in terms of unification via th… subsumption ordering on the feature structure domain. The overridding of defaults a… to except…ions from this ordering, exceptions which are not catered for by standa… ification. Thus if defaults are to be employed in a system based on feature struct… instances of unification which are to allow defaults to be overridden must be repla… some other operation, such as a form of prioritised unification, where one opera… has precedence over the other. We follow Pollard and Sag in avoiding default inheri… in our fragments, though we note that it may be unavoidable in a full grammar.

5 Law-Like Dependencies of L… 'stic Significance

Grammar rules, lexical rules and phrasal rules … be phrased as law-like depen-dencies which linguistic classifications and class fi… y systems respect. We consider such dependencies in this section.

5.1 Grammar Rules

As mentioned in section 3, when a grammar is viewed as a classification or classificatory system, the rules of the grammar correspond to law-like dependencies that the system respects. A simple phrase structure grammar, for example, might be viewed as a classification having the types noun phrase, verb phrase and sentence. The phrase structure rule:

$$\textsf{sentence} \quad \rightarrow \quad \textsf{noun phrase} \quad \textsf{verb phrase}$$

corresponds to the system respecting the law-like dependency:

$$v_1 : \textsf{noun phrase} \ \wedge \ v_2 : \textsf{verb phrase} \ \Rightarrow \ \mathit{concat}(v_1, v_2) : \textsf{sentence}$$

Furthermore, all tokens of type sentence are licensed by some phrase structure rule, so we have a further law-like dependency which the system respects:

$$v : \textsf{sentence} \ \Rightarrow \ (\mathit{subj}(v) : \textsf{noun phrase} \ \wedge \ \mathit{pred}(v) : \textsf{verb phrase}) \ \vee \ \ldots$$

where the right hand side corresponds to the disjunction of the phrase structure rules which license sentences.

Similarly, in the treatment of a categorial grammar as a classificatory system in section 3.2, the schematic grammar rules

$$X/Y \quad Y \quad \rightarrow \quad X$$

$$Y \quad Y\backslash X \quad \rightarrow \quad X$$

correspond to law-like dependencies.

The grammar rules of HPSG, which may also be seen as schematic in that each rule subsumes several rules of a classical phrase structure grammar, may likewise be phrased in terms of law-like dependencies which a classificatory system respects.

Grammar Rule 1, which licenses phrases of type non-lexically headed phrase, is stated in [Pollard & Sag 87, p. 149] as:

$$\begin{bmatrix} \text{SYN|LOC|SUBCAT} & \langle\,\rangle \\ \text{DTRS} & \begin{bmatrix} \text{HEAD-DTR|SYN|LOC|LEX} & - \\ \text{COMP-DTRS} & \langle [\,] \rangle \end{bmatrix} \end{bmatrix}$$

Sorting information is very important in this and other grammar rules: not only must the description be of sort *sign*, but all daughters must also be described by feature structures of sort *sign*. This leads to some obvious law-like dependencies corresponding to the fact that all subconstituents of any constituent must also be constituents, and the sorting information could instead be replaced by law-like dependencies enforcing this.

Within our hierarchical classificatory system, the relevant law-like dependency may be stated as:

$$v : \textsf{non-lexically headed phrase} \Rightarrow (\mathit{head}_t(v) : \textsf{sign} \wedge \mathit{comp}_t^1(v) : \textsf{sign})$$

The function head_t maps headed phrases to their heads, and is undefined for tokens which are not of type **headed phrase**. The function comp_t^1 maps headed phrases with at least one complement to their first complement. The corresponding functions within the domain of feature structures are:

$$\mathit{head}_{fs}\left(\left[\text{DTRS}|\text{HEAD-DTR}\ \boxed{1}\right]\right) = \boxed{1}$$

$$\mathit{comp}_{fs}^1\left(\left[\text{DTRS}|\text{COMP-DTRS}\ \langle\boxed{1}|\_\rangle\right]\right) = \boxed{1}$$

This law-like dependency simply states that the head of a token of type **non-lexically headed phrase** is a token of type **sign** as is the first complement daughter. This encodes the complete grammar rule, and most of the work is done by the description of the type **non-lexically headed phrase**, which is subsumed by Pollard and Sag's Grammar Rule 1:

$$\Delta(\textsf{non-lexically headed phrase}) \sqsupseteq \begin{bmatrix} \text{SYN}|\text{LOC}|\text{SUBCAT} & \langle\,\rangle \\ \text{DTRS} & \begin{bmatrix} \text{HEAD-DTR}|\text{SYN}|\text{LOC}|\text{LEX} & - \\ \text{COMP-DTRS} & \langle[\,]\rangle \end{bmatrix} \end{bmatrix}$$

Similarly, Grammar Rule 2 and Grammar Rule 3 give the following law-like dependencies:

$$v : \textsf{head/non-subject phrase} \Rightarrow \mathit{head}_t(v) : \textsf{sign}$$

$$v : \textsf{head/non-subject phrase} \Rightarrow \mathit{comp}_t^n(v) : \textsf{sign}$$

$$v : \textsf{inverted phrase} \Rightarrow \mathit{head}_t(v) : \textsf{sign}$$

$$v : \textsf{inverted phrase} \Rightarrow \mathit{comp}_t^n(v) : \textsf{sign}$$

where $comp_t^n$ maps phrases consisting of a head with at least n complement daughters to their n^{th} complement daughter.

As the classificatory system requires that head/non-subject phrase and inverted phrase partition lexically headed phrase, these rules may be merged to give:

$$v : \textsf{lexically headed phrase} \Rightarrow head_t(v) : \textsf{sign}$$

$$v : \textsf{lexically headed phrase} \Rightarrow comp_t^n(v) : \textsf{sign}$$

Furthermore, lexically headed phrase and non-lexically headed phrase partition head/comp phrase, so we may merge

$$v : \textsf{lexically headed phrase} \Rightarrow head_t(v) : \textsf{sign}$$

and

$$v : \textsf{non-lexically headed phrase} \Rightarrow head_t(v) : \textsf{sign}$$

to get

$$v : \textsf{head/comp phrase} \Rightarrow head_t(v) : \textsf{sign}$$

In fact, the heads of all headed phrases must be of type sign, so more generally still we have:

$$v : \textsf{headed phrase} \Rightarrow head_t(v) : \textsf{sign}$$

Similarly, the complements of all tokens of type head/comp phrase are of type sign:

$$v : \textsf{head/comp phrase} \Rightarrow comp_t^n(v) : \textsf{sign}$$

This merging of law-like dependencies corresponding to grammar rules is only possible because of the hierarchical structuring of the domain of constituents and the inheritance mechanism between types and their dominating supertypes. This is one of the key motivations for CPSG's constituent hierarchy: it provides a natural mechanism to capture generalisations (such as the above) across grammar rules.

5.2 Lexical Rules

Lexical rules capture relationships between individual words with the same stem. Within a classification, for example, 'persuaded' might be classified as being of type past participle and 'persuading' as being of type present participle. Lexical rules capture the

relationship between these verbs and their base form. In terms of a classification or classificatory system, lexical rules correspond to law-like dependencies.

That lexical rules can be stated so as to relate base forms of words to inflected forms suggests the possibility of classifying only base forms in a "base" lexicon, and closing this lexicon under the action of the lexical rules. One advantage of this is that the need for cross-classification becomes less apparent — as stated in section 1.3, the need for cross-classification seems to arise principally from the classification of inflected forms.

As an example of a lexical rule, consider *passivisation* (see, for example, [Bresnan 82b] for passivisation with respect to LFG, or [Pollard & Sag 87] for passivisation with respect to HPSG). In informal terms we may relate the base forms of verbs to their passive forms as follows:

- the phonology of the passive form of a verb is given by the action of the function *pas* on the base form of that verb (this function maps 'chase' to 'chased', 'bite' to 'bitten', etc.),
- the subject of the passive fills the same argument role as the object of the base form,
- the passive subcategorises for an optional complement marked by the preposition 'by', which if present fills the argument role corresponding to the subject of the base form, and
- the passive subcategorises for all other complements which the base form subcategorises for.

In terms of a classificatory system, this rule may be formalised as a law-like dependency which the system respects:

$$v : \mathsf{base} \Rightarrow pas_t(v) : \mathsf{passive}$$

pas_t is a partial function from verbs in base form to their passive forms. It is partial as modal auxiliaries, for example, have no passive forms.

pas_{fs} is given by:

$$pas_{fs}\left(\begin{bmatrix}\text{PHONOLOGY} & \boxed{1}\\ \text{SYN|LOC|SUBCAT} & \boxed{2}\end{bmatrix}\right) = \begin{bmatrix}\text{PHONOLOGY} & pas(\boxed{1})\\ \text{SYN|LOC|SUBCAT} & passivise(\boxed{2})\end{bmatrix}$$

where *passivise* carries out the appropriate action on the value of the SUBCAT attribute.

All other features of the descriptions of passive verbs (HEAD features, LEX features and NONLOCAL features) are inherited from the type passive as $\delta(pas_t(v)) \sqsupseteq \Delta(\text{passive})$.

Viewing lexical rules as declarative relations (rather than as procedural operations) suggests that the corresponding law-like dependencies have a bi-conditional nature: if $pas_t(v)$ is the passive form of the base verb v, then $pas_t^{-1}(v')$ is the base form of the passive verb v'. Law-like dependencies corresponding to lexical rules thus come in pairs:

$$v : \text{base} \Rightarrow pas_t(v) : \text{passive}$$

$$v : \text{passive} \Rightarrow pas_t^{-1}(v) : \text{base}$$

Another lexical rule relates the present participle forms of verbs to their base forms: if s is a token of type base, then $prp(s)$ is a token of type present participle:

$$v : \text{base} \Rightarrow prp_t(v) : \text{present participle}$$

where

$$prp_{fs}\left(\begin{bmatrix} \text{PHONOLOGY} & \boxed{1} \\ \text{SYN|LOC|SUBCAT} & \boxed{2} \end{bmatrix}\right) = \begin{bmatrix} \text{PHONOLOGY} & prp(\boxed{1}) \\ \text{SYN|LOC|SUBCAT} & \boxed{2} \end{bmatrix}$$

Again HEAD features, LEX features and NONLOCAL FEATURES of the result of prp_{fs} are inherited from the description of the type present participle. This ensures that if s is of type strict intransitive, then $prp_t(s)$ is also of type strict intransitive, and similarly for all subtypes of main and auxiliary. We do thus not need to state dependencies such as:

$$v : \text{strict intransitive} \Rightarrow prp(v) : \text{strict intransitive}$$

$$v : \text{strict transitive} \Rightarrow prp(v) : \text{strict transitive}$$

$$v : \text{strict ditransitive} \Rightarrow prp(v) : \text{strict ditransitive}$$

$$\vdots$$

Other lexical rules include rules which relate singular and plural forms of nouns, base and other inflected forms of verbs, causative verbs to corresponding intransitive verbs, pairs of dative forms, intransitive verbs to prenominal modifiers, and pairs of raising verbs.

Having said this, lexical rules in CPSG are not formulated as law-like dependencies. The structure of the lexicon within CPSG is discussed in Chapter 9.

5.3 Phrasal Rules

Phrasal rules are the phrasal analogues of lexical rules: phrasal rules relate various forms of phrases in the same way as lexical rules relate various forms of words. Phrasal rules may be posited to capture, for example, the relationship between "canonical phrases" and the corresponding inverted phrases. Although this is effectively achieved at the lexical level in CPSG via a lexical rule, we consider such a phrasal rule below. Phrasal rules may also be posited to capture the relationship between phrases involving "extraction" or "movement" and "canonical phrases".

Transformational grammar originally derived question sentences from base form sentences via a process termed *subject-auxiliary inversion*, whereby a question sentence is derived from the corresponding base form sentence by permuting the subject of the base form and the first auxiliary in its verbal chain (see, for example, [Akmajian & Heny 75]). Such a rule allows, for example, 'Did Tom miaow' to be derived from 'Tom did miaow' and 'Will Tigger be hungry' to be derived from 'Tigger will be hungry'.

This rule may be stated as a law-like dependency:

$$v : \textsf{base sentence} \Rightarrow \mathit{invert}_t(v) : \textsf{inverted sentence}$$

where the function invert_t maps base sentences to their inverted forms.

Analysing inverted phrases in these terms need not commit us to a transformational approach where inverted phrases are derived from canonical forms. Canonical phrasal forms have exactly the same status as base lexical forms. Phrasal rules merely relate various forms, and so have corresponding inverse forms just as lexical rules:

$$v : \textsf{inverted sentence} \Rightarrow \mathit{invert}_t^{-1}(v) : \textsf{base sentence}$$

Within HPSG, inverted phrases are admitted by Grammar Rule 3. Within the hierarchy of section 3.3 they are classified as being of type inverted phrase. Base form sentences are classified as being of type non-lexically headed phrase. The phrasal rule of inversion thus takes the form:

$$v : \textsf{non-lexically headed phrase} \Rightarrow \mathit{invert}_t(v) : \textsf{inverted phrase}$$

$invert_{fs}$ may be defined as:

$$invert_{fs}\left(\left[\begin{array}{ll}\text{PHONOLOGY} & \boxed{1} \\ \text{DTRS|HEAD-DTR|DTRS|HEAD-DTR} & \boxed{2}\end{array}\right]\right) = \left[\begin{array}{ll}\text{PHONOLOGY} & invert(\boxed{1}) \\ \text{DTRS|HEAD-DTR} & \boxed{2}\end{array}\right]$$

This ensures that the head of the head of the base sentence (normally the first auxiliary) is the head of the inverted sentence. The subcategorisation requirements of the auxiliary and the other grammar rules interact with this information and the typing constraints applied by the law-like dependency (that v should be of type non-lexically headed phrase and that $invert_t(v)$ should be of type inverted phrase) to correctly determine the description of inverted structures from their corresponding base forms.

6 Conclusion

In this chapter we have illustrated how the lexical hierarchy may be phrased in terms of a classificatory system. This led us to the notion of a hierarchical classificatory system, in which feature inheritance may be employed to efficiently encode the descriptions associated with types. We also considered how, by extrapolating the structure on the lexical domain to the phrasal domain, the lexical hierarchy may be taken as forming one branch of a larger constituent hierarchy, with a phrasal hierarchy forming another branch. In CPSG this constituent hierarchy is postulated as a linguistic universal. Finally, we illustrated how lexical rules, grammar rules and phrasal rules may be phrased as law-like dependencies which linguistic classifications respect.

Chapter 3

Specifiers, Complements and Adjuncts

We begin our examination of syntax within CPSG with the fragment of English covered by HPSG in [Pollard & Sag 87]. The HPSG fragment includes four grammar rules. Grammar Rule 1 yields the equivalent of our head/specifier phrases, Grammar Rule 2 yields the equivalent of our head/complement phrases, and Grammar Rule 4 yields head/adjunct phrases. In this chapter we discuss each of these in turn. The effect of Grammar Rule 3, which licences inverted phrases, is achieved in CPSG via a combination of processes, which we also discuss.

Before examining each of the above types of phrases, we highlight some important differences between the feature structure descriptions employed by HPSG and CPSG.

1 $\overline{\text{X}}$-Syntax in CPSG

In HPSG as presented in [Pollard & Sag 87], subcategorisation is treated via the SUBCAT attribute which takes as its value a (possibly empty) list of feature structures. Within HPSG the elements that a head subcategorises for are referred to as complements, and each feature structure on this list corresponds to one such complement. If for some token the SUBCAT attribute is the empty list, then that token is fully saturated. In the case of intransitive verbs, for example, which require only a subject noun phrase to become saturated, the SUBCAT list is a one element list containing a feature structure describing

a prototypical subject noun phrase. For transitive verbs, the SUBCAT list contains two elements, corresponding to the subject and direct object noun phrases. The SUBCAT list of a common noun contains one element corresponding to a determiner. Elements on the SUBCAT list are ordered according to obliqueness, with least oblique elements last. Thus direct objects are ordered before subjects but after indirect objects, and so on.

HPSG's use of a list to represent all subcategorisation information does not distinguish significantly between subjects and other complements. [Borsley 87] presents several arguments for such a distinction. These include facts based on unbounded dependencies, the inability of the framework to represent verb phrases and sentences as a natural class, and problems with the word order in non-predicative prepositional phrases. In addition to these arguments we point out that the phrase structure rules of HPSG distinguish between the last element of the SUBCAT list and all others, but not between any other element: Grammar Rule 2 combines a head with all but the last element on its SUBCAT list, and Grammar Rule 1 combines a head resulting from Grammar Rule 2 with its final complement. For these reasons we follow Borsley and replace the SUBCAT attribute of [Pollard & Sag 87] by individual attributes for specifiers (the SPECIFIER attribute, usually abbreviated to SPEC) and other complements (the COMPLEMENTS attribute, usually abbreviated to COMPS). [Pollard 89a] also adopts this distinction. We use the phrase "syntactic argument" to include both complements and specifiers and reserve the word "complement" for those syntactic arguments which are not specifiers.

Within CPSG, the value of the LOCAL attribute is thus a feature structure of schematic form:

$$\begin{bmatrix} \text{HEAD} & \ldots \\ \text{SPEC} & \ldots \\ \text{COMPS} & \ldots \end{bmatrix}$$

where ellipses indicates further as yet unspecified structure.

The value associated with the attribute SPEC is either a feature structure describing a constituent or the atomic value *null*. The value of the attribute COMPS is either a (possibly empty) multi-set of feature structures describing constituents, or the atomic value *null*.

Note that complements are not ordered as they are in HPSG — we employ a multi-set rather than a list to describe them. Consequently, complements cannot be distinguished by their position on the SUBCAT list as they can in HPSG. It is, of course, though necessary to distinguish complements. As (1) illustrates, we must be able to distinguish complements so that their semantic content can be associated with the correct argument roles of the head which subcategorises for them. In (1a), the complement 'it' fills the goal argument role and 'the cat' fills the patient argument role, whereas in (1b) this situation is reversed.

(1) a. ... gives it the cat

b. ... gives the cat it

In cases such as (1), it is the grammatical functions which the complements bear to the phrase in which they occur which distinguish them (and this grammatical function is determined by, or reflected in, the word order). We thus introduce an attribute GRAMMATICAL-FUNCTION (usually abbreviated to GRAM-FN), whose value is an atom specifying the grammatical function that the constituent described by the feature structure bears in its immediately super-ordinate phrase. Lexical entries specify the grammatical functions which their arguments bear in phrases which they head. Thus, a ditransitive verb, which subcategorises for a specifier and two NP complements, will specify that the specifier is the subject and that one of the complements is the direct object whilst the other is the indirect object. For a concrete example, consider the partial lexical entry for 'gives':

$$\begin{bmatrix} \text{PHON} & \langle \textit{gives} \rangle \\ \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{Verb} \end{bmatrix} \\ \text{SPEC} & \text{NP[nom, subj]}\boxed{1} \\ \text{COMPS} & \left\{ \text{NP[acc, obj]}\boxed{2}, \text{NP[acc, ind-obj]}\boxed{3} \right\} \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{CONTENT} & \begin{bmatrix} \text{RELN} & \textit{give} \\ \text{AGENT} & \boxed{1} \\ \text{PATIENT} & \boxed{2} \\ \text{GOAL} & \boxed{3} \end{bmatrix} \end{bmatrix} \\ \text{DTRS} & \textit{null} \end{bmatrix}$$

NP[x,y] abbreviates the feature structure description for an NP constituent whose SYN|LOC|HEAD|CASE attribute has value x and whose SYN|LOC|GRAM-FN attribute has

value y. The subscripted indices represent the value of the SEMANTIC|CONTENT attribute of such abbreviations. For full details of the abbreviations used, see section 3 of Appendix A.

We take GRAM-FN to be a local attribute. Unsaturated lexical constituents serve as heads to phrases and are consequently marked as [SYN|LOC|GRAM-FN *head*]: such constituents fill the *head* grammatical function in phrases which they form. As the arguments which heads subcategorise for are saturated, these saturated elements will receive their grammatical function from their sister head (i.e., the head which subcategorises for them). Consequently, the propagation of grammatical functions proceeds as in Figure 3.1, where the Greek labels on the arrows correspond to variables over grammatical functions, and only phrasal nodes which are not dominated lack a grammatical function.

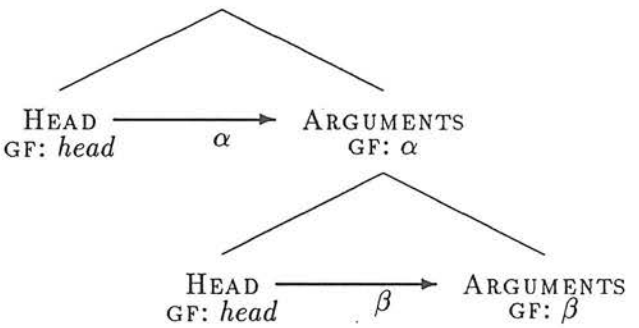


Figure 3.1: The Propagation of Grammatical Functions

All grammar rules licensing headed phrases are such that they mark the heads of the phrases licensed as bearing the grammatical function *head*.

Whilst we do not investigate the possibility in this thesis, grammatical functions may also be of relevance in a treatment of anaphoric binding. In HPSG, anaphoric binding is constrained via a notion of "obliqueness-command", or O-command ([Pollard & Sag ∞a]). In HPSG this is defined in terms of the total ordering on a head's arguments. As this ordering is not present in CPSG, O-command cannot be defined in this way, but the notion could be reconstructed in CPSG from grammatical function information.

The motivation for employing a multi-set rather than a list to describe complements comes from the fact that whilst the obliqueness hierarchy might provide an ordering

on complements, it is not obvious that that ordering is *total*, as required by the list representation. As a potential example of when complements cannot be ordered with respect to each other, consider (2). Given an analysis where the two optional PPs are regarded as complements, HPSG would require that those complements be ordered with respect to each other. CPSG does not impose this requirement. In fact, there is no reason why several complements may not bear the same grammatical function in the phrase in which they occur. This is how we treat the case of 'argues'. Note though of course that each subconstituent of a constituent may only bear one grammatical function.

(2) a. ... argues with Tigger about Fido

b. ... argues about Fido with Tigger

The lexical entry for 'argues' is partially given by:

$$\left[\begin{array}{ll}
\text{PHON} & \langle \textit{argues} \rangle \\
\text{SYN} & \left[\begin{array}{ll}
\text{LOC} & \left[\begin{array}{ll}
\text{HEAD} & \left[\text{MAJ} \quad \textit{Verb}\right] \\
\text{GRAM-FN} & \textit{head} \\
\text{SPEC} & \text{NP[nom, subj]}\boxed{1} \\
\text{COMPS} & \left\{\text{PP[with, pp-obj]}\boxed{2}, \text{PP[about, pp-obj]}\boxed{3}\right\}
\end{array}\right]
\end{array}\right] \\
\text{SEM} & \left[\begin{array}{ll}
\text{CONTENT} & \left[\begin{array}{ll}
\text{RELN} & \textit{argues} \\
\text{AGENT} & \boxed{1} \\
\text{PATIENT} & \boxed{2} \\
\text{THEME} & \boxed{3}
\end{array}\right]
\end{array}\right] \\
\text{DTRS} & \textit{null}
\end{array}\right]$$

Note that in CPSG all set-valued attributes are multi-sets. We interpret multi-sets of feature structures as such that each element of the multi-set describes a different token. Thus a multi-set of three feature structures describes three tokens. This interpretation of *multi-sets* of feature structures is different to the interpretation of *sets* of feature structures employed by [Pollard & Sag 87] and [Pollard & Moshier 89], where several elements of the set may describe the same token.

The use of separate SPEC and COMPS attributes allows a fairly direct encoding of $\overline{\text{X}}$-

syntax. A constituent is of category $\overline{\overline{X}}$ iff its description extends:

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{SPEC} & \textit{null} \\ \text{COMPS} & \textit{null} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

A constituent is of category $\overline{X}$ iff its description extends:

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{SPEC} & [\ ] \\ \text{COMPS} & \textit{null} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

A constituent is of category X iff its description extends:

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{SPEC} & [\ ] \\ \text{COMPS} & \{\ldots\} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Note that this does not force constituents of category X to subcategorise for some complements: a constituent of category X may be specified as [SYN|LOC|COMPS { }], as in the case of intransitive verbs.

The above encoding allows for one more logical possibility, namely constituents whose description extends:

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{SPEC} & \textit{null} \\ \text{COMPS} & \{\ldots\} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Such descriptions are suitable for non-predicative prepositions, as well as adverbs and adjectives and, as illustrated in section 5, auxiliaries heading inverted structures. When necessary we use subscripts to indicate subcategorisation requirements as illustrated in Table 3.1. Under this scheme, non-predicative prepositions, adverbs, adjectives, and inverted auxiliaries are lexically specified as being of category Z_{c*}.

Note that the use of SPEC and COMPS attributes renders the LEXICAL attribute redundant: constituents of category $\overline{X}$ and $\overline{\overline{X}}$ cannot be lexical. CPSG does not employ such an attribute.

A final note concerns the representation of phrase structure trees in CPSG. As in HPSG, phrase structure trees are described in terms of feature structures via the DAUGHTERS attribute. Lexical constituents are described by feature structures bearing the specification

$\overline{X}$ Syntax	CPSG Syntax
$\overline{\overline{X}}$	Z
$\overline{X}$	Z_s
	Z_{c^*}
X	Z_{s,c^*}

Table 3.1: A Comparison of Schematic Categories.

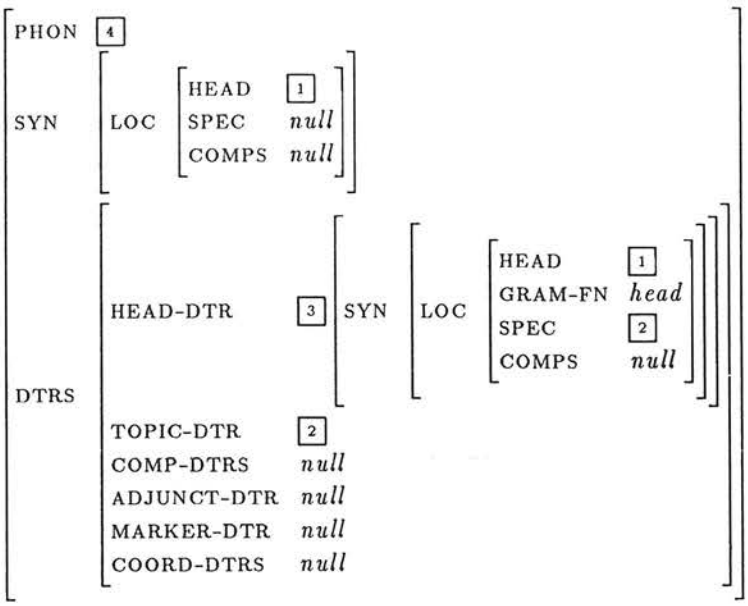
[DAUGHTERS *null*]. For a phrasal constituent, the value of the DAUGHTERS attribute is a feature structure having attributes for each possible type of daughter. These include head daughters, specifier daughters, complement daughters, adjunct daughters, filler daughters and coordinate daughters. Phrasal constituents may thus be schematically described as:

$$\left[\begin{array}{ll} \text{DAUGHTERS} & \left[\begin{array}{ll} \text{HEAD-DTR} & \ldots \\ \text{TOPIC-DTR} & \ldots \\ \text{COMP-DTRS} & \ldots \\ \text{ADJUNCT-DTR} & \ldots \\ \text{MARKER-DTR} & \ldots \\ \text{COORD-DTRS} & \ldots \end{array}\right] \end{array}\right]$$

Any one phrase will not normally contain more than two types of daughters. The grammar rules will ensure that in general most of these attributes take the value *null*.

2 Head/Specifier Phrases

Grammar Rule 1 of [Pollard & Sag 87] may be translated directly into CPSG. Any constituent of type head/specifier phrase is described by an extension of:



where *order-phonology*([4],{[2],[3]})

and *constituent*([3])

and *constituent*([2])

This feature structure encodes what might be written in $\overline{\text{X}}$-syntax as:

$$\overline{\overline{\text{X}}} \rightarrow \text{Spec}_{\text{x}} \ \overline{\text{X}}$$

This subsumes phrase structure rules such as:

$$\overline{\overline{\text{N}}} \rightarrow \text{Det} \ \overline{\text{N}}$$

and

$$\overline{\overline{\text{V}}} \rightarrow \overline{\overline{\text{N}}} \ \overline{\text{V}}$$

An example of a head/specifier phrase is given in Figure 3.2.

Note that the specification [SYN|LOC|COMPS *null*] on the head daughter of any constituent of type **head/specifier phrase** ensures that the head daughter is non-lexical. (All major lexical constituents bear the specification [SYN|LOC|COMPS x] where x is a (possibly empty) multi-set of feature structures.) Furthermore, that the value of the TOPIC-DTR attribute is required to describe a constituent (and thus cannot be *null*) ensures that the rule does not license local phrases of the form:

$$\overline{\overline{\text{X}}} \rightarrow \overline{\overline{\text{X}}}$$

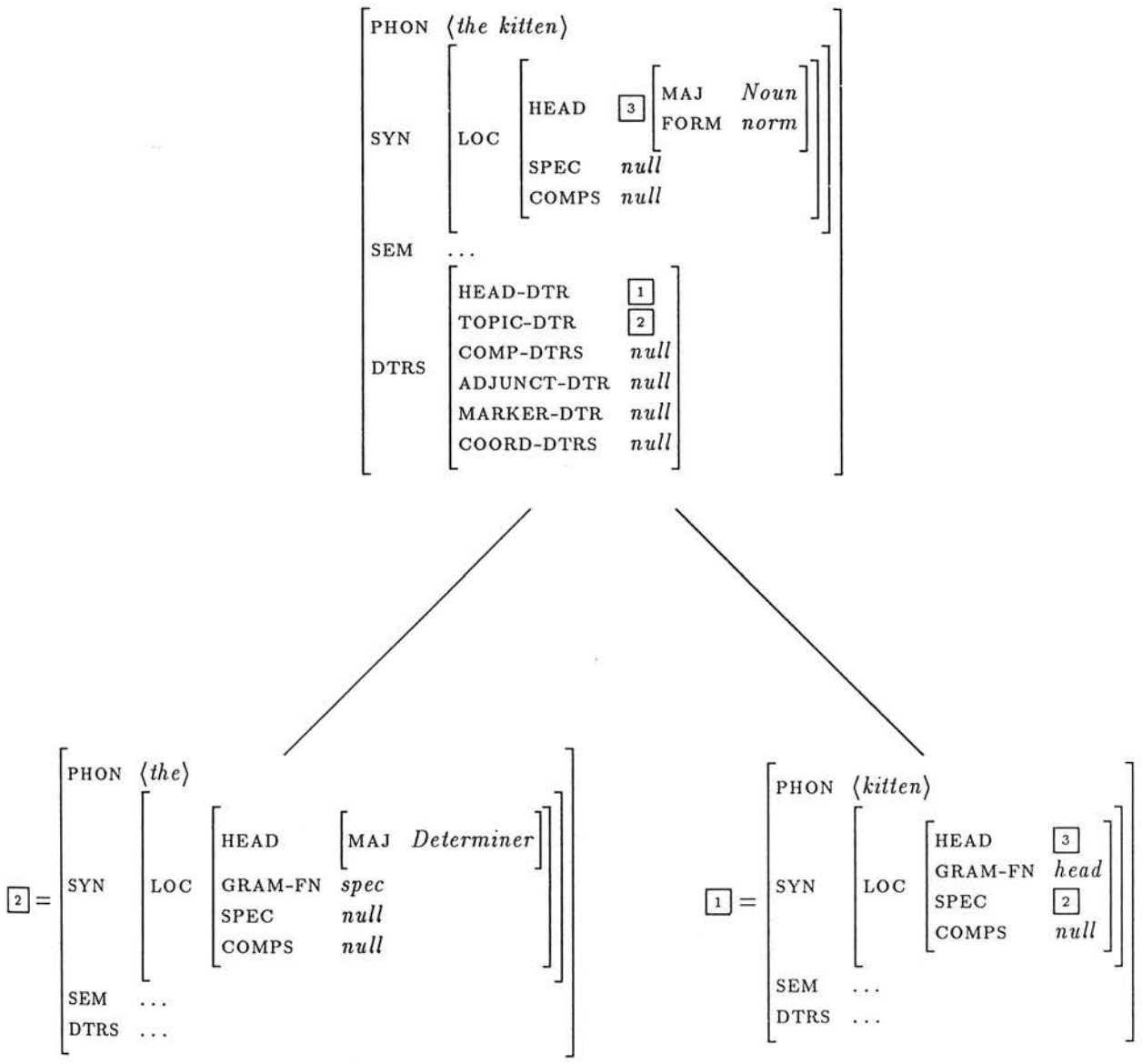


Figure 3.2: An Example Head/Specifier Phrase

Word order is achieved through the relation *order-phonology*, which relates lists of words to multi-sets of descriptions of constituents. Note that complete feature structure descriptions of constituents are available to *order-phonology*, rather than just the corresponding values of the PHONOLOGY attribute, and so this relation can refer to any information in the description of constituents when ordering those constituents. In particular, it may refer to the GRAM-FN attribute. For the case of English, *order-phonology* orders (non-lexical) heads after their specifiers, implementing the corresponding linear precedence rule. When the phrase in question involves only one argument of the head, as in the present case, the relation is functional, and could be expressed directly in terms of a list concatenation function. We may thus give the relevant case of *order-phon*:

$$\textit{order-phon}\left(\boxed{1}\,\hat{}\,\boxed{2},\left\{\begin{bmatrix}\text{PHON} & \boxed{2}\\ \text{SYN|LOC|GRAM-FN} & \textit{head}\end{bmatrix},\begin{bmatrix}\text{PHON} & \boxed{1}\\ \text{SYN|LOC|GRAM-FN} & \textit{subj}\end{bmatrix}\right\}\right)$$

The symbol '^' indicates list concatenation.

Note that it is important that all sub-constituents of a phrase are specified as having a grammatical function in the immediate super-ordinate phrase. In this case, without such a function, it would be non-trivial distinguishing the head from the subject.

3 Head/Complement Phrases

Grammar Rule 2 of [Pollard & Sag 87] may also be translated directly into CPSG. Any constituent of type head/complement phrase is described by an extension of:

$$\begin{bmatrix}
\text{PHON} & \boxed{5}\\
\text{SYN} & \begin{bmatrix}\text{LOC} & \begin{bmatrix}\text{HEAD} & \boxed{1}\\ \text{SPEC} & \boxed{2}\\ \text{COMPS} & \textit{null}\end{bmatrix}\end{bmatrix}\\
\text{DTRS} & \begin{bmatrix}
\text{HEAD-DTR} & \boxed{4}\begin{bmatrix}\text{SYN} & \begin{bmatrix}\text{LOC} & \begin{bmatrix}\text{HEAD} & \boxed{1}\\ \text{GRAM-FN} & \textit{head}\\ \text{SPEC} & \boxed{2}\\ \text{COMPS} & \boxed{3}\end{bmatrix}\end{bmatrix}\\ \text{DTRS} & \textit{null}\end{bmatrix}\\
\text{TOPIC-DTR} & \textit{null}\\
\text{COMP-DTRS} & \boxed{3}\\
\text{ADJUNCT-DTR} & \textit{null}\\
\text{MARKER-DTR} & \textit{null}\\
\text{COORD-DTRS} & \textit{null}
\end{bmatrix}
\end{bmatrix}$$

where *order-phonology*(5,{4} ⊎ 3)

and *constituent*(4)

and *set-of-constituents*(3)

The function ⊎ maps a pair of multi-sets to their disjoint union.

This feature structure encodes what might be written in $\overline{X}$-syntax as:

$$\overline{X} \rightarrow X \quad Comp^*$$

Instances of the rule include:

$$\overline{N} \rightarrow N$$

for common nouns,

$$\overline{N} \rightarrow N \quad PP[of]$$

for "picture" nouns,

$$\overline{V} \rightarrow V$$

for intransitive verbs,

$$\overline{V} \rightarrow V \quad NP$$

for transitive verbs,

and so on.

The rule does not require that the SPEC attribute of the head be *null*. Hence it also licenses phrases of the form

$$Z \rightarrow Z_{c^*} \quad Comp^*$$

Non-predicative prepositional phrases are licensed by an instance of this rule:

$$PP \rightarrow Prep \quad NP$$

Our analysis of subject/auxiliary inversion also employs this rule.

An example of a head/complement phrase is give in Figure 3.3.

The unary relation *set-of-constituents* ensures that 3 is a (possibly empty) multi-set of feature structure descriptions of constituents, rather than the atomic value *null*. Hence this rule does not license local phrase structures of the form:

$$\overline{X} \rightarrow \overline{X}$$

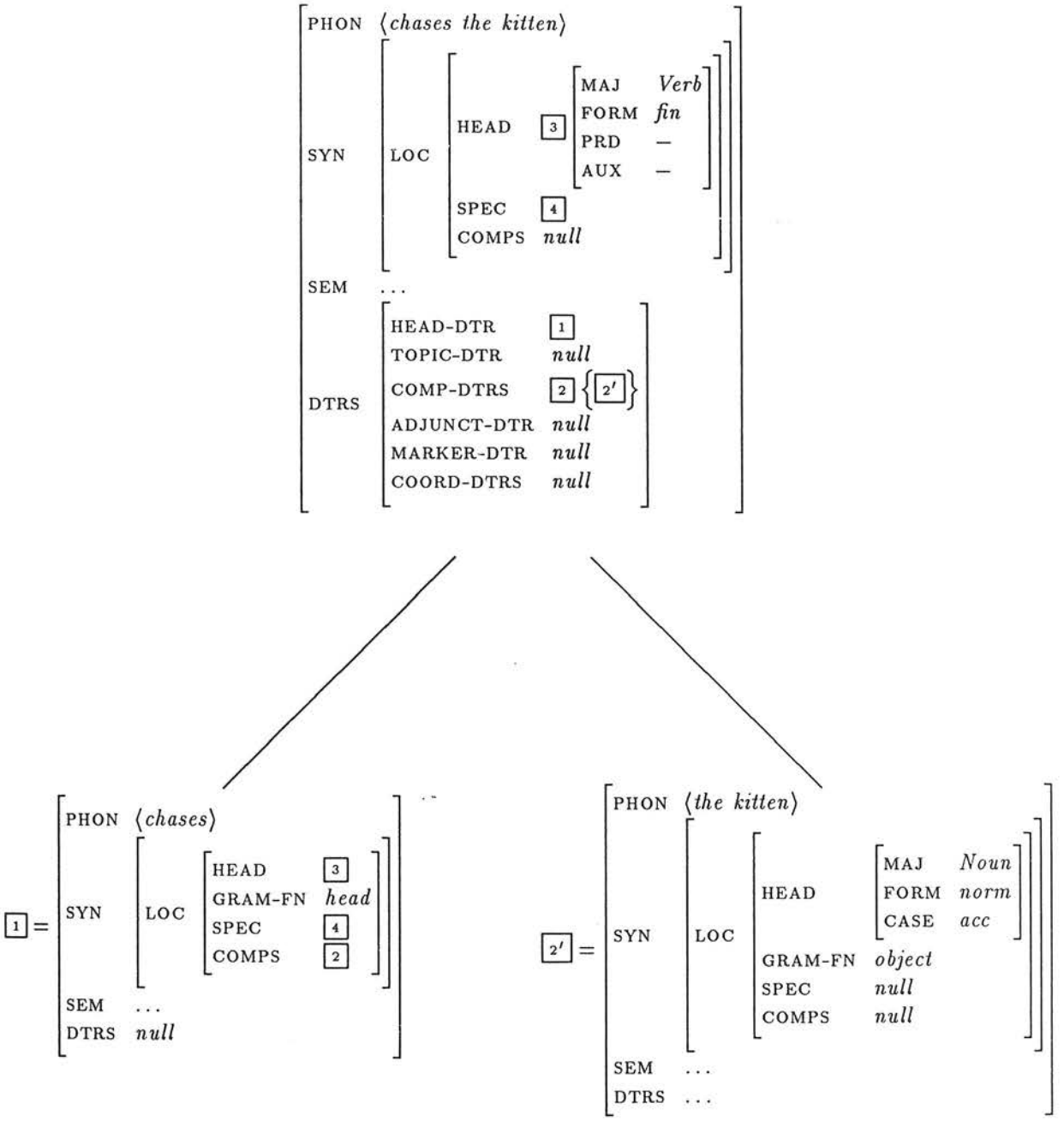


Figure 3.3: An Example Head/ComplementPhrase

Furthermore, that the head daughter is specified as [DTRS *null*] ensures that the head is lexical.

Again *order-phonology* governs the word order. In this case, the specification for English requires that the (lexical) head precede its complements. The relative ordering of complements is, in general, dependent on their grammatical functions.

Treating complements in terms of multi-sets, rather than lists, is a major departure from HPSG and linguistic frameworks such as Categorial Grammar which impose a total order on complements. The decision to employ multi-sets rather than lists is motivated by the fact that it is not always possible to motivate a *total* order on complements. As explained in section 1, whilst the obliqueness hierarchy as employed by HPSG might motivate a *partial* ordering on complements, it is far from clear that it may motivate a total ordering. A further argument against an ordered list of complements stems from the disparity that it gives rise to between syntax and semantics. Unlike functional semantic formalisms such as those of Montague ([Montague 70], [Montague 73]), the unification based semantic formalism which both HPSG and CPSG employ does not require or suggest any semantic ordering on arguments. It thus seems unnatural to force a syntactic ordering on a head's arguments.

In the case of ordering complements, *order-phonology* again makes crucial reference to the grammatical functions that the complements bear in the phrase. As a head marks each of its arguments (i.e., its specifier and any complements) with the grammatical function which those arguments bear to the super-ordinate phrase (via the attribute SYN|LOC|GRAM-FN), *order-phonology* can effectively implement a theory of linear precedence based on grammatical relations such as that outlined in [Sag 86].

The specification of *order-phonology* must ensure that heads follow their specifiers but precede their complements. Thus it must be faithful to the (transitive closure of the) following linear precedence rules:

$$\text{subject} \prec \text{object} \prec \text{indirect object} \prec \text{vcomp}$$

Although we have treated *order-phonology* as a relation, in English it is, for the most part, functional. Only in examples such as (2), where several complements bear the same

grammatical relation, is the relational nature of *order-phonology* clearly demonstrated. Note though that if we were to treat phonology in terms of posets rather than lists, as suggested by Mike Reape (p.c.), then full functionality could be achieved.

4 Interlude: Specifiers and Complements

4.1 Schematic Phrase Structure

The above treatment of specifiers and complements results in the schematic phrase structure of Figure 3.4. This structure is most commonly instantiated with X as N or V.

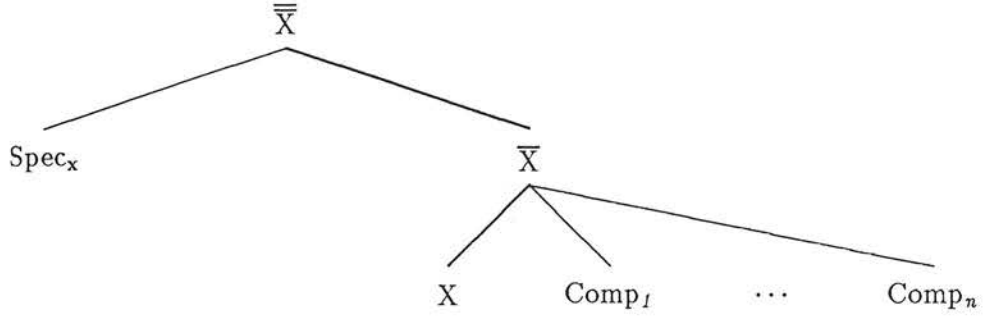


Figure 3.4: Schematic Phrase Structure

That lexical items need not subcategorise for a specifier (i.e., that their category may be an instantiation of Z_{c*}) means that lexical items may form fully saturated phrasal projections according to a second schemata, that illustrated in Figure 3.5. Common instantiations of this schemata include non-predicative prepositional phrases and inverted clauses (see section 5).

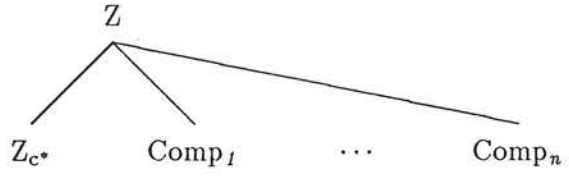


Figure 3.5: Schematic Phrase Structure

4.2 Grammar Principles

The grammar principles of HPSG are principles which hold of all phrases of a language. In this sense they are abstractions across all phrases of a language. [Pollard & Sag 87] cite four such principles: the Head Feature Principle, the Subcategorisation Principle, the Semantics Principle, and the Adjuncts Principle. Here we are concerned with the Subcategorisation Principle and the Head Feature Principle.

4.2.1 The Subcategorisation Principle

[Pollard & Sag 87] state the Subcategorisation Principle as:

$$\left[\text{DTRS}\ \textit{headed-structure}\left[\ \right]\right] \Rightarrow \left[\begin{array}{ll} \text{SYN} & \left[\text{LOC}\ \left[\text{SUBCAT}\ \boxed{2}\right]\right] \\ \text{DTRS} & \left[\begin{array}{ll} \text{HEAD-DTR} & \left[\text{SYN}\ \left[\text{LOC}\ \left[\text{SUBCAT}\ \text{append}(\boxed{1},\boxed{2})\right]\right]\right] \\ \text{COMP-DTRS} & \boxed{1} \end{array}\right] \end{array}\right]$$

The conditional prevents this principle from having any force on phrases which are not headed (such as coordinate structures). The Subcategorisation Principle might thus be better seen as an abstraction across headed phrases. However, given that the principle concerns the combination of heads with their arguments, it seems more correct to treat the principle as an abstraction across instances of head/argument phrase.

Given CPSG's division of arguments into complements and specifiers, there seems to be no elegant way in which the Subcategorisation Principle of HPSG may be stated within CPSG. However, it might be argued that the generalisation contained in HPSG's Subcategorisation Principle is invalid. In particular, HPSG's general machinery provides no arguments against structures such as that in Figure 3.6: they are perfectly consistent with the generalisation on which the Subcategorisation Principle is based. The use of separate SPEC and COMP attributes within CPSG argues against such structures, and the subcategorisation generalisation which might condone them.

CPSG effectively makes use of two instances of the Subcategorisation Principle, one in head/specifier phrases:

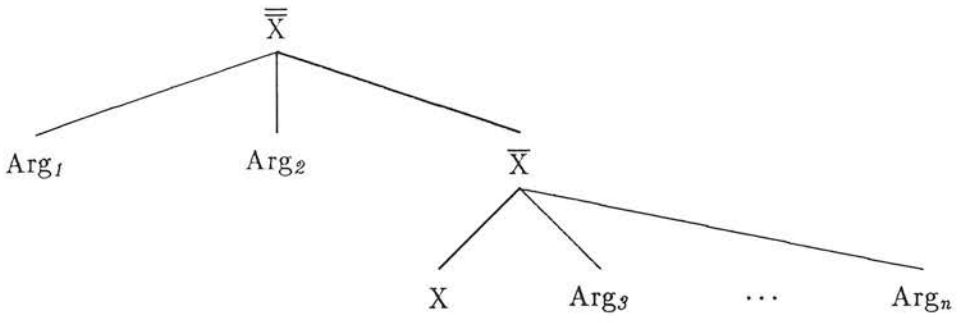


Figure 3.6: A Possible Schematic Phrase Structure in HPSG

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{SPEC} & \boxed{1} \end{bmatrix} \end{bmatrix} \\ \text{DTRS} & \begin{bmatrix} \text{TOPIC-DTR} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

and one in head/complement phrases:

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{COMPS} & \boxed{1} \end{bmatrix} \end{bmatrix} \\ \text{DTRS} & \begin{bmatrix} \text{COMP-DTRS} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Neither of these suggest the generalisation to HPSG's Subcategorisation Principle.

As discussed below, the third instance of the Subcategorisation Principle employed in [Pollard & Sag 87], that implicated in the treatment of inverted phrases, is not employed by our system, which treats inversion as a lexical process.

4.2.2 The Head Feature Principle

The Head Feature Principle states that all headed phrases share their head features with their head daughter. This is thus an abstraction across headed phrases, rather than all phrases: the Head Feature Principle has no force over coordinate phrases, which are not headed. The Principle does have force, however, over head/specifier phrases and head/complement phrases, as well as all other headed phrases in the phrasal hierarchy of CPSG. Within CPSG, the Head Feature Principle is stated in terms of the feature structure description associated with the node **headed phrase**. All instances of subtypes of this type inherit the property of sharing their head features with their head daughter's through the hierarchy. We thus have:

$$\Delta(\text{headed phrase}) \sqsupseteq \left[\begin{array}{ll} \text{SYN} & \left[\text{LOC}\ \left[\text{HEAD}\ \boxed{1}\right]\right] \\ \text{DTRS} & \left[\text{LOC}\ \left[\text{HEAD-DTR}\ \left[\text{SYN}\ \left[\text{LOC}\ \left[\text{HEAD}\ \boxed{1}\right]\right]\right]\right]\right] \end{array}\right]$$

We might augment this principle to reflect the fact that the head daughters of all headed phrases must bear the grammatical function *head*. That is:

$$\Delta(\text{headed phrase}) \sqsupseteq \left[\begin{array}{ll} \text{SYN} & \left[\text{LOC}\ \left[\text{HEAD}\ \boxed{1}\right]\right] \\ \text{DTRS} & \left[\text{LOC}\ \left[\text{HEAD-DTR}\ \left[\text{SYN}\ \left[\text{LOC}\ \left[\begin{array}{ll}\text{HEAD} & \boxed{1} \\ \text{GRAM-FN} & \textit{head}\end{array}\right]\right]\right]\right]\right] \end{array}\right]$$

4.3 Two Categorial-Like Equivalents

To illustrate the relationship between CPSG and Categorial Grammar, in this section we present the rules of CPSG in a categorial-like notation. Firstly we give an almost exact translation. Then we give a translation which adheres to the usual categorial principle of binary branching. What these translations show is that CPSG is not necessarily so far removed from Categorial Grammar as it may at first seem. The second translation also hints at how the rules might be reformulated to include a binary branching constraint and yield a more incremental framework.

4.3.1 Version 1: An (Almost) Exact Translation

As in most Categorial Grammars we begin by inductively defining the set of all categories, CAT, in terms of a set of atomic categories, ATCAT. The elements of ATCAT correspond to those categories which are saturated. We also make use of an intermediate set of categories, ATCAT′, whose elements are the saturated and almost saturated categories. In CPSG terms, a category is almost saturated iff it it is of the form Z_s.

Define ATCAT′ as the smallest set such that

$$x \in \text{ATCAT} \Rightarrow x \in \text{ATCAT}'$$

and

$$x, z \in \text{ATCAT} \Rightarrow z\backslash x \in \text{ATCAT}'$$

Define CAT as the smallest set such that

$$x \in \text{ATCAT}' \Rightarrow x \in \text{CAT}$$

and

$$x \in \text{ATCAT}' \text{ and } y \subset \text{ATCAT}' \Rightarrow x/y \in \text{CAT}$$

Note that y here is a set, the set of subcategorised for complements. y may be empty.

There are several things to note about this definition of CAT:

- Specifiers must be of category z where z is an element of ATCAT. Consequently specifiers must be saturated.
- Each complement must be of category y where y is an element of ATCAT'. Hence complements must be saturated (as in the case of NPs) or almost saturated (as in the case of VPs).
- The definition effectively means that '\' binds more tightly than '/': $Z\backslash X/Y$ is not ambiguous and must be read as $(Z\backslash X)/Y$ — $Z\backslash(X/Y)$ is not a legal category.

It is the above restrictions on specifiers and complements which make this translation not exact.

To complete the categorial definition we give two natural deduction style rules, admitting head/specifier and head/complement phrases:

Head/Specifier Rule

$$\frac{\text{Z} \quad \text{Z}\backslash\text{X}}{\text{X}}$$

Head/Complement Rule

$$\frac{\text{X}/\{\text{Y}_0,\ldots,\text{Y}_n\} \quad \text{Y}_0 \ldots \text{Y}_n}{\text{X}}$$

To express word order constraints we must add features to the syntactic categories and constrain the application of the head/complement rule. The constraint will be of the form: the sequence $\langle\ Y_0 \ldots Y_n\ \rangle$ is a legitimate ordering of the set $\{Y_0,\ldots,Y_n\}$ with respect to the linear precedence rules of the grammar.

As an illustration of this system, consider the following lexical assignments, where CAT is the set $\left\{\overline{\overline{V}}, \overline{\overline{N}}, \overline{\overline{P}}, \overline{\overline{D}}\right\}$.

$$\begin{array}{rcl} \text{Tigger} & \mapsto & \overline{\overline{N}} \\ \text{Fido} & \mapsto & \overline{\overline{N}} \\ \text{the} & \mapsto & \overline{\overline{D}} \\ \text{kitten} & \mapsto & \overline{\overline{D}}\backslash\overline{\overline{N}} \end{array} \qquad \begin{array}{rcl} \text{about} & \mapsto & \overline{\overline{P}}[\text{about}]/\{\overline{\overline{N}}\} \\ \text{with} & \mapsto & \overline{\overline{P}}[\text{with}]/\{\overline{\overline{N}}\} \\ \text{argues} & \mapsto & \overline{\overline{N}}\backslash\overline{\overline{V}}/\{\overline{\overline{P}}[\text{about}],\overline{\overline{P}}[\text{with}]\} \end{array}$$

We then have the following derivation:

$$\dfrac{\dfrac{\text{Tigger}}{\overline{\overline{N}}} \quad \dfrac{\dfrac{\text{argues}}{\overline{\overline{N}}\backslash\overline{\overline{V}}/\{\overline{\overline{P}}[\text{about}],\overline{\overline{P}}[\text{with}]\}} \quad \dfrac{\dfrac{\dfrac{\text{with}}{\overline{\overline{P}}[\text{with}]/\{\overline{\overline{N}}\}} \quad \dfrac{\text{Fido}}{\overline{\overline{N}}}}{\overline{\overline{P}}[\text{with}]} \quad \dfrac{\dfrac{\dfrac{\text{about}}{\overline{\overline{P}}[\text{about}]/\{\overline{\overline{N}}\}} \quad \dfrac{\dfrac{\text{the}}{\overline{\overline{D}}} \quad \dfrac{\text{kitten}}{\overline{\overline{D}}\backslash\overline{\overline{N}}}}{\overline{\overline{N}}}}{\overline{\overline{P}}[\text{about}]}}{\overline{\overline{N}}\backslash\overline{\overline{V}}}}{\overline{\overline{V}}}$$

'Tigger argues about the kitten with Fido' is similarly licensed by the grammar.

4.3.2 Version 2: A Translation with Binary Branching

Again we define the set CAT of syntactic categories in terms of a set of atomic categories ATCAT. This time, however, we do not maintain the distinction inherent in the previous formulation between categories of the form $z\backslash x/\{\ \}$ and $z\backslash x$. This corresponds to the use of strictly binary rules: in the previous formulation, the head/complement rule has unary, binary, ternary, etc. instantiations.

Define ATCAT′ as the smallest set such that

$$x \in \text{ATCAT} \Rightarrow x \in \text{ATCAT}'$$

and

$$x, z \in \text{ATCAT} \Rightarrow (z/\{\ \})\backslash x \in \text{ATCAT}'$$

Define CAT as the smallest set such that

$$x \in \text{ATCAT}' \text{ and } y \subset \text{CAT} \Rightarrow x/y \in \text{CAT}$$

Note that in this system ATCAT′ is not a subset of CAT: all legitimate syntactic categories are of the form x/y where y is a (possibly empty) set of syntactic categories.

Again we have two natural deduction style rules:

Head/Specifier Rule

$$\frac{\text{Z} \quad \text{Z}\backslash\text{X}/\text{Y}}{\text{X}/\text{Y}}$$

Head/Complement Rule

$$\frac{\text{X}/\{\text{Y}_0,\ldots,\text{Y}_n\} \qquad \text{Y}_i}{\text{X}/\{\text{Y}_0,\ldots,\text{Y}_{i-1},\text{Y}_{i+1},\ldots,\text{Y}_n\}}$$

One might restrict application of the head/specifier rule to heads of category $z\backslash x/\{\ \}$, enforcing the constraint of the previous formulation that specifiers are the last arguments to combine with heads. Without this constraint, although the grammar is spuriously ambiguous it does allow more incrementality. Again features must be added to the syntactic categories to allow the necessary word order constraints to be stated. These constraints again restrict the application of the head/complement rule.

As an illustration of this system, consider the following lexicon and assignment of categories, where CAT is the set $\left\{\overline{\overline{\text{V}}}, \overline{\overline{\text{N}}}, \overline{\overline{\text{P}}}, \overline{\overline{\text{D}}}\right\}$.

Tigger	$\mapsto$	$\overline{\overline{\text{N}}}/\{\ \}$	about	$\mapsto$	$\overline{\overline{\text{P}}}[\text{about}]/\{\overline{\overline{\text{N}}}/\{\ \}\}$
Fido	$\mapsto$	$\overline{\overline{\text{N}}}/\{\ \}$	with	$\mapsto$	$\overline{\overline{\text{P}}}[\text{with}]/\{\overline{\overline{\text{N}}}/\{\ \}\}$
the	$\mapsto$	$\overline{\overline{\text{D}}}/\{\ \}$	argues	$\mapsto$	$(\overline{\overline{\text{N}}}/\{\ \})\backslash\overline{\overline{\text{V}}}/\{\overline{\overline{\text{P}}}[\text{about}]/\{\ \},\overline{\overline{\text{P}}}[\text{with}]/\{\ \}\}$
kitten	$\mapsto$	$(\overline{\overline{\text{D}}}/\{\ \})\backslash\overline{\overline{\text{N}}}/\{\ \}$			

We then have the following derivation:

Tigger	argues	with	Fido	about	the	kitten
$\overline{\overline{\text{N}}}/\{\ \}$	$(\overline{\overline{\text{N}}}/\{\ \})\backslash\overline{\overline{\text{V}}}/\{\overline{\overline{\text{P}}}[\text{about}]/\{\ \},\overline{\overline{\text{P}}}[\text{with}]/\{\ \}\}$	$\overline{\overline{\text{P}}}[\text{with}]/\{\overline{\overline{\text{N}}}/\{\ \}\}$	$\overline{\overline{\text{N}}}/\{\ \}$	$\overline{\overline{\text{P}}}[\text{about}]/\{\overline{\overline{\text{N}}}/\{\ \}\}$	$\overline{\overline{\text{D}}}/\{\ \}$	$(\overline{\overline{\text{D}}}/\{\ \})\backslash\overline{\overline{\text{N}}}/\{\ \}$
	$\overline{\overline{\text{V}}}/\{\overline{\overline{\text{P}}}[\text{about}]/\{\ \},\overline{\overline{\text{P}}}[\text{with}]/\{\ \}\}$	$\overline{\overline{\text{P}}}[\text{with}]/\{\ \}$			$\overline{\overline{\text{N}}}/\{\ \}$	
	$\overline{\overline{\text{V}}}/\{\overline{\overline{\text{P}}}[\text{about}]/\{\ \}\}$			$\overline{\overline{\text{P}}}[\text{about}]/\{\ \}$		
		$\overline{\overline{\text{V}}}/\{\ \}$				

Note that in not restricting the head/specifier rule, this is only one of three derivations possible for this string. Again 'Tigger argues about the kitten with Fido' is similarly licensed by the grammar.

4.4 Specifiers and Subjects

In the preceding sections, we have treated determiners as specifiers to nouns and noun phrases as specifiers to verbs. This follows much of the transformational literature, but [Borsley ∞] has argued against the identification of subjects and specifiers made by this treatment. In arguing for this position, Borsley cites examples such as those in (3), where in the first he claims that 'with' acts as a specifier to the sentence 'Fido too tired to bark', which in turn consists of a verb phrase having 'Fido' as its subject, and in the second be claims that 'so' acts as a specifier to the predicative adjective 'boisterous', whose subject is filled (via the control of the copula) by 'Tigger'.

(3) a. With Fido too tired to bark ...

b. Tigger was so boisterous that ...

Each case seems to require a distinction to be drawn between two non-complement elements which combine with the head (a verb in the first instance and an adjective in the second). Borsley achieves this distinction by differentiating between specifiers and subjects.

The distinction between specifiers and subjects is also supported by semantic facts — specifiers, unlike subjects, are semantically not arguments, and so the contribution of a subject to the semantic content of a constituent is very different from the contribution of a specifier — and facts concerning extraction — specifiers, unlike arguments (including embedded subjects) and modifiers cannot be extracted.

Whilst this distinction might be easily incorporated into CPSG by including a further head attribute, SUBJECT, along with a corresponding grammar rule and some minor tinkering with other grammar rules to account for the optionality of specifiers, we do not investigate the possibility here.

5 Inverted Phrases

[Pollard & Sag 87] posit a grammar rule, Grammar Rule 3, that admits inverted clauses, such as 'did Tigger miaow'. Although this rule could be translated directly into CPSG, we are reluctant to do so for two reasons. Firstly, its domain of applicability is relatively small, applying only to phrases headed by inverted auxiliary verbs. It is difficult to motivate the existence of such a rule when all other phrase structure rules are highly schematic and apply to heads of several syntactic categories. Secondly, because of our division of arguments into specifiers and complement, the required rule would need to amalgamate the arguments. That is, the required rule would not make use of the distinction between specifiers and complements which we have made in other contexts. If such a rule were necessary, it would weaken our arguments for making the distinction in the first place.

Rather than incorporating a special purpose rule for inverted phrases, CPSG treats "subject-auxiliary inversion" in terms of a lexical rule which relates lexical entries for auxiliaries of the form given in [Pollard & Sag 87] to "inverted auxiliaries" (see [Bach 83] for a similar proposal within categorial grammar). Inverted auxiliaries are similar to auxiliaries except that instead of subcategorising for a noun phrase specifier and a verb phrase complement, they subcategorise for two complements, a noun phrase and a verb phrase. More precisely, the feature structure description of the type auxiliary is:

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{Verb} \\ \text{FORM} & \textit{fin} \\ \text{INV} & - \\ \text{AUX} & + \end{bmatrix} \\ \text{GRAM-FN} & \textit{head} \\ \text{SPEC} & \boxed{1} \\ \text{COMP} & \{\boxed{2}\} \end{bmatrix} \\ \text{DTRS} & \textit{null} \end{bmatrix} \end{bmatrix}$$

Inverted auxiliaries are described by:

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{Verb} \\ \text{FORM} & \textit{fin} \\ \text{INV} & + \\ \text{AUX} & + \end{bmatrix} \\ \text{GRAM-FN} & \textit{head} \\ \text{SPEC} & \textit{null} \\ \text{COMP} & \{\boxed{1},\boxed{2}\} \end{bmatrix} \end{bmatrix} \\ \text{DTRS} & \textit{null} \end{bmatrix}$$

According to this analysis, inverted auxiliaries are unusual in that they subcategorise for a complement without subcategorising for a specifier. That is, they are of category Z_{c^*}. Inverted auxiliaries thus combine with their complements via the head/complement rule, yielding the phrase structure of Figure 3.7 for inverted phrases. The auxiliary is the head of such phrases, and like all lexical heads, precedes its complements. This structure is identical to that yielded by GPSG's metarule approach ([Gazdar *et al.* 85]), and that yielded by HPSG's analysis of "subject-auxiliary inversion" via Grammar Rule 3.

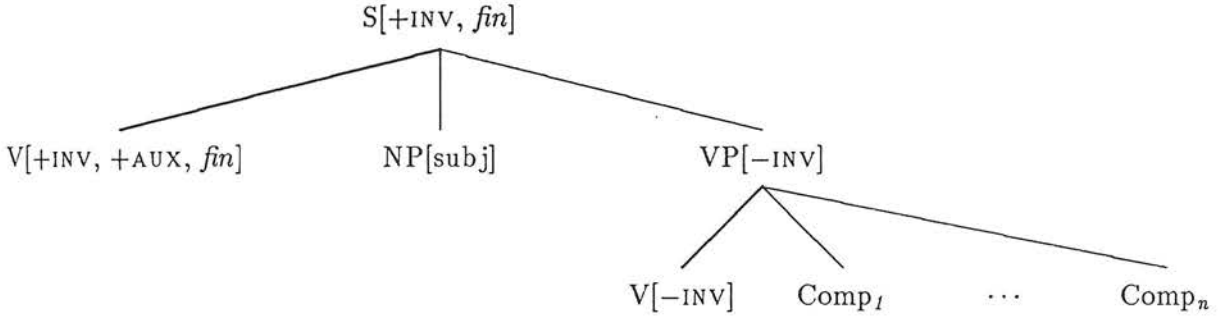


Figure 3.7: The Phrase Structure of Inverted Clauses

The control relations between the subcategorised complements are identical to those between the subcategorised complements of a transitive raising verb: the noun phrase is identified with the subject of the verb phrase. Inverted auxiliaries might therefore be viewed as "subjectless" transitive raising verbs. Note also that the importance of the attribute INV is diminished: it is no longer needed to distinguish between heads of Grammar Rule 2 and Grammar Rule 3. The attribute is still required, however, to ensure that embedded sentences are not inverted.

As noted in [Pollard & Sag 87, p. 64], some auxiliaries (such as first person singular 'aren't') are necessarily inverted, whilst others (such as 'better') cannot be inverted. These examples fall outside of the scope of the lexical rule that relates auxiliaries and inverted auxiliaries: the partial function $invert_t$ involved in the lexical rule is undefined on 'better' and its inverse is undefined on 'aren't'.

A similar lexical rule may be employed to treat bare plural noun phrases, mapping lexical constituents of category N_{s,c^*} to constituents with plural agreement features of category N_{c^*}.

6 Head/Adjunct Phrases

Given that adjuncts are constituents which combine with a constituent of category α to form another constituent of category α, a lexicalist theory has a choice in providing a treatment. Either the adjunct might be marked with the categories of the heads it may modify, or the head may be marked with the categories of the adjuncts by which it may be modified. We may represent these options in terms of schematic immediate dominance rules, the first of which takes the form:

$$\alpha \rightarrow \beta_{\alpha} \ , \ \alpha$$

and the second of which takes the form:

$$\alpha \rightarrow \beta \ , \ \alpha_{\beta}$$

where α is the head and β is the adjunct.

Categorial Grammars generally pursue the first of these options (cf. [Lambek 58], [van Benthem 86], [Uszkoreit 86], [Steedman 87], [Zeevat *et al.* 87]), though [Steedman 85] is a notable exception. HPSG pursues the second.

Within our framework, and within HPSG, both of which are consistent with the notions of $\overline{X}$-syntax, there is the additional question of which type of categories adjuncts can modify. Adjectival phrases, for example, are normally treated within $\overline{X}$-syntax as adjuncts of $\overline{N}$ but not as adjuncts of N or $\overline{\overline{N}}$. This treatment stems from distributional and semantic considerations. Most other adjuncts seem to behave in an analogous manner, modifying only heads of category $\overline{X}$ (cf. [Andrews 83]). This presents a slight problem

for a lexical theory adhering to the option whereby adjuncts are selected for by heads, as it would seem that the selection must be marked in the lexicon (i.e., at bar level 0) but apply only at bar level 1. To overcome this we might treat this selection as a head feature (so that if X bears the feature, $\overline{X}$ and $\overline{\overline{X}}$ will also bear the feature), and ensure that the grammar rule that licenses head/adjunct phrases only applies to heads of category $\overline{X}$.

[Radford 88, pp. 255–257], however, argues that the possibility should be left open for adjuncts to modify heads of category X and $\overline{\overline{X}}$, as well as heads of category $\overline{X}$. As possible examples, Radford suggests 'enough', as in 'He isn't [[proud] enough] of his country', as a modifier of heads of category X, and 'even', as in 'He might [even [have got lost]]', as a modifier of heads of category $\overline{\overline{X}}$. Whilst it is not clear that either 'enough' or 'even' are modifiers (neither can be iterated), presentential adverbs may be taken as adjuncts to $\overline{\overline{V}}$. On the basis of this we assume that adjuncts do not universally modify signs of category $\overline{X}$, although the type of phrase which an adjunct can modify is specific to that adjunct (or class of adjuncts). That is, an adjunct can, for example, be an $\overline{X}$ adjunct without being an X adjunct or an $\overline{\overline{X}}$ adjunct. Similarly, an adjunct might be an X adjunct without being an $\overline{X}$ or $\overline{\overline{X}}$ adjunct. Given this, it would seem that in a lexical theory adjuncts cannot be selected for by heads, but must select the heads that they modify, as if it were the other way around, it is not clear how, in the case of one adjunct, phrases such as [$\overline{X}$ $\overline{\overline{Adj}}$] could be licensed but phrases such as [$\overline{\overline{X}}$ $\overline{\overline{Adj}}$] and [X $\overline{\overline{Adj}}$] be blocked, whilst in the case of another adjunct, phrases such as that [X $\overline{\overline{Adj}}$] could be licensed with the other phrases being blocked. We thus adopt the approach whereby adjuncts select for the categories which they adjoin to, though note that if we were not so concerned with the lexical approach, we could have three head/adjunct rules, one for each bar level, and include features on adjuncts and in the rules to indicate which rules apply for which adjuncts.

A further advantage of this approach is that it eliminates the need for the kind of complex relational dependencies which [Pollard & Sag 87] require in their treatment of adjuncts within HPSG. In treating heads as selecting for adjuncts, Pollard and Sag employ a set valued head feature whose elements are partial descriptions of possible adjuncts. In a head/adjunct construction, the adjunct must be compatible with one

of these partial descriptions. Unifying the description of the adjunct with any element of the set, or replacing the set with the disjunction of its elements and unifying the description of the adjunct with the resultant feature structure, is unsuitable as such an approach would prevent iteration of adjuncts: the nodes on the projection of every head could dominate a total of at most one adjunct. However, although a head can be modified by a number of adjuncts, each adjunct can only modify a single head, so this problem does not arise when we treat adjuncts as selecting the heads which they modify: the selection may take the form of a feature-structure valued attribute (which is the disjunction of all possible partial descriptions of suitable heads) which is unified with the head that the particular instance of the adjunct modifies. There is no need for a set valued attribute and no need for checking that signs unify without actually unifying them.

Within $\overline{\text{X}}$-syntax is it also normally required that adjuncts themselves be maximal projections. We adopt this constraint within the grammar rules licensing head/adjunct phrases. Note though, as mentioned above, that the specification of which categories an adjunct phrase can modify, which is born by the lexical head of the adjunct phrase, must be a head attribute so that it will be inherited by the phrasal projection of the head of the adjunct phrase (i.e., by the adjunct phrase as a whole).

It remains to specify precisely how adjuncts are marked for the categories which they may modify and precisely how head/adjunct phrases relate to their sub-constituents. For the first of these we tentatively postulate a further head attribute, ADJOINS-TO. We take this to be a feature structure valued attribute, where the value for any constituent C is the disjunction of all possible constituents which the phrase headed by C may adjoin to. In the case of, for example, a noun, which cannot adjoin to anything, this attribute will take the value *null*. Similarly in the case of verbs. For adjectives on the other hand, the value will be the partial description of an $\overline{\text{N}}$, and for prepositions, the value will be the disjunction of the partial description of $\overline{\text{N}}$ and the partial description of $\overline{\text{V}}$. To be more precise, prepositions will bear the value:

$$\left[\begin{array}{l}\text{SYN}\ \left[\begin{array}{l}\text{LOC}\ \left[\begin{array}{ll}\text{HEAD} & \left[\begin{array}{ll}\text{MAJ} & \textit{Noun} \vee \textit{Verb} \\ \text{ADJOINS-TO} & \textit{null}\end{array}\right] \\ \text{SPEC} & [\ ] \\ \text{COMPS} & \textit{null}\end{array}\right]\end{array}\right]\end{array}\right]$$

for their SYN|LOC|HEAD|ADJOINS-TO attribute. Note that in the above the SPEC is specified as having value [ ] to ensure that the phrase modified is indeed of category $\overline{\text{X}}$ and not of category $\overline{\overline{\text{X}}}$.

The problem of word order remains. In English some adjuncts precede their heads, others follow them, and still others may occur in either position. We consider two possible solutions here, though the first is clearly problematic. In each, word order is governed by the grammatical functions *premodifier* and *postmodifier* which a modifier can bear in any phrase.

The first possibility involves marking modifiers with their grammatical relations lexically. Thus, we might mark non-argument prepositions as [SYN|LOC|GRAM-FN *postmodifier*]. Adjectives, on the other hand, might be marked as [SYN|LOC|GRAM-FN *premodifier*]. Adverbs, which are not fussy about their relation with their head, may be unspecified for their SYN|LOC|GRAM-FN attribute.

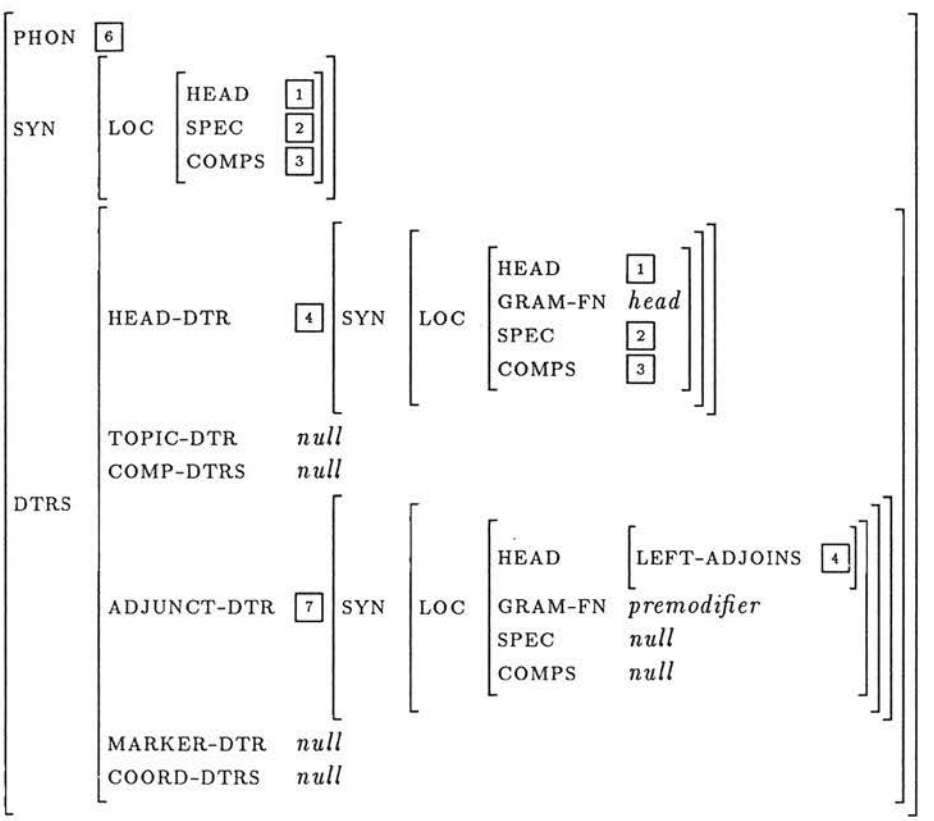
Given such a scheme (and a suitable grammar rule), word order within head/adjunct phrases might be determined via the usual relation *order-phonology*, employing the following linear precedence rules:

$$\text{premodifier} \prec \text{head} \prec \text{postmodifier}$$

This option requires that if a modifier can *premodify* phrases of α, then if it can also modify phrases of category β it *must* also premodify them (and similarly for postmodification). The real problem with this approach though, is that it is not clear how a lexical marking of heads of modifier phrases could be percolated up to the modifier phrases: a preposition is not a postmodifier in a prepositional phrase, it is a head, it is the prepositional phrase as a whole which is a postmodifier in an [$\overline{\text{N}}$ PP] complex, and so clearly this cannot be solved by making GRAMMATICAL-FUNCTION a head attribute.

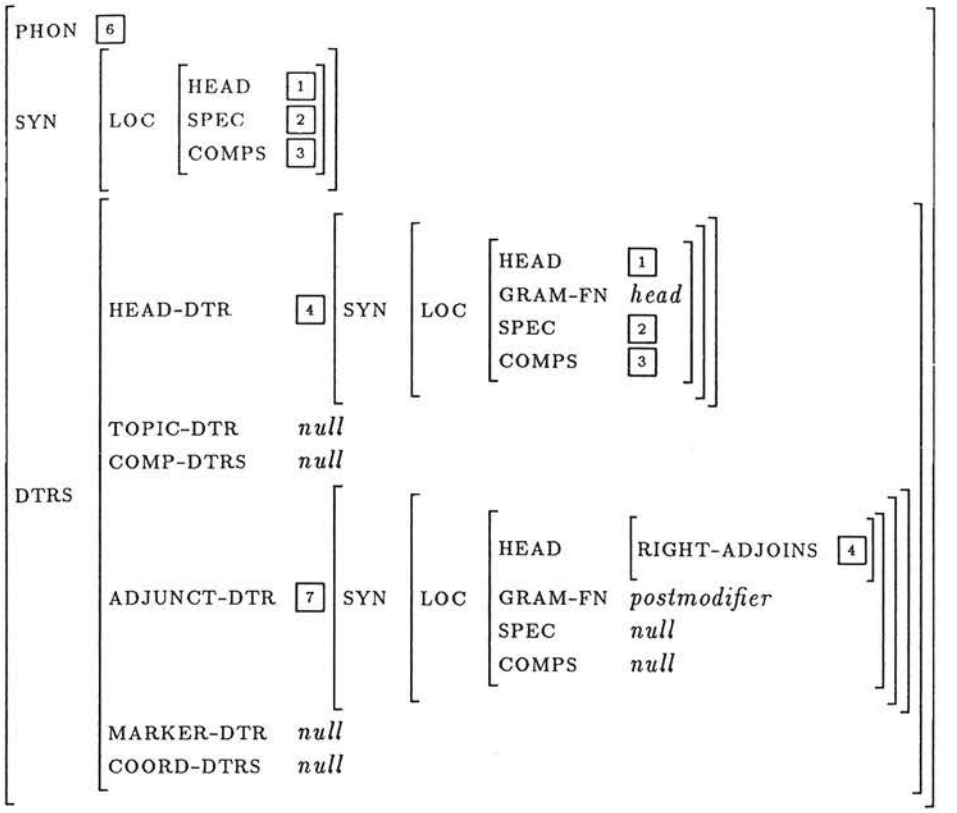
The other possibility, which *is* feasible, involves splitting the ADJOINS-TO attribute into two attributes, LEFT-ADJOINS and RIGHT-ADJOINS and using this to govern the word order. In this way, a modifier may specify the relative order in which the modifier phrase which it heads stands in relation to the phrase which the modifier phrase modifies. Thus we replace our original ADJOINS-TO attribute with two attributes, LEFT-ADJOINS and RIGHT-ADJOINS, and consider two subtypes of head/adjunct phrase: head/pre-adjunct phrase and head/post-adjunct phrase.

Head/pre-adjunct phrases are described by:



and *order-phonology*(6,{4,7})

Head/post-adjunct phrases are described by:



and *order-phonology*(6,{4,7})

Order-phonology must implement the above linear precedence rule, i.e.:

$$\text{premodifier} \prec \text{head} \prec \text{postmodifier}$$

We illustrate this treatment of adjuncts with two examples:

The Preposition *in*

$$\begin{bmatrix}
\text{PHON} & \langle in \rangle \\
\text{SYN} & \text{LOC} \begin{bmatrix}
\text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{Preposition} \\ \text{FORM} & \textit{in} \\ \text{LEFT-ADJOINS} & \textit{null} \\ \text{RIGHT-ADJOINS} & \overline{\text{X}} \end{bmatrix} \\
\text{GRAM-FN} & \textit{head} \\
\text{SPEC} & \textit{null} \\
\text{COMPS} & \left\{ \begin{bmatrix} \text{SYN} & \text{LOC} \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{Noun} \\ \text{CASE} & \textit{acc} \\ \text{FORM} & \textit{norm} \end{bmatrix} \\ \text{GRAM-FN} & \textit{obj} \\ \text{SPEC} & \textit{null} \\ \text{COMPS} & \textit{null} \end{bmatrix} \end{bmatrix} \right\}
\end{bmatrix} \\
\text{DTRS} & \textit{null}
\end{bmatrix}$$

where $\overline{\text{X}}$ is

$$\begin{bmatrix}
\text{SYN} & \text{LOC} \begin{bmatrix}
\text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{Noun} \vee \textit{Verb} \\ \text{FORM} & \textit{norm} \end{bmatrix} \\
\text{SPEC} & [\ ] \\
\text{COMPS} & \textit{null}
\end{bmatrix}
\end{bmatrix}$$

The head/complement grammar rule licenses phrases of the form 'in the garden', which may then post-modify constituents of category $\overline{\text{N}}$ or $\overline{\text{V}}$ via the head/post-adjunct rule: see Figure 3.8.

The Adverb *quickly*

$$\begin{bmatrix}
\text{PHON} & \langle quickly \rangle \\
\text{SYN|LOC} & \begin{bmatrix}
\text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{Adverb} \\ \text{LEFT-ADJOINS} & \boxed{1} \\ \text{RIGHT-ADJOINS} & \boxed{1} \end{bmatrix} \\
\text{GRAM-FN} & \textit{head} \\
\text{SPEC} & \textit{null} \\
\text{COMPS} & \{\ \}
\end{bmatrix}
\end{bmatrix}$$

where $\boxed{1}$ is

$$\begin{bmatrix}
\text{SYN} & \text{LOC} \begin{bmatrix}
\text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{Verb} \end{bmatrix} \\
\text{COMPS} & \textit{null}
\end{bmatrix}
\end{bmatrix}$$

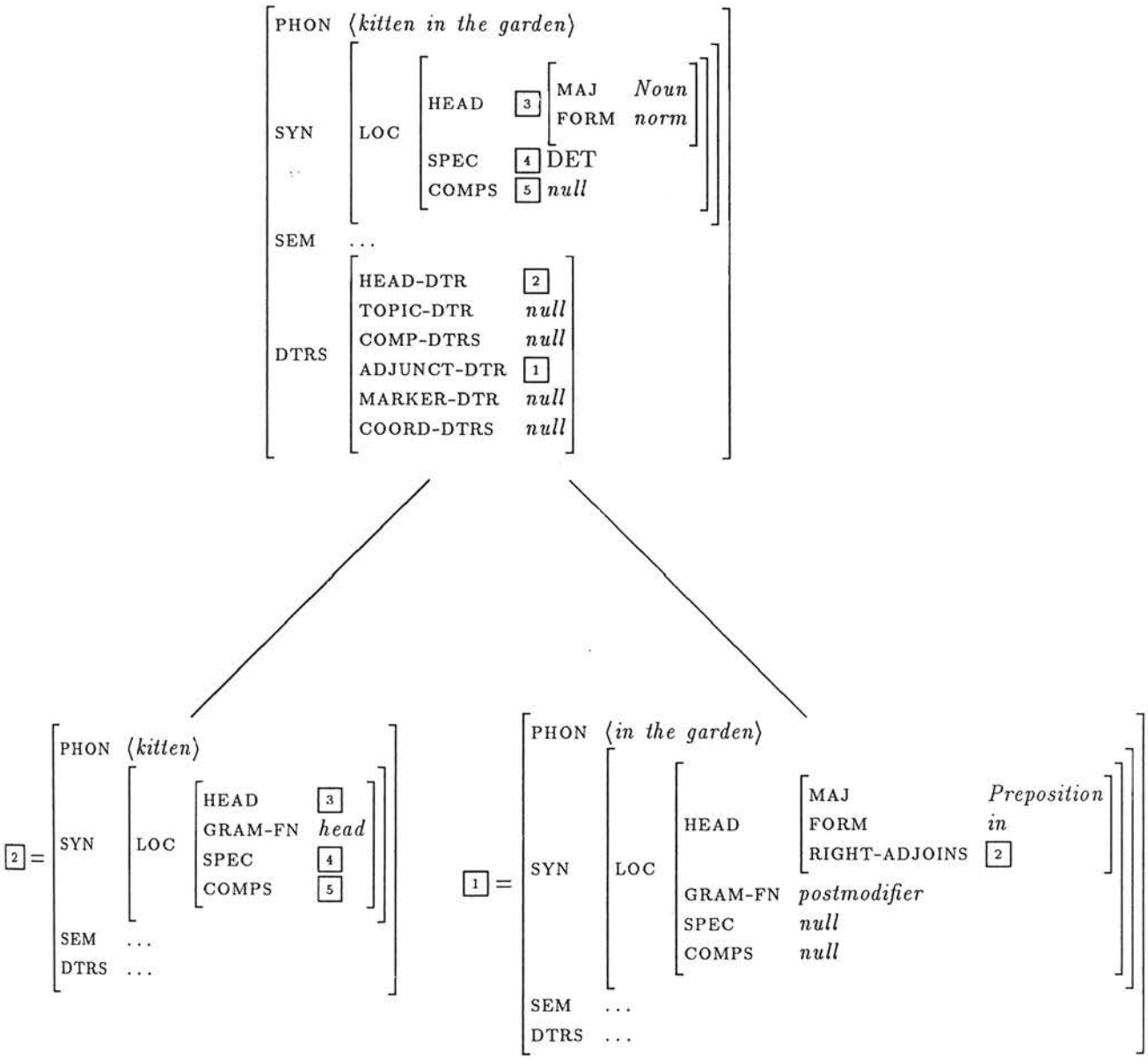


Figure 3.8: An Example Head/Post-Adjunct Phrase

The head/complement rule allows adverbial phrases to be formed (by combining the adverb with all of its required (i.e., no) complements). Either head/adjunct rule will then license the modification of non-lexical verbal categories. That is, it may left-adjoin to a VP or an S, or it may right-adjoin to a VP or an S. Note that this ability to modify either VPs or Ss arises from the fact that the SPEC attribute is not defined in [1]. This in turn relates back to Borsley's argument about treating VPs and Ss as forming a natural class.

7 Summary

In this chapter we have introduced the basic syntactic notions of CPSG via a consideration of three types of phrases, head/specifier phrases, head/complement phrases, and head/adjunct phrases. In so doing, a number of major divergences from HPSG have been noted. Primarily these include the use of separate SPECIFIER and COMPLEMENT attributes to indicate subcategorisation requirements, the use of a set valued attribute to represent subcategorised for complements, and the associated head feature GRAMMATICAL-FUNCTION, which indicates the grammatical function of a phrase within its immediate super-ordinate phrase, and the treatment of adjuncts as selecting for heads, rather than being selected for by heads.

Chapter 4

Unbounded Dependencies

Unbounded dependencies occur in a variety of linguistic contexts. We begin this chapter by examining these contexts and emphasising the similarities between [NP VP] constructs and [XP S[SLASH XP]] constructs. In all our discussion we presuppose a GPSG-style treatment of unbounded dependencies as involving three mechanisms: slash termination, slash percolation, and slash binding. In section 2 we present some background to our account by reviewing the GPSG treatment and presenting the attributes employed by CPSG in its treatment. Several established approaches to slash termination are rephrased in CPSG in section 3. None of these approaches are found to be without fault, and rather than being stipulative, the question of exactly how slash termination should be accomplished is left open. Section 4 examines the percolation of binding attributes, and gives a reconstruction of GPSG's Foot Feature Principle within CPSG. This reconstruction does not make recourse to defaults as required in GPSG. Section 5 sees a discussion of slash binding and a presentation of the rule licensing head/filler phrases. This is followed in sections 6 and 7 by a discussion of relative clauses and *wh*-questions, and the features REL and QUE. Finally, section 8 presents two alternatives to our account of extraction inspired by the similarities between head/filler and head/specifier phrases.

1 The Data: Two Dimensions of Variation

Topicalisation is perhaps the simplest construction involving an unbounded dependency. This is illustrated by the examples in (1).

(1) a. [Tom]$_i$ Tigger chases $\__i$
Cf. Tigger chases Tom

b. [Tom]$_i$ Tigger believes Fido chases $\__i$
Cf. Tigger believes Fido chases Tom

c. [Tom]$_i$ Tigger believes Fido thinks Rover chases $\__i$
Cf. Tigger believes Fido thinks Rover chases Tom

d. [To Tom]$_i$ Tigger gives the bone $\__i$
Cf. Tigger gives the bone to Tom

e. [Tom]$_i$ Tigger gives the bone to $\__i$
Cf. Tigger gives the bone to Tom

In each case, a saturated constituent occurs in the phrase initial position, and that constituent is followed by a sentence "missing" the constituent. The phrase initial constituent is said to be extracted. The dependency between the extracted element and the head which subcategorises for it is not clause bounded. Example (1b) illustrates a dependency across one clause boundary and example (1c) illustrates a dependency across two clause boundaries.

In this section we look more closely at the unbounded dependency data, discussing in particular two independent dimensions along which the data may be carved. The first concerns the grammatical function of the extracted element, and the second concerns two features which that element might bear: ±REL and ±QUE.

1.1 Dimension I: Grammatical Function

Each of the above examples in (1) involves the extraction of a constituent which serves as a complement to some head. In (1a), (1b) and (1c), 'Tom' is a complement of 'chases'. In (1d), 'to Tom' is a complement of 'gives'. In (1e), 'Tom' is a complement of 'to'. Constituents which serve other grammatical functions may also be extracted. It

is often argued, however, that specifiers cannot be extracted. Arguments for or against specifier extraction tend to be theory internal, and consequently it may be argued that it is a question that cannot be resolved by stipulation within a formal theory. What can be said in a theory neutral way is that if specifier extraction is allowed, standard sentences are structurally ambiguous between a form involving extraction and a form not involving extraction. This is our principal argument for the CPSG stipulation that specifiers cannot be extracted. Note that this stipulation really depends on the division in CPSG of arguments into complements and specifiers. Such a stipulation in HPSG, where specifiers and complements are not distinguished, would be much more difficult to motivate.

Whilst specifiers cannot normally be extracted, embedded specifiers — subjects of sentences which are themselves arguments to higher functors — can be:

(2) a. [Tigger]$_i$ Rover expects $\__i$ will miaow

b. [Tigger]$_i$ it appears $\__i$ is hungry

c. [Tigger]$_i$ Piglet is afraid $\__i$ will frighten Eeyore

Note though the well known "* that trace" phenomenon:

(3) a. Rover expects that Tigger will miaow

b. *[Tigger]$_i$ Rover expects that $\__i$ will miaow

c. It appears that Tigger is hungry

d. *[Tigger]$_i$ it appears that $\__i$ is hungry

e. Piglet is afraid that Tigger will frighten Eeyore

f. *[Tigger]$_i$ Piglet is afraid that $\__i$ will frighten Eeyore

The principal verbs which license the extraction of their embedded subjects are raising verbs, but as the examples involving the adjective 'afraid' demonstrate, the phenomenon is not restricted to verbs. It seems that all elements which subcategorise for sentential complements license the extraction of their embedded subject. Given that we do not allow extraction from specifier position, these examples cannot be analysed in terms of the sentential complement containing a gap. In section 3.2, we discuss how a lexical

rule which parallels GPSG's Slash Termination Metarule 2 might be employed to license such gaps.

A final class of constituent which may be extracted is the class of adjuncts:

(4) a. [To the park] Tigger trotted

b. [Clumsily] Tigger crept up on Eeyore

What makes these cases slightly different from the extraction of complements and embedded subjects is that it is not possible to tell from the sentential constituent in isolation, the constituent which the adjunct is notionally extracted from, that it has had an adjunct extracted from it: the subconstituent 'Tigger trotted' is saturated and as such it is difficult to motivate an analyses of the constituent as containing a gap.

The simplest approach to such examples might be to postulate a grammar rule such as:

$$S \rightarrow XP \ S$$

where some constituent in S licenses the adjunct XP.

Such a rule is effectively employed by the LFG approach to unbounded dependencies ([Kaplan & Zaenen 87]), where it licenses all instances of extraction. However, a global notion of saturation is crucial to the use of this one rule for all instances of extraction. In LFG this is provided by the principles of *completeness* and *coherence*, which are well-formedness conditions on f-structures, but within an HPSG-style grammar there is no such global notion of saturation, and it is thus difficult to reconcile such a rule with a treatment of the extraction of complements and embedded subjects. Nevertheless, adjunct extraction is clearly very closely related (if not identical) to the extraction of constituents serving other grammatical functions (as evidenced by the independence of this dimension and the dimension discussed below), and so a single rule licensing the combination of extracted elements with their licensing constituents is definitely to be preferred.

In summary then, the topicalisation data suggests two principal sentential structures. The first corresponds to constituents of type head/specifier phrase:

$$S \rightarrow XP \ S[\text{SPEC } XP]$$

We introduce a new type **head/filler phrase**, for constituents of the second type. Such phrases take the form:

$$S \rightarrow XP \ S[\textsc{slash} \ \{XP\}]$$

As we argue throughout the following subsections, these two sentential structures have several characteristics in common, most notably each is head final, and involves a head combining with a single non-complement element. The commonalities that these structures possess motivates the CPSG type **head/topic phrase**. This type is taken to dominate **head/specifier phrase** and **head/filler phrase**, and corresponds to what those phrases have in common.

1.2 Dimension II: The REL and QUE Features

All examples in the previous section involve topicalisation: the extraction of some argument or adjunct to a presentential position. Some relative clauses and *wh*-questions also involve unbounded dependencies. Considering relative clauses first, if the filler in the above examples is replaced by an analogous constituent containing a relative pronoun (i.e., a constituent bearing the feature +REL), then the examples become grammatical relative clauses. This is illustrated for some of the above examples in (5):

(5) a. ...[whom]$_i$ Tigger believes Fido chases $\__i$

b. ...[to whom]$_i$ Tigger gives the bone $\__i$

c. ...[whom]$_i$ Rover expects $\__i$ will miaow

d. ...[who]$_i$ it appears $\__i$ is hungry

e. ...[who]$_i$ Piglet is afraid $\__i$ will frighten Eeyore

f. ...[to which] Tigger trotted

Again for embedded subject extraction the "* that trace" facts hold:

(6) a. *...[who]$_i$ it appears that $\__i$ is hungry

b. *...[who]$_i$ Piglet is afraid that $\__i$ will frighten Eeyore

The REL feature is independent of extraction: relative pronouns may occur in specifier position without there being any unbounded dependency or extracted element. This is

illustrated in (7).

(7) a. ...who will eat all of the chocolate

b. ...who is afraid of Eeyore

Such relative clauses follow the structure of "standard" (untopicalised) sentences. Indeed, if we take the topicalisation data to suggest two principal sentential structures as argued above, then the relative clause data suggests two analogous structures:

$$S[+\text{REL}] \rightarrow XP[+\text{REL}]\ \ S[\text{SPEC XP},-\text{REL}]$$

$$S[+\text{REL}] \rightarrow XP[+\text{REL}]\ \ S[\text{SLASH }\{XP\},-\text{REL}]$$

Putative relative clauses where the REL feature is marked on the second, sentential constituent, are not grammatical. That is, for a relative clause the REL feature must be marked on the clause initial XP. This is thus another generalisation that holds across the two rules, and which reinforces our treatment of the two rules as corresponding to subtypes of a single type.

One final comment on relative clauses concerns the analysis given to them by some categorial grammars (cf. [Steedman 85], [Morrill 88]). These involve treating the relative pronoun as essentially the head of a noun modifying clause. The relative pronoun is therefore taken to subcategorise for a sentence lacking an NP in some position (either pre-verbal or post-verbal), and a noun which the relative clause modifies. This treatment does not obviously generalise to examples where the relative pronoun does not head the clause initial constituent, such as examples (5b) and (5f) above. We therefore do not consider it as a viable analysis of nominal modification by relative clauses.

Wh-question sentences exhibit similar behaviour with respect to the QUE feature as relative clauses do with respect to the REL feature. A slight complication concerns the requirement that, for *wh*-questions which involve extraction, the clause from which the element is extracted must be inverted. This inversion is illustrated by the examples

in (8), where it has been necessary to insert auxiliaries to license the inversion:

(8) a. [whom]$_i$ does Tigger believe Fido chases $\__i$?

b. [to whom]$_i$ does Tigger give the bone $\__i$?

c. [whom]$_i$ does Rover expect $\__i$ will miaow?

d. [who]$_i$ does it appear $\__i$ is hungry?

e. [who]$_i$ is Piglet afraid $\__i$ will frighten Eeyore?

f. [to where] does Tigger trot?

That in (direct) *wh*-questions not involving extraction, as in (9), the auxiliary is not required implies that there are significant difference between such cases and the cases involving extraction, but there is also again the similarity that, for direct questions, the feature QUE must be marked on the sentence initial constituent, exactly paralleling the case of the REL feature, and further supporting the claim that both rule schema correspond to subtypes of a less instantiated type.

(9) a. What chases Tigger?

b. Who believes Tigger chases Fido?

For the most part we do not examine the syntax of *wh*-questions. The issues are blurred by various classes of question, such as direct questions and echo questions, and not of immediate concern to this thesis.

1.3 Other Examples

Unbounded dependencies occur in a variety of other contexts. These include:

"Tough" movement: Cases of "tough" movement, such as those in (10) have also been treated in terms of constituents containing gaps.

(10) a. Tigger is tough for Tom to talk to _

b. Tigger is easy to please _

In contrast to GPSG, which treats these constructions via a phrase structure rule, [Pollard 89b] and [Pollard & Sag ∞a] treat these constructions via the lexical

categories assigned to adjectives such as 'easy' and 'tough'. They take such constituents to subcategorise for complements containing gaps. This lexical treatment is a direct consequence of the lexical marking of subcategorisation requirements. On a different note, it is cases such as these, where heads impose conditions on the BINDING attributes of their complements, which require default specifications in GPSG to be overridden in the application of the Foot Feature Principle. We discuss this issue in detail in section 4.

Cleft sentences: It-clefts (11a) and pseudo-clefts (11b) may also involve constituents with extracted elements.

(11) a. It is Rover that Tigger chases _

b. What Tigger chases _ is Rover

Cleft-sentences also provide further support for the assimilation of S[SLASH {XP}] and S[SPEC XP] into a single category: the examples above are equally grammatical if the S[SLASH {XP}] subconstituents in each are replaced by constituents of category S[SPEC XP], as in (12):

(12) a. It is Rover that chases Tigger

b. What chases Tigger is Rover

Parasitic gap sentences: In parasitic gap sentences (cf. [Engdahl 83]), two gaps are filled by a single element, and one gap is licensed by the other. In (13a), both gaps are co-indexed with the relative pronoun, but the second gap is optional, as illustrated by (13b). In contrast to this, the first gap is not optional, as illustrated by (13c). The second gap is thus said to be parasitic on the first: it is licensed by the first.

(13) a. That is the kitten which Tigger chased _ before liking _

b. That is the kitten which Tigger chased _ before liking Rover

c. *That is the kitten which Tigger chased Rover before liking _

Purpose clauses: A final context in which extraction may occur involves purpose clauses, as in 14:

(14) a. There is a kitten for Tigger to miaow to _

b. Here are some chocolates to give _ to Tigger

1.4 Summary

In summary, this section has argued two principal points: that grammatical function and the features REL/QUE are two independent dimensions of variation relevant to unbounded dependencies, and that head/filler phrases and head/specifier phrases have several features in common and thus may be treated as subtypes of a less instantiated type, head/topic phrase.

The first of these points allows us to concentrate mainly on the mechanisms involved in the treatment of unbounded dependencies divorced from concerns of the REL and QUE features. These features are initially ignored in our discussion, and only reconsidered when all the mechanisms required by a CPSG treatment of unbounded dependencies have been developed.

The second of these points relates very strongly to the CPSG notion of a constituent hierarchy, motivating local hierarchical structure relating to the types head/specifier phrase and head/filler phrase.

2 The Background to an Account

2.1 The Elements of GPSG's Account

The account of extraction in CPSG draws heavily on that of GPSG, and for this reason we begin the account with a summary of the relevant aspects of the GPSG account.

Unbounded dependencies in GPSG are analysed as consisting of three parts: a top, a middle, and a bottom. The top part concerns the licensing of a local tree whose leaves include an extracted element and a constituent from which that element is extracted.

The middle concerns the licensing of local trees which propagate the dependency between the extracted element and its extraction site (or the head which subcategorises for it). The bottom concerns the licensing of local trees lacking some element: local trees whose heads are not as fully saturated as is normally required.

In GPSG, separate mechanism are involved in licensing each of the three parts. The top part, or *slash introduction*, is effected by special phrase structure rules. For topicalisation, for example, there is a special non-lexical immediate dominance rule which states that a sentence can be composed of an XP and an S/XP, where an S/XP is an S from which an XP has been extracted. The middle part, or *slash percolation*, is effected by two feature instantiation principles, the Head Feature Convention and the Foot Feature Principle. The bottom part, or *slash termination* is effected by the interaction of metarules and Feature Specification Defaults which govern the distribution of the NULL feature, which corresponds in GPSG to a phonologically empty constituent or trace.

The CPSG treatment of extraction also consists of these three parts. Slash introduction is licensed by a grammar rule for topicalised sentences, but can be lexically specified (as in "tough" adjectives, which subcategorise for complements containing gaps). Slash percolation is governed by a single principle, the Binding Inheritance Principle (following HPSG). As with all principles, this is really a well-formedness condition on feature structure descriptions of certain types of tokens. Whilst several options are available for slash termination, each is somehow flawed. In section 3 we discuss these options without favouring any above the others.

2.2 Binding Attributes in CPSG

Following HPSG, CPSG uses the attributes SLASH, REL, and QUE to treat unbounded dependencies. In [Pollard & Sag 87], these attributes are organised as:

$$\begin{bmatrix} \text{SYNTAX} & \begin{bmatrix} \text{BINDING} & \begin{bmatrix} \text{SLASH} & \{\ldots\} \\ \text{REL} & \{\ldots\} \\ \text{QUE} & \{\ldots\} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Each of the binding attributes take sets of feature structures for their values, where the feature structures describe extracted constituents, relative pronouns, and *wh*-question

words respectively.

This organisation is sufficient for most, but not all, instances of unbounded dependencies. The problem arises from the fact that not all gaps are equal. There exist constituents whose sub-constituents contain gaps which are not explicitly bound by an extracted element, but which nevertheless should not be marked as containing a gap. Missing object constructions are a case in point. Whilst 'to please _' contains an unfilled gap, 'easy to please _' does not, and it is not clear how the naïve presentation of binding attributes in [Pollard & Sag 87] can be augmented with a gap percolation principle which will prevent the gap from appearing to be unfilled in forming the phrase. Similar comments apply to the REL and QUE features: 'the kitten which hissed at Tigger' contains a relative pronoun, but does not bear the feature +REL, and 'Tigger wondered why Fido couldn't miaow' contains a *wh*-question word, but does not bear the feature +QUE.

This defect is remedied in [Pollard & Sag ∞a], where binding attributes are subdivided into INHERITED and TO-BIND: only those classified as INHERITED are actually inherited by a phrase, the others are bound. Thus, [Pollard & Sag ∞a] have:

$$\begin{bmatrix} \text{INHERITED} & \begin{bmatrix} \text{SLASH} & \{\ldots\} \\ \text{REL} & \{\ldots\} \\ \text{QUE} & \{\ldots\} \end{bmatrix} \\ \text{TO-BIND} & \begin{bmatrix} \text{SLASH} & \{\ldots\} \\ \text{REL} & \{\ldots\} \\ \text{QUE} & \{\ldots\} \end{bmatrix} \end{bmatrix}$$

CPSG adopts a similar approach, but divides first according to the type of binding attribute, and then according to the inheritance properties. We also employ different attribute names:

$$\begin{bmatrix} \text{SLASH} & \begin{bmatrix} \text{BOUND} & \{\ldots\} \\ \text{ALL} & \{\ldots\} \end{bmatrix} \\ \text{REL} & \begin{bmatrix} \text{BOUND} & \{\ldots\} \\ \text{ALL} & \{\ldots\} \end{bmatrix} \\ \text{QUE} & \begin{bmatrix} \text{BOUND} & \{\ldots\} \\ \text{ALL} & \{\ldots\} \end{bmatrix} \end{bmatrix}$$

Dividing first according to the type of binding attribute is primarily a matter of convenience, though it does encourage each binding attribute to be considered in isolation.

The binding attributes of SLASH are interpreted such that the value of ALL is the set of all gaps inherited from the sub-constituents of any constituent. The value of BOUND is the subset of those gaps in ALL which are actually bound, implicitly or explicitly, in the constituent and which therefore do not percolate. Thus we have the following sample values for the SYN|BIND|SLASH attribute:

easy $\begin{bmatrix} \text{BOUND} & \{\,\} \\ \text{ALL} & \{\,\} \end{bmatrix}$

to please $\begin{bmatrix} \text{BOUND} & \_ \\ \text{ALL} & \{\text{NP[acc]}\} \end{bmatrix}$

to talk to $\begin{bmatrix} \text{BOUND} & \_ \\ \text{ALL} & \{\text{NP[acc]}, \text{NP[acc]}\} \end{bmatrix}$

easy to please $\begin{bmatrix} \text{BOUND} & \{\,\} \\ \text{ALL} & \{\,\} \end{bmatrix}$

easy to talk to $\begin{bmatrix} \text{BOUND} & \_ \\ \text{ALL} & \{\text{NP[acc]}\} \end{bmatrix}$

In the above the symbol squote_ denotes an indeterminate value. The value of BOUND for any constituent is determined by the context in which the constituent occurs (subject to the constraint that it is a subset of the value of ALL), and can thus often not be determined for a constituent in isolation. If 'to please' occurs as a complement of 'easy', then the single gap in 'to please' will be bound as part of the subcategorisation requirement of 'easy', and hence in this instance BOUND will be instantiated to the singleton consisting of the one element in the ALL set. If, however, 'to please' occurs as a complement of 'try', as in 'Tigger I try to please', then the single gap in 'to please' will not be bound as part of the subcategorisation requirement of 'try', and hence in this instance BOUND will be instantiated to the empty set. This approach is very similar to gap threading (see, for example, [Pereira & Shieber 87]), however, the lexical specification of subcategorisation and binding requirements means that a head may explicitly bind gaps in its complements, as required by adjectives such as 'easy'.

The precise details of the instantiation of binding attributes is discussed in detail in section 4.

The attributes for REL and QUE are interpreted similarly. The value of ALL is the set of all +REL/+QUE constituents inherited from the sub-constituents of any constituent. The value of BOUND is the subset of those constituents in ALL which are actually bound, implicitly or explicitly, in the constituent and which therefore do not percolate.

A further question which we have neglected above concerns the need for set valued attributes for REL and QUE at all. When discussing the data the binary features ±REL and ±QUE where employed. Why can these binary features not be employed in developing the formal theory? The reasons are primarily semantic. Whilst in giving a syntactic treatment it is sufficient to only mark constituents as ±REL or ±QUE, when incorporating semantics it is necessary to know the semantic content of any element which makes a constituent ±REL or ±QUE. We thus use the set representation to store the whole description corresponding to the relevant subconstituent. Given the use of re-entrancy, this in computational terms is really only a pointer to the relevant subconstituent, which is also described as one of the daughters of the constituent. Note that, as [Pollard 89b] points out, it would be sufficient to indicate in these sets just the relevant substructures of the semantic content. We choose to use full descriptions for the sake of clarity.

The final point regarding this question is that a set valued attribute is also necessary in constituents involving several +QUE subconstituents, as in 'Who chased whom?'. The need for a set for the REL attribute is less clear, and our principal reason for adopting such a value is consistency with the other binding attributes.

The notation +QUE and +REL, which we use throughout this chapter, should thus be read as an abbreviation indicating a non-empty set value for a constituent's QUE|ALL attribute and REL|ALL attribute respectively. Similarly, −QUE and −REL indicate empty set values for a constituent's QUE|ALL and REL|ALL attributes.

3 Approaches to Slash Termination

"Slash termination" is the term employed within GPSG for the mechanism which licenses a local tree in which a sub-constituent is "missing". Within the framework of CPSG so

far presented, there are several options available for slash termination. We might choose to

- follow the GB tradition and include a "trace", a constituent whose phonology is the empty list, in the lexicon;
- follow [Sag 82] and augment the rule base to include specialised gap introduction rules (perhaps via metarules) which don't involve a trace; or
- follow [Popowich 88] and add lexical rules which map lexical items in their "standard" form to lexical items whose subcategorisation requirements are transferred from their SPEC and/or COMPS attributes to their SLASH attribute.

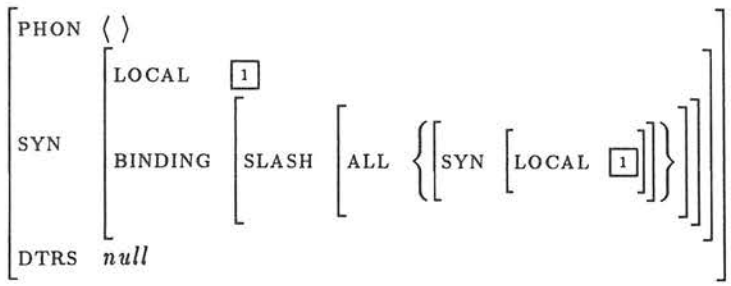
Note that GPSG as presented in [Gazdar *et al.* 85] essentially uses the first of these approaches combined with metarules to constrain the contexts in which traces may occur.

In this section we consider each of these possibilities in more depth, indicating precisely how each may be incorporated into CPSG. Each of the mechanisms does, however, have serious drawbacks, and these drawbacks prevent a conclusive argument from being advanced for any option over the others. We thus make no firm statement of how slash termination is effected in CPSG— this section really only presents some options.

3.1 Trace

Extraction in transformational terms in normally treated via a phonologically null constituent, or trace. The idea is that sentences involving extraction are derived from deep structures in a base form, and when an element is extracted or moved it leaves behind a phonologically null constituent. GPSG gives a non-transformational account of extraction which also involves the use of a trace. In the GPSG account, the trace fills the subcategorisation requirements of the relevant head and, by virtue of feature co-occurrence restrictions and universal feature instantiation principles, instantiates the head with a feature indicating that the phrase as a whole contains a gap. This account is modified for HPSG in [Pollard & Sag ∞a]. All that is required by this approach to slash termination is a lexical entry for trace, together with a principle to ensure the correct percolation of binding features.

Trace is only very partially instantiated, and can fill a subcategorisation requirement of any head. On filling such a subcategorisation requirement, trace also introduces an element into the SLASH set of the head (via its own SLASH set). Rephrasing this account in CPSG, we have as the lexical sign for trace:



This sign may fill a subcategorisation requirement of any head. When it does, its local features will become instantiated with those required of the argument (i.e., complement or specifier), thus instantiating the element in the SLASH set. This value may then be percolated to the constituent as a whole by some form of binding inheritance principle.

One problem with this, and any, account employing traces is that it licenses trace as a constituent in its own right. As such, trace may occur in place of any constituent. Whilst this means that there is no difficulty in licensing extraction of adjuncts, restrictions must be imposed on the occurrence of trace so as not to allow extraction from certain other sites. Firstly, to capture the constraint that extraction is not possible from specifier position we need to stipulate that trace cannot appear in specifier position, except in embedded sentential complements. In GPSG this constraint is effected by metarules which constrain the occurrence of the NULL feature which governs the distribution of trace. A more troublesome constraint concerns capturing the coordination data. Gaps may occur in coordinated phrases, as in (15a) and (15b), but gaps themselves may not be coordinated (cf. (15c), (15d), (15e)).

(15) a. This kitten [Tom likes _] but [Tigger chases _]

b. ...the kitten which [Tom takes pictures of _] and [Tigger detests _]

c. *This kitten Tom likes [[_] and [pictures of _]]

d. *This kitten Tom likes [[pictures of _] and [_]]

e. *This kitten Tom likes [[_] and [_]]

GPSG avoids this problem as metarules, including the metarule licensing the NULL feature, only apply to lexical Immediate Dominance rules, and the rule for coordinate structures is not lexical. However, in a theory in which gaps are not constituents in their own right, there is no need to resort to such lengths to prevent gap coordination: if a gap isn't a constituent a coordination rule which coordinates only constituents cannot license the examples in (15c), (15d) or (15e).

3.2 Lexical Rules

Slash termination (for complement gaps) might be accomplished without the use of a trace by a lexical rule which effectively moves elements from the COMPLEMENTS set of any lexical item to the SLASH set of that lexical item. For example, a lexical rule of "complement slash termination" might relate constituents of type **verb** to constituents of type **slashed verb** such that a lexical constituent described by:

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{Verb} \end{bmatrix} \\ \text{SPEC} & \boxed{1} \\ \text{COMPS} & \boxed{2} \end{bmatrix} \\ \text{BINDING} & \begin{bmatrix} \text{SLASH} & \begin{bmatrix} \text{ALL} & \{\,\} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{DTRS} & \textit{null} \end{bmatrix}$$

would be related to a constituent described by:

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{Verb} \end{bmatrix} \\ \text{SPEC} & \boxed{1} \\ \text{COMPS} & \boxed{3} \end{bmatrix} \\ \text{BINDING} & \begin{bmatrix} \text{SLASH} & \begin{bmatrix} \text{ALL} & \boxed{4} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{DTRS} & \textit{null} \end{bmatrix}$$

where $\boxed{3} \uplus \boxed{4} = \boxed{2}$

A further, independent lexical rule would be required to license the extraction of specifiers if so desired. It is not clear, however, how this kind of lexical rule approach could be generalised to allow the extraction of adjuncts.

The extraction of embedded subjects may also be licensed by a lexical rule. Such a lexical rule might map all lexical constituents which subcategorise for a finite sentential complement to similar constituents where the finite sentential complement is replaced by a finite verb phrase and an XP is added to the slash set corresponding to the subject of the initial sentential complement.

For transitive raising verbs, for example, we may posit a lexical rule which maps lexical constituents of type transitive raising (S[fin]) to constituents of type slashed transitive raising. Constituents of type transitive raising (S[fin]) may be described by:

$$\left[\begin{array}{ll} \text{SYN} & \left[\begin{array}{ll} \text{LOC} & \left[\begin{array}{ll} \text{HEAD} & \left[\text{MAJ} \quad \textit{Verb}\right] \\ \text{SPEC} & \text{NP} \\ \text{COMPS} & \{\text{S[fin]}\} \end{array}\right] \\ \text{BIND} & \left[\text{SLASH} \quad \left[\text{ALL} \quad \{\,\}\right]\right] \end{array}\right] \\ \text{DTRS} & \textit{null} \end{array}\right]$$

'Believes', as in 'Tom believes Tigger is miaowing', is an instance of this type. Note that the verbal complement is a finite sentence, and not a noun phrase and an infinitive verb phrase: 'believes' in this context is distinguished from 'believes' in 'Tom believes Tigger to be miaowing'.

Constituents of type slashed transitive raising may be described by:

$$\left[\begin{array}{ll} \text{SYN} & \left[\begin{array}{ll} \text{LOC} & \left[\begin{array}{ll} \text{HEAD} & \left[\text{MAJ} \quad \textit{Verb}\right] \\ \text{SPEC} & \text{NP} \\ \text{COMPS} & \{\text{VP[fin]}\} \end{array}\right] \\ \text{BIND} & \left[\text{SLASH} \quad \left[\text{ALL} \quad \{\text{NP}\}\right]\right] \end{array}\right] \\ \text{DTRS} & \textit{null} \end{array}\right]$$

That embedded subjects may be extracted from all sentential complements, whether the head is a transitive raising verb or not, suggests that this might be generalised to be independent of type. That is, such that for all constituents whose descriptions extend:

$$A = \begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{COMPS} & \{\text{S[fin]},\ldots\} \end{bmatrix} \\ \text{BIND} & \begin{bmatrix} \text{SLASH} & \begin{bmatrix} \text{ALL} & \{\,\} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{DTRS} & \textit{null} \end{bmatrix}$$

there is an analogous constituent having description:

$$B = \begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{COMPS} & \{\text{VP[fin]},\ldots\} \end{bmatrix} \\ \text{BIND} & \begin{bmatrix} \text{SLASH} & \begin{bmatrix} \text{ALL} & \{\text{NP}\} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{DTRS} & \textit{null} \end{bmatrix}$$

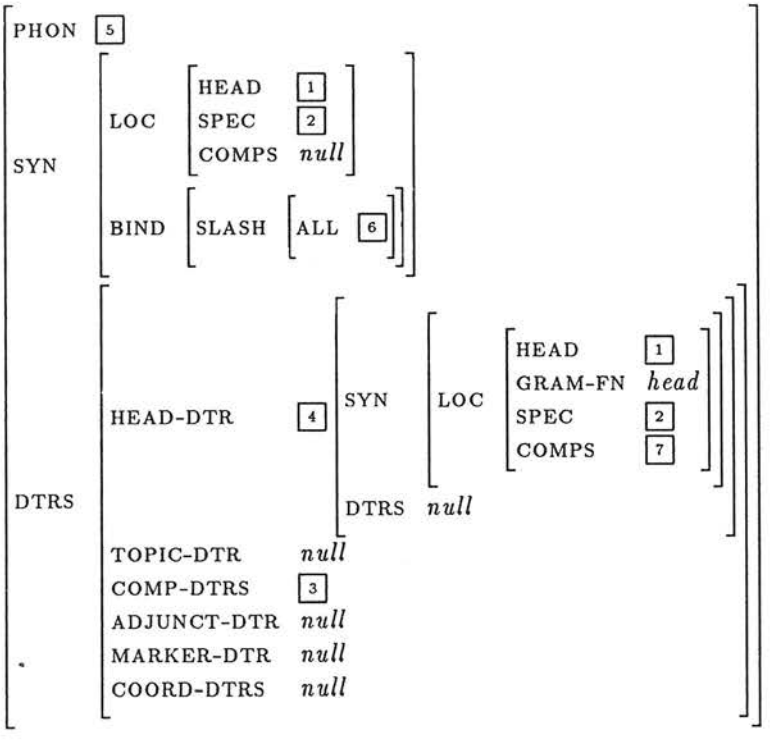
where all that is unspecified remains unchanged. This lexical rule, which may be formalised in terms of priority unification as "whenever there exists a lexical sign X which unifies with A, there exists a corresponding lexical sign X/B", where X/B is the priority unification of X with B, B taking priority, is the direct analogue of GPSG's Slash Termination Metarule 2. X/B might be read as "X in the context of B".

A potential problem in the lexical rule approach arises from the possible interaction of lexical rules. Given our use of a lexical rule for inversion, which relates a standard auxiliary verb to one requiring a sentential complement, the unconstrained application of a lexical rule for embedded subject extraction will license two structures for standard non-topicalised sentences involving auxiliaries. As well as the standard structure, there will be a structure where the subject has been extracted from the sentential complement of an inverted auxiliary. Thus if this approach is to be adopted we need some kind of constraint such as "the auxiliary inversion lexical rule applies after the slash termination lexical rule". Such a rule clearly has an undesirable *ad hoc* flavour to it.

3.3 Modified Phrasal Rules

An alternate approach to the licensing of gaps in place of complements involves a generalisation of the rule which licenses head complement phrases. The original rule presented in Chapter 3 licenses a constituent consisting of a head and all of its complements.

The generalisation is to allow constituents consisting of a head and some subset of its complements, provided that those complements which are not present as complement daughters are marked as gaps in the constituent. To capture this, the description of the type head/complement phrase might be generalised to:



where *order-phonology*([5],{[4]} ⊎ [3])

and *constituent*([4])

and *set-of-constituents*([3])

and [7] = [3] ⊎ [6]

Such an approach easily captures the constraint against specifier extraction: gaps cannot occur in specifier position because the above rule only refers to complements. Clearly, though, specifier extraction could just as easily be allowed by generalising the head/specifier rule in an analogous manner. The approach also shares with the lexical rule approach the feature that gaps are not constituents in their own right, and so again there is no question of constraining their occurrence in coordinate structures. It is not clear, however, how this approach might be extended to allow the extraction of embedded subjects or adjuncts. Both seem to present real difficulties for a treatment which is otherwise remarkably simple.

3.4 Summary and Discussion

Gap introduction rules have in common the lack of a need for a phonologically empty constituent or trace. This in itself has several benefits. Firstly, there is no direct evidence for the existence of such constituents, so positing their existence when other techniques avoid this seems unnecessary. Secondly if there is no phonologically empty constituent, then the question of where in the linear ordering of a constituent that constituent should appear does not arise, and consequently there is no specific "extraction site". [Pollard & Moshier 89, p. 7] argue that

> [O]ne of the most distinctive properties of a gap is that it is phonologically null; in terms of the phonological shape of a phrase containing it, the gap has no location. This point is difficult to appreciate with respect to a relatively fixed-constituent-order language like English where we sense that we can tell where the gap must be by finding the spot in the sentence where something appears to be missing; but in free-constituent-order languages (like Japanese), even this appearance is no longer present; there simply is no such thing as the location of the gap in the phonological structure of the phrase that contains it.

This notion of gap, which, unlike that of Pollard and Moshier, does not involve a notion of trace, avoids the issue of locating a trace within the phonological structure altogether. Thirdly, without such a constituent there is no question of a relationship being established between a filler and its extraction site. This is consistent with the psycholinguistic arguments of [Pickering & Barry 89], which suggest that in sentence processing some relation is established between fillers and the heads which subcategorise for them, rather than between fillers and extraction sites. Lastly, without such constituents, and all other things being equal, the complexity of parsing is reduced, as there is no question of locating empty constituents within a string.

Despite the above advantages of avoiding the use of a trace, neither of the methods suggested above for doing this are universally applicable. In particular, neither method allows for extraction of adjuncts. We are therefore reluctant to select any one over the other for the CPSG treatment of extraction. One alternative which could avoid the use of phonologically empty constituents which is not examined above is that employed by LFG. The LFG principles of completeness and coherence allow an elegant and simple treatment of extraction ([Kaplan & Zaenen 87]). However it is not clear how these principles can be incorporated into CPSG.

Of course the option of a combination of the above strategies is also available. As noted above, GPSG's approach employs a trace but constrains the appearance of that trace via metarules. Within CPSG, these meterules translate as lexical rules (much as in section 3.2), and there is the option of developing an approach using a binary feature, such as GPSG's NULL, whose distribution is governed by lexical rules. Another possible combination of approaches might involve the use of lexical rules for embedded subject extraction, phrasal rules for complement extraction, and a further schema for adjunct extraction. Such an approach is, however, clearly difficult to motivate, given that there seems to be little if anything to distinguish each of the cases.

For the purposes of the implementation discussed in Chapter 9 and listed in Appendix B, the lexical rule approach is favoured. This is despite the requirements it places on the ordering of application of lexical rules as discussed in section 3.2. The implementation applies the lexical rule of gap termination before that of subject-auxiliary inversion, thus ensuring that no problematic interactions occur.

4 Slash Percolation

In detailing the mechanism in CPSG for slash percolation, it is of some benefit to begin with the GPSG mechanism. Slash percolation in CPSG can then be seen to be effected by a reconstruction of GPSG's Foot Feature Principle.

4.1 Slash Percolation in GPSG

Slash percolation in GPSG is primarily governed by the Foot Feature Principle (FFP), though Feature Co-occurrence Restrictions (FCRs) and the Head Feature Convention (HFC) also play a crucial role. As SLASH in GPSG is a foot feature, the FFP requires that any SLASH specification that is instantiated on a daughter category of a local tree must also be instantiated on the mother category of that tree. The FFP only applies to *instantiated* foot features, and not to *inherited* foot features: foot features which are explicitly marked in immediate dominance rules or which have arisen through the operation of metarules. The principle therefore distinguishes between two different sorts of foot features. SLASH in GPSG is also a head feature, and hence its distribution is also subject to the HFC, which requires that, where possible, the SLASH feature of a mother

percolate to its head daughter. This, combined with FCR 6, which requires that lexical constituents be ¬SLASH, requires that, for non-lexically headed constituents, the SLASH feature should be shared by the mother and the head daughter. For lexically-headed constituents this need not be the case, as the requirements of the HFC would conflict with FCR 6.

4.2 A Reconstruction of the Foot Feature Principle

Crucial to GPSG's FFP is the division of the SLASH feature into two sorts: *instantiated* and *inherited*. An instantiated SLASH feature is one which has arisen through the application of FCRs and FSDs, or via the requirements of some principle. An inherited SLASH feature is one which is explicitly marked in an immediate dominance rule (which may or may not have arisen through the operation of metarules). The immediate dominance relations of GPSG are, for the most part, treated in HPSG and CPSG lexically in terms of the subcategorisation frames of heads, so inherited SLASH features may be seen, in HPSG and CPSG, in terms of constraints which a head applies to its sisters. Normally heads don't require any slash specifications on any of their sisters, but "tough" adjectives, for example, do. Instantiated SLASH features, on the other hand, arise from feature passing principles.

The distinction between the two sorts of SLASH feature motivates the subdivision of SLASH in CPSG into two set valued attributes: BOUND and ALL. Given this subdivision, a head may specify BOUND slash elements on its daughters, and any ALL slash elements on daughters which are not bound by the specification of the head percolate up to the mother. ALL must contain at least all members of BOUND. This allows the FFP to be reconstructed without employing any notion of default specification within CPSG. In the case of SLASH, percolation is governed by the requirement that all feature structures describing tokens of type phrasal constituent must extend:

$$\left[\begin{array}{ll} \text{SYN} & \left[\text{BIND}\ \left[\text{SLASH}\ \left[\text{ALL}\ \boxed{2}\right]\right]\right] \\ \text{DTRS} & \boxed{1} \end{array}\right]$$

where *dtrs-set*(1,3)

and *collect-slash*(3,2)

dtrs-set/2 holds if the second argument is the set of all descriptions of daughters in the first argument. *collect-slash/2* may be defined as follows:

$$\textit{collect-slash}(5,2) \text{ iff } 2 = \bigcup_{i \in 5} (X_i - Y_i)$$

where

$$i = \left[\text{SYN}\left[\text{BIND}\left[\text{SLASH}\left[\begin{array}{ll}\text{BOUND} & Y_i \\ \text{ALL} & X_i\end{array}\right]\right]\right]\right]$$

This corresponds to HPSG's Binding Inheritance Principle.

Given that, as discussed in Chapter 5, all daughters in a coordinate phrase must share identical binding features, this principle may in fact be stated at the headed-phrase node of the constituent hierarchy, with a more restrictive version of it being stated at the coordinate-phrase node, requiring that the ALL sets of each conjunct unify and that the BOUND sets of each conjunct unify and that the ALL set of the resultant is the set difference of the resultant ALL and BOUND attributes on any conjunct.

4.3 REL and QUE Again

Just as SLASH is not fully inherited in the case of missing object constructions, QUE is not inherited in some opaque contexts: 'Tigger wondered who would miaow' is −QUE, although 'who would miaow' has the description of 'who' in its QUE set (and hence is +QUE). Similarly, elements in the REL set are not inherited in [NP + RelCl] constructions. The mechanisms governing the percolation of SLASH therefore extend to REL and QUE, and the above may be extended to:

$$\left[\begin{array}{ll}\text{SYN} & \left[\text{BIND}\left[\begin{array}{ll}\text{SLASH} & [\text{ALL}\ 2] \\ \text{REL} & [\text{ALL}\ 3] \\ \text{QUE} & [\text{ALL}\ 4]\end{array}\right]\right] \\ \text{DTRS} & 1\end{array}\right]$$

where *dtrs-set*(1,5)

and *collect-slash*(5,2)

and *collect-rel*(5,3)

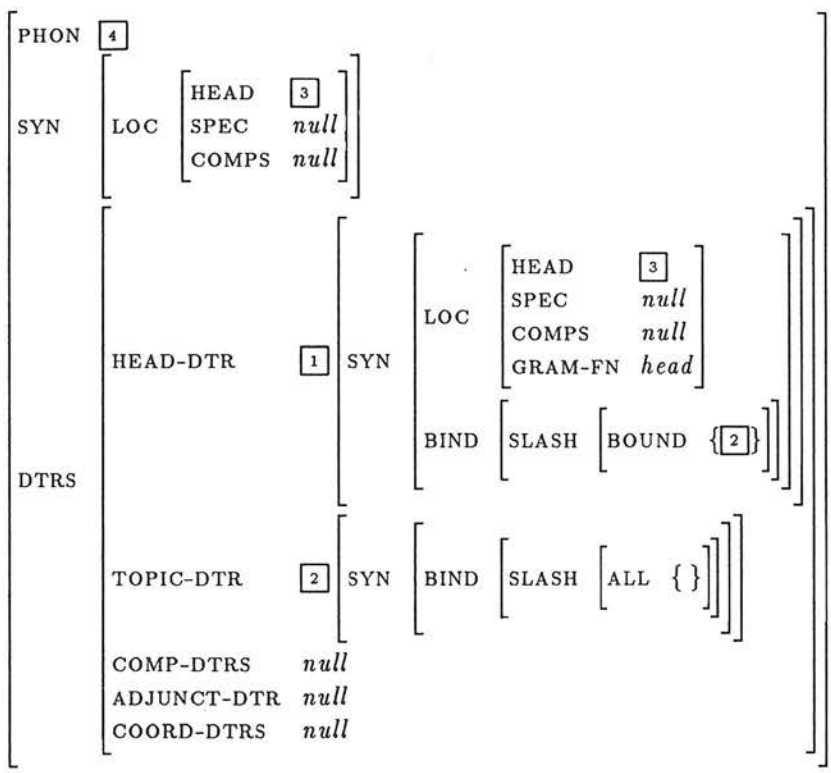
and *collect-que*(5,4)

where *collect-rel/2* and *collect-que/2* are defined in an analogous way to *collect-slash/2*.

5 Slash Binding

A further phrase structure rule is required to license the binding of SLASH, or equivalently, gap elimination. The rule states that a constituent may consist of an $\overline{\overline{X}}$ head with a singleton set value for its SYN|BIND|SLASH|BOUND attribute preceded by a constituent that unifies with the single member of that set. In less declarative terms, the clause initial constituent binds a single element on the SLASH set of the head of the phrase.

Any token of type head/filler phrase is (partially) described by:



where *order-phonology*(4,{1,2})

All varieties of topicalised phrases, including object relative clauses and inverted *wh*-questions, are examples of tokens of type head/filler phrase. As head/filler phrase is a subtype of headed phrase, CPSG's Binding Inheritance Principle applies, constraining the values of the SLASH attributes further: the value of SYN|BIND|SLASH|ALL for the

head daughter must be a superset of {[2]}, and the value of SYN|BIND|SLASH|ALL for the phrase as a whole must be the set difference of that superset and {[2]}. Normally the superset will be {[2]} itself and the set difference will be the empty set. By building into the rule a requirement like this, multiple topicalisation may be prevented.

Note that in this rule we have explicitly disallowed gaps within fillers by requiring that the filler daughter be partially specified as [SYN|BIND|SLASH|ALL { }]. (From the Binding Inheritance Principle SYN|BIND|SLASH|BOUND must also be { } as it must be a subset of SYN|BIND|SLASH|ALL.) As in the case of head/specifier phrases we might also have "hard-wired" word order into the rule, requiring fillers to precede their heads, rather than relying on the relation *order-phonology/2*.

The following token/type specifications outline this treatment of filler/gap dependencies:

To Sandy Kim gave Tigger _

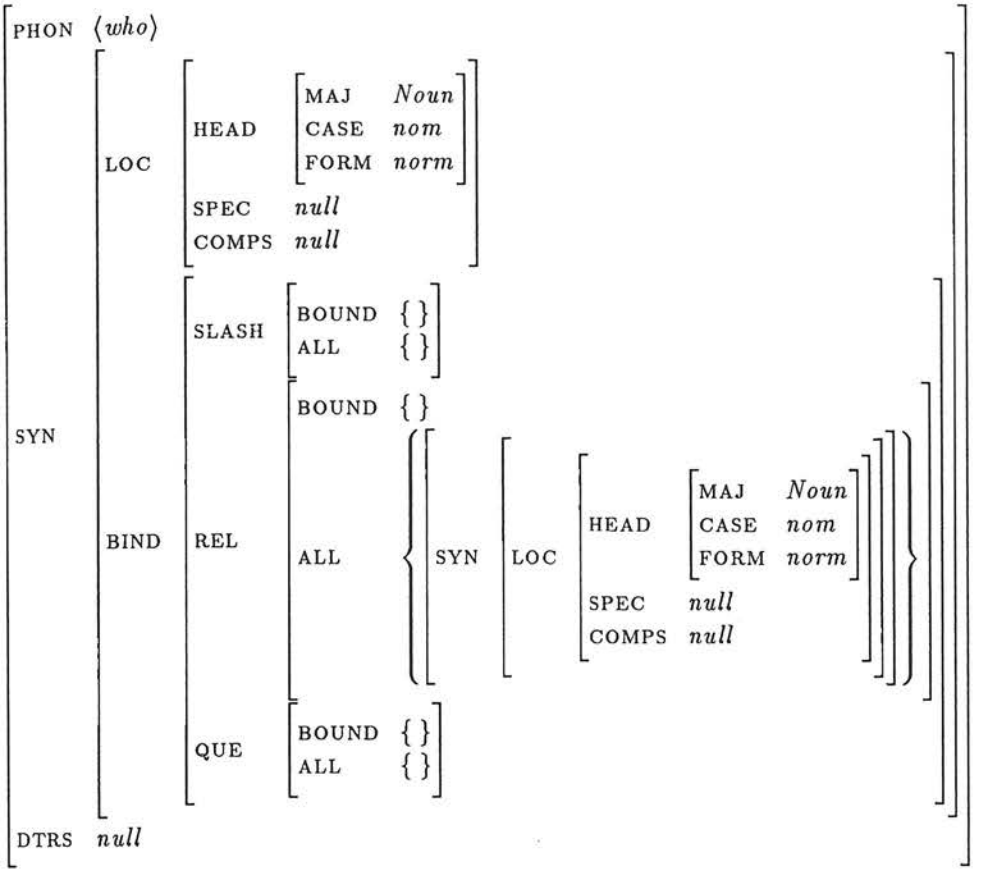
- 'gave Tigger _' is a token of type head/complement phrase.
- 'Kim gave Tigger _' is a token of type head/specifier phrase.
- 'To Sandy' is a token of type head/complement phrase.
- 'To Sandy Kim gave Tigger _' is a token of type head/filler phrase.

Sandy Kim pretends to give Tigger to _

- 'to _' is a token of type head/complement phrase.
- 'give Tigger to _' is a token of type head/complement phrase.
- 'to give Tigger to _' is a token of type head/complement phrase.
- 'pretends to give Tigger to _' is a token of type head/complement phrase.
- 'Kim pretends to give Tigger to _' is a token of type head/specifier phrase.
- 'Sandy Kim pretends to give Tigger to _' is a token of type head/filler phrase.

6 Relative Clauses

Following HPSG, CPSG uses the REL attribute to link relative pronouns with their antecedents. The description of the relative pronoun 'who' is:



'Whom' is similar except that its case, and the case of the element in the REL set, is *acc*. Most other lexical constituents are [SYN|BIND|REL|ALL { }]. As [Pollard 89b] points out, we really only need to store semantic information, and not syntactic information in the REL set: that the relative pronoun has nominative or accusative case is not important to the noun phrase which the relative clause containing the pronoun modifies.

The feature we have indicated in the preceding sections by +REL is realised as a non-empty set value for the REL attribute. Such a value originates on a relative pronoun and must be percolated up the phrase structure tree until it is "bound" by a noun which the relative clause modifies. Only in the case of object relative clauses is extraction involved. In such cases the relative pronoun (or some super-ordinate phrase) acts as a filler for a gap in a complement. Subject relative clauses are analysed as standard head/specifier phrases in which the relative pronoun (or some super-ordinate phrase) is unified with the

specifier of the verb. The binding inheritance mechanisms discussed in section 4 together with the appropriate lexical information (that non-relative pronouns have the empty set as values for both REL attributes) and minor extensions to the previous grammar rules, ensure that the REL attribute percolates correctly. Essentially, most grammar rules do not bind relative pronouns, and so must mark the SYN|BIND|REL|BOUND attributes of each of their daughters as { }. Only in the case of forming a modified noun phrase from a relative clause and another noun phrase is this not the case. Consequently we have the following extensions:

- head/topic phrases are described by:

$$\begin{bmatrix} \text{DTRS} & \begin{bmatrix} \text{HEAD-DTR} & [\text{SYN|BIND|REL|BOUND} \ \{\}] \\ \text{TOPIC-DTR} & [\text{SYN|BIND|REL|BOUND} \ \{\}] \end{bmatrix} \end{bmatrix}$$

- head/complement phrases are described by:

$$\begin{bmatrix} \text{DTRS} & \begin{bmatrix} \text{HEAD-DTR} & [\text{SYN|BIND|REL|BOUND} \ \{\}] \\ \text{COMP-DTRS} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

where *no-rels-bound*($\boxed{1}$)

no-rels-bound($\boxed{1}$) holds iff $\boxed{1}$ is a set of feature structures such that each element of $\boxed{1}$ is an extension of [SYN|BIND|REL|BOUND { }].

- head/adjunct phrases are described by:

$$\begin{bmatrix} \text{DTRS} & \begin{bmatrix} \text{HEAD-DTR} & [\text{SYN|BIND|REL|BOUND} \ \{\}] \\ \text{ADJUNCT-DTR} & [\text{SYN|BIND|REL|BOUND} \ \{\}] \end{bmatrix} \end{bmatrix}$$

Note again that, as with SLASH, the value of REL|BOUND for any constituent is determined by the head of the phrase it is involved in, whereas the value of REL|ALL is determined by its sub-constituents.

Some constraints on the occurrence of REL are necessary. As noted in section 1, +REL constituents may not head sentential head/topic phrases. Neither of the following schema are valid:

$$* \text{S}[+\text{REL}] \rightarrow \text{NP} \quad \text{S}[\text{SPEC NP}, +\text{REL}]$$

$$* \text{S}[+\text{REL}] \rightarrow \text{XP} \quad \text{S}[\text{SLASH } \{\text{XP}\}, +\text{REL}]$$

These schema lead to the putative relative clauses in (16):

(16) a. * ... the kitten [Tigger [likes pictures of whom]]

b. * ... the kitten [pictures of Tigger [who likes _]]

+REL constituents may, however, head nominal head/topic phrases, as in '[the [picture of whom]]'. Two options for enforcing the required constraint are:

- differentiate determiners (as specifiers to $\overline{\text{N}}$s) from $\overline{\overline{\text{N}}}$ (as subjects of $\overline{\text{V}}$s), licensing $\overline{\overline{\text{N}}}$s and $\overline{\overline{\text{V}}}$s with different rules. This is in line with the treatment of [Borsley ∞] as discussed in section 4.4 of Chapter 3 and would correlate with the semantic differences discussed in Chapters 7 and 8.

- employ feature co-occurrence restrictions, requiring any constituent which is a verbal head to be −REL.

Given that we have an analysis of how relative clauses may be licensed as constituents, it remains to indicate how they may modify $\overline{\text{N}}$s. There are at least three options:

- We could require that heads are marked for adjuncts, as in standard HPSG, and as we have argued against in Chapter 3.

- We could posit a further type of phrase corresponding to the phrase structure rule:

$$\overline{\text{N}} \rightarrow \overline{\text{N}} \quad \text{S}[+\text{REL}]$$

- We could introduce feature co-occurrence restrictions, as in GPSG, requiring that any constituent with the features corresponding to those of a relative clause also have those of an $\overline{\text{N}}$ adjunct. This would involve overriding the value for the adjunct attributes specified by the Head Feature Principle, which in turn would require a theory of defaults in the phrasal hierarchy.

We leave a decision on this point to future research, though presumably the mechanism will require that there be exactly one element in the REL|ALL set of the relative

clause, and that the REL|BOUND attribute of the relative clause be shared with this set, capturing the fact that the relative pronoun is bound when the relative clause attaches to a noun phrase. Binding inheritance mechanisms will then ensure that the description of the modified noun phrase be the empty set, corresponding to the fact that the REL element has become bound. That is, the +REL specification on a relative clause must not percolate up to its mother phrase. We may also apply constraints on SLASH to whatever mechanism we choose, requiring that relative clauses are specified as [SYN|BIND|SLASH|ALL { }], forcing them to be extraction islands and thus enforcing the complex noun phrase constraint of [Ross 67].

7 *Wh*-Questions

We treat questions in much the same way as relative clauses. *Wh*-elements introduce an element to the QUE|ALL set. The feature +QUE is realised as a non-empty set value for the QUE|ALL attribute. Unlike the case of English relative clauses, this set may contain more than a single element, as it will for the description of the question 'Who chased whom?'.

The binding inheritance mechanisms apply to QUE attributes in much the same was as they do to SLASH and REL attributes, though QUE is less constrained than REL. The main problem lies in accounting for inversion in topicalised questions: +QUE constituents may only serve as fillers to inverted gapped phrases.

(17) a. Tom chases whom?

b. *Whom Tom chases _?

c. *Whom Tom did chase _?

d. Whom did Tom chase _?

Again, it seems that a feature co-occurrence restriction might be required, stating that any constituent with a +QUE filler daughter must be inverted. In notation similar to that of GPSG this might be written as:

$$\left[\text{DTRS|HEAD-DTR|SYN|BIND|SLASH|BOUND} \left\{\left[\text{SYN|BIND|QUE|ALL } \{\_\}\right]\right\}\right] \supset \left[\text{SYN|LOC|HEAD|INV } -\right]$$

8 Assimilating SPEC and SLASH

Given the above noted similarities between specifiers and fillers, it is of interest to ask just how far the parallels can be pushed. In this section we briefly outline two proposals that do away with the distinction altogether. Each proposal is clearly very simplistic and both need further development. Though they illustrate an interesting direction for future research, they should each be seen as tentative and speculative. Problems with the attempted assimilations are mentioned and no attempt has been made to incorporate them into the computational implementation.

8.1 Specifiers and Fillers as Topics

One way to assimilate specifiers and fillers into a single category might be to revise the role and treatment of complements, essentially allowing specifiers to be complements under certain conditions. To do this we replace the SPECIFIER attribute by an attribute TOPIC, whose value is either a single feature structure describing a constituent or *null*. The attribute COMPLEMENTS remains as before, taking either a set of feature structures or *null* as its value. As before, all arguments are marked for a grammatical function by the constituent which subcategorises for them. For head/complement, head/specifier and head/adjunct phrases all is therefore as before except that the SPECIFIER attribute is relabelled TOPIC. However, we do away with the SLASH attribute and the head/filler rule, as well as the slash termination lexical rules suggested in section 3. We replace these by a lexical rule acting on the attributes TOPIC and COMPLEMENTS, which roughly takes the form:

$$\left[\text{SYN}\left[\text{LOC}\left[\begin{array}{l}\text{TOPIC}\ \boxed{1}\\ \text{COMPS}\ \{\ldots\boxed{2}\ldots\}\end{array}\right]\right]\right]$$

$$\Downarrow$$

$$\left[\text{SYN}\left[\text{LOC}\left[\begin{array}{l}\text{TOPIC}\ \boxed{2}\\ \text{COMPS}\ \{\ldots\boxed{1}\ldots\}\end{array}\right]\right]\right]$$

This rule thus swaps the topic of a lexical entry with some other complement. Consequently, the head/complement rule and the head/topic rule are sufficient to generate structures containing phrase bounded filler-gap dependencies (i.e., dependencies such as that in 'That kitten Tigger despises', but not dependencies such as that in 'That kitten

Tom believes Tigger despises'). There is no analogue of slash percolation in this naïve system.

Given this system, we have the structure illustrated in Figure 4.1 for head/specifier phrases and the structure illustrated in Figure 4.2 for head/filler phrases, both of which are admitted by the head/topic rule.

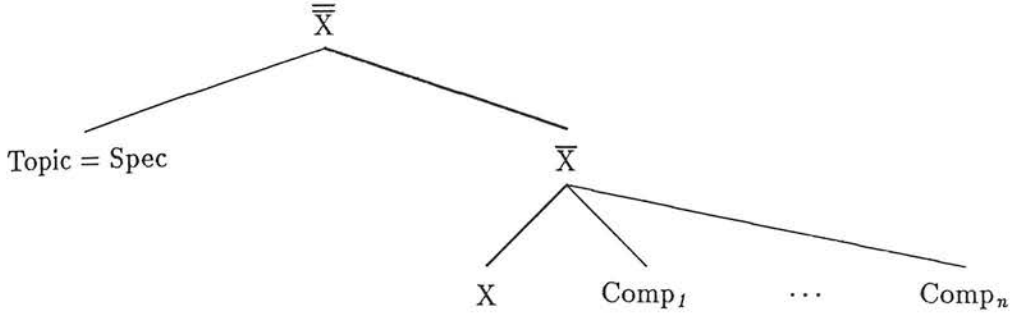


Figure 4.1: Alternate Phrase Structure (Head/Specifier Phrases)

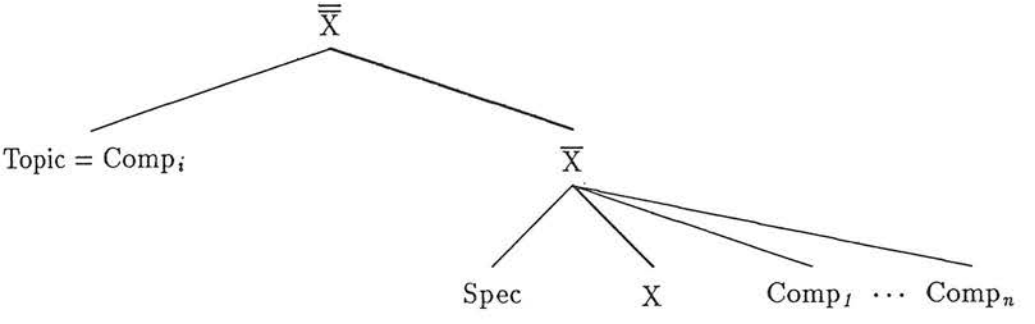


Figure 4.2: Alternate Phrase Structure (Head/Filler Phrases)

To account for word order we introduce one further binary head feature, TOP, which is specified as + on topics by the rule which licenses constituents of type head/topic phrase. All non-topic constituents are marked as [TOP −] by the rules which integrate them into larger structures (and in particular the head/complement rule). The relation *order-phonology* must then implement the (transitive closure of the) following linear precedence rules:

$$\text{XP}[+\text{TOP}] \prec \text{XP}[-\text{TOP}]$$

$$\text{XP}[+\text{SUBJ}] \prec \text{XP}[+\text{HEAD}] \prec \text{XP}[+\text{OBJ}] \prec \text{XP}[+\text{I-OBJ}] \prec \text{XP}[+\text{VCOMP}]$$

The order of these linear precedence rules is important. In particular, the first rule overrides all other rules: the topic precedes all other constituents in a phrase, and the subject precedes the head. Note that, as illustrated in Figure 4.2, within this system it is not the case that lexical heads precede *all* their complements.

Under this system, the two sentential structures:

$$S \rightarrow XP \quad S[\textsc{spec}\ XP]$$

$$S \rightarrow XP \quad S[\textsc{slash}\ \{XP\}]$$

are collapsed into a single structure:

$$S \rightarrow XP \quad S[\textsc{topic}\ XP]$$

8.2 Specifiers as Fillers

If we were to allow subject extraction, all verb phrases would be analysable as either standard verb phrases or sentences with extracted subjects. This leads to an ambiguity in basic sentence structure:

$$S \rightarrow XP \quad S[\textsc{spec}\ XP]$$

$$S \rightarrow XP \quad S[\textsc{slash}\ \{XP\}]$$

One way to avoid this ambiguity is to do away with one of the rule schemas, so that there is in fact only a single rule. This may be done by treating specifiers as fillers. If our lexicon contains entries of the form:

$$\begin{bmatrix} \textsc{phon} & \textit{appears} \\ \textsc{syn} & \begin{bmatrix} \textsc{loc} & \begin{bmatrix} \textsc{comps} & \{\text{VP[inf]}\} \end{bmatrix} \\ \textsc{bind} & \begin{bmatrix} \textsc{slash} & \{\text{NP[nom]}\} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

where the attribute SPEC is no longer employed, then the schema

$$S \rightarrow XP \quad S[\textsc{spec}\ XP]$$

is not required. Under this proposal slash termination and slash percolation may proceed as originally outlined. The head/filler schema does, however, need some revision. In cases which would not normally be analysed as involving an unbounded dependency, little need change — the above head/filler rule schema is adequate. In other cases however, the slash set will contain multiple elements. A first generalisation therefore is:

$$\text{S[SLASH } \{\beta, \ldots\}] \rightarrow \alpha \text{ S[SLASH } \{\alpha, \beta, \ldots\}]$$

This, together with a suitable rule of slash termination, over generates, allowing, for example, a subject and an object to be extracted, with the subject being extracted to the left of the object, as in (18):

(18) [Tigger]$_i$ [Fido]$_j$ $\_\_i$ chases $\_\_j$

Noting that complements, whether or not they be extracted, are marked by their head with a grammatical function, we may restrict the above schema such that the set $\{\beta, \ldots\}$ on the left hand side does not contain a constituent whose grammatical function is *subject*. This restriction prevents the over-generation (where embedded subject extraction is not involved), and yields (almost traditional) phrase structures such as those in Figure 4.3 for topicalised sentences.

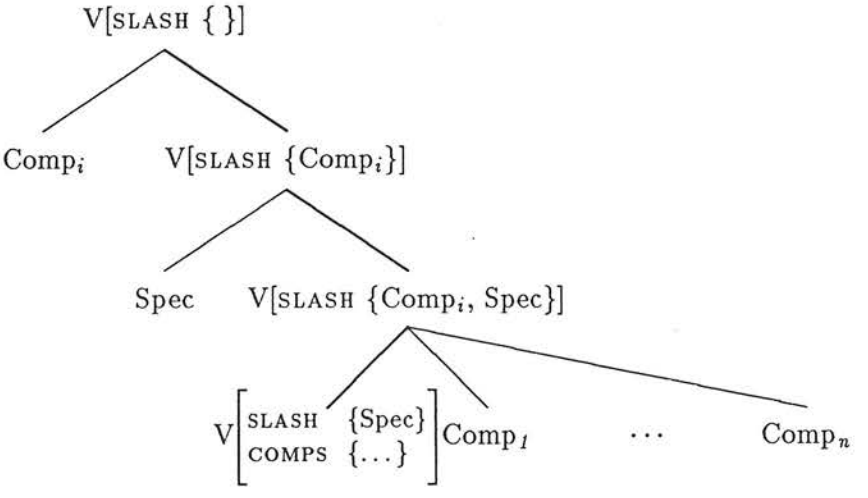


Figure 4.3: Alternate Phrase Structure (Specifiers as Fillers)

There remain problems with this approach in cases of embedded subject extraction. In such cases, more than one element of the slash set may have the grammatical function *subject*. Consequently the above schema, with the restriction on subjects, cannot apply — the set on the left hand side of the schema cannot be required not to contain a subject element in these cases. It is not clear, however, how such a schema could in general be restricted to allow the extraction of embedded subjects and still not over-generate. It would seem any attempt to allow '[Tigger]$_i$ I believe $\_\_i$ miaows' will not dissallow

'I [Tigger]$_i$ believe $\__i$ miaows'. The assimilation of SLASH and SPEC thus remains a question for future research.

9 Summary

In this chapter we have adopted an approach to unbounded dependencies which follows that of GPSG. The approach involves three parts: slash introduction, slash percolation and slash termination. With respect to slash introduction, we have argued, on the basis of empirical evidence, that head/filler phrases, such as topicalised clauses, and head/specifier phrases have several features in common, and as such may be treated as subtypes of a type we have called head/topic phrase. With respect to slash percolation we have reconstructed GPSG's Foot Feature Principle, which governs, to a large extent, the percolation of the SLASH feature, without employing any notion of default. A notion of default is crucial to the GPSG account. Thirdly, with respect to slash termination, we have presented three options which are available to the CPSG treatment. Each of these options has failings, and as such we do not select from amongst them. The chapter also considers the features REL and QUE, whose distribution is governed by similar principles to the that of SLASH. Constraints on the distribution of binding features are discussed only minimally. This is clearly an area where future research is required.

The last section of the chapter explores the suggestion that head/filler phrase and head/specifier phrase be assimilated into a single type. As it stands, each is taken to be a subtype of head/topic phrase, but the parallels between the two suggest that further assimilation might be possible. The section is primarily exploratory, and again represents an area which would benefit from further research.

Chapter 5

Coordination

Very little is said in [Pollard & Sag 87] about how coordination might be treated within HPSG. In this chapter we develop a treatment of constituent coordination, drawing again from GPSG. The behaviour of head features under coordination means that we cannot directly translate the GPSG approach into CPSG without first extending our domain of descriptions. Essentially the problem comes down to admitting composite objects: objects which are, for example, a cross between a noun phrase and an adjectival phrase. We begin this chapter by illustrating this problem, and considering various proposed "solutions" within domains of feature structures. We then modify the Rounds-Kasper Logic presented in Chapter 1 to allow composite objects. An earlier version of this material appears in [Richard Cooper 90b]. The remainder of the chapter considers the other issues involved in constituent coordination, looking at subcategorisation and binding (section 2), phrases consisting of a head preceded by a conjunction (section 3), and the two subtypes of coordinate phrase, binary coordinate phrase and iterated coordinate phrase (section 4). We do not in this chapter discuss the agreement features or semantics of coordinate structures. Agreement in CPSG is treated semantically, and these topics are discussed in Chapters 7 and 8.

Note that all of HPSG's grammar principles presented in [Pollard & Sag 87] are keyed to headed phrases: the Head Feature Principle, the Subcategorisation Principle, and the Semantics Principle all take the form of conditionals, the antecedents of which restrict their applicability to headed phrases. Coordinate phrases are thus not constrained by these principles. Within CPSG, this correlates with the fact that coordinate phrase

and headed phrase are extensionally disjoint subtypes of phrasal constituent, and so, for example, the equivalent of HPSG's Head Feature Principle, which is stated at the headed phrase node, does not apply to constituents of type coordinate phrase.

1 Head Features

The importance of the notion of *head* in HPSG and CPSG raises the question of whether coordinate structures should be treated as having a head, and if so what that head should be. Following GPSG (cf. [Sag *et al.* 84], [Gazdar *et al.* 85]), we might regard such structures as multi-headed, with each conjunct being a head. This approach has also been adopted by Proudian and Goddeau ([Proudian & Goddeau 87]) and M^{c}Intyre ([McIntyre 89]) in the context of HPSG, and by Dowty ([Dowty 85]) and Steedman ([Steedman 85], [Steedman 87]) in the context of categorial grammar. Other categorial treatments (cf. [Lambek 58], [Geach 72], [Morrill 89]) take coordinate structures to be singly headed, with the conjunction being the head. Examples involving more than one constituent marking the conjunction, such as those in (1), suggest that this second option, that of taking the conjunction as the head, is not feasible as there need not be a unique conjunction.

(1) a. either α or β

b. both α and β

c. α and β and γ and ... and ω

d. neither α nor β nor γ nor ... nor ω

This thus argues against treating, for example, *and* as a constituent of polymorphic category x which subcategorises for a set of constituents of category x, where x ranges over some set of syntactic categories. Nevertheless, unless we are willing to redesign all of our grammar rules, we must describe coordinate structures by feature structures bearing the syntactic attributes born by the single conjuncts. To see this, consider the case of constituents of type head/specifier phrase. Clearly it is desirable to have a single grammar rule license all head/specifier phrases, whether the head, the specifier or both are coordinate structures or not. The current grammar rule licensing constituents of type head/specifier phrase refers to various SYNTAX attributes of the head daughter of

such constituents, including the HEAD attribute, other LOCAL attributes, and BINDING attributes. Thus if a coordinate phrase is to head a constituent of type head/specifier phrase, it must be specified for these SYNTAX attributes. Thus coordinate structures, whether they be regarded as headed phrases or not, must have descriptions bearing the usual SYNTAX attribute, itself having LOCAL and BINDING attributes, and all attributes of descriptions of typical phrasal constituents, including a HEAD attribute. Note that no grammar rules refer to the DAUGHTERS attribute of any daughters, so the existing grammar rules do not impose any requirements on the "internal structure" (i.e., the daughters) of (descriptions of) coordinate structures.

1.1 The Problem

Given that the description of a coordinate construction must involve the same syntactic attributes as each of the conjuncts in the construction, any treatment of coordination must specify how the values of the syntactic attributes of the description of a coordinate structure are related to the values of the syntactic attributes of the individual conjuncts. It is well known that coordination is not restricted to like categories (see (2)), so it is too restrictive to require that all conjunct daughters have identical values for syntactic attributes, and that these values are also the values of the syntactic attributes of the description of the entire construction.

(2) a. Tigger became famous and a complete snob

b. Tigger is a large bouncy kitten and proud of it

Furthermore, it is only possible to coordinate certain phrases within certain syntactic contexts. Whilst the examples in (2) are grammatical, those in (3) are not, although the same constituents are coordinated in each case. The difference is the syntactic context in which the coordinated phrase appears.

(3) a. *Famous and a complete snob chased Fido

b. *A large bouncy kitten and proud of it likes Tom

1.2 GPSG's Solution

The relevant generalisation, made by [Sag *et al.* 84] with respect to GPSG and rephrased here in terms of CPSG, is that constituents may coordinate if and only if the description of each constituent is consistent with the relevant description in the grammar rule which licenses the phrase containing the coordinate structure. Example (2a) is grammatical because the phrase structure rule which licenses the constituent 'became famous and a complete snob' requires that 'famous and a complete snob' be consistent with the partial description of the object subcategorised for by 'became', and the descriptions of each of the conjuncts, 'famous' and 'a complete snob', actually are consistent with that partial description. In an HPSG-like framework, 'became' might subcategorise for an object described by:

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{Adjective} \vee \textit{Noun} \\ \text{PRED} & + \end{bmatrix} \\ \text{GRAM-FN} & \textit{obj} \\ \text{SPEC} & \textit{null} \\ \text{COMPS} & \textit{null} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

'Famous' is described by:

$$\begin{bmatrix} \text{PHON} & \langle \textit{famous} \rangle \\ \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{Adjective} \\ \text{PRED} & + \end{bmatrix} \\ \text{SPEC} & \textit{null} \\ \text{COMPS} & \textit{null} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

which is consistent with the object description. 'A complete snob' is described by:

$$\begin{bmatrix} \text{PHON} & \langle \textit{a complete snob} \rangle \\ \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{Noun} \\ \text{PRED} & + \end{bmatrix} \\ \text{SPEC} & \textit{null} \\ \text{COMPS} & \textit{null} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

which is similarly consistent with the object description.

(2b) is grammatical for analogous reasons. (3a) is ungrammatical as 'chased' requires that its subject bear the feature specification [SYN|LOC|HEAD|MAJ *Noun*]. Whilst this specification is born by 'a complete snob', it conflicts with that born by 'famous', so the description of 'famous' is not consistent with the description which 'chase' requires of its subject. Similarly the second conjunct in (3b) bears a feature specification which conflicts with [SYN|LOC|HEAD|MAJ *Noun*], which is required of the subject of 'likes', and so the sentence is ungrammatical. Note that this approach implies that for (almost) any constituents α and β, the string 'α and β' may occur as a subconstituent of some constituent, provided that constituent provides a suitable context.

1.3 Two Approaches to a Solution

Two approaches to this problem are immediate. Firstly, we may try to capture the intuition that each conjunct must satisfy the requirements of the appropriate grammar rule by generalising all grammar rules to allow for coordinated phrases in all positions. This approach follows that of [Shieber 89], and involves the use of semi-unification. It does not involve a grammar rule licensing 'α and β' as a constituent, rather the grammar rules must be generalised so that if constituents both of the form '$X\ \alpha\ Y$' and '$X\ \beta\ Y$' are licensed, then constituents of the form '$X\ \alpha\ and\ \beta\ Y$' and '$X\ \beta\ and\ \alpha\ Y$' (and so on for all other conjuncts) must also be licensed.

An alternate approach is to preserve the original grammar rules, but generalise the notion of syntactic category to license composite categories, categories built from other categories, and introduce a rule licensing coordinate structures which have such composite syntactic categories. That is, if α and β are constituents, we allow the formation of a constituent 'α and β', whose syntactic category is a composite of the syntactic categories of α and β. Within a feature-based approach, this generalisation of syntactic category requires a generalisation of the logic of feature structures, with an associated generalisation of the subsumption ordering and feature structure compatibility. This is the approach which we adopt. One of the consequences of this approach is that for (almost) any constituents α and β, the grammar should also license the string 'α and β' as a constituent, irrespective of whether there are any contexts in which this constituent may occur.

1.4 Unification-Based "Solutions"

The problem remains then of just what the description of a coordinate structure should be like, and how that description may be arrived at from the descriptions of the individual conjuncts. Two possible solutions based on feature structure unification have been suggested.

1.4.1 Generalisation

One possibility is to take the value of the head feature of the coordinate phrase to be the *generalisation* (see section 5 of Chapter 1) of the values of the head features of the conjuncts and require identity of all other syntactic feature value pairs (cf. [McIntyre 89]). There are two serious problems with this "solution".

Firstly, the generalisation of any two feature structures, all of whose features conflict, is the feature structure [ ], i.e., the feature structure in which no attributes have defined values. This is consistent with any feature structure, so, for example, in the case of 'likes', which requires an object whose description is consistent with

$$\left[\begin{array}{ll}\textsc{syn} & \left[\begin{array}{ll}\textsc{loc} & \left[\begin{array}{ll}\textsc{head} & \left[\begin{array}{ll}\textsc{maj} & \textit{Noun}\\ \textsc{case} & \textit{acc}\\ \textsc{form} & \textit{norm}\end{array}\right]\\ \textsc{spec} & \textit{null}\\ \textsc{comps} & \textit{null}\end{array}\right]\end{array}\right]\end{array}\right]$$

generalisation will license 'likes [there and to give Tigger a bone]' as a valid verb phrase, as 'there and to give Tigger a bone' will satisfy [SYN|LOC|HEAD [ ]]. Generalisation clearly does not capture the relationship which [Sag *et al.* 84] propose.

The second problem arises from the possibility of disjunctive values. The verb 'become' subcategorises for an object which, as the examples in (4) (modified from [Pollard & Sag 87, p. 122]) demonstrate, must bear (at least) the specification:

$$\left[\begin{array}{ll}\textsc{syn} & \left[\begin{array}{ll}\textsc{loc} & \left[\begin{array}{ll}\textsc{head} & \left[\textsc{maj} \quad \textit{Adjective} \vee \textit{Noun}\right]\\ \textsc{spec} & \textit{null}\\ \textsc{comps} & \textit{null}\end{array}\right]\end{array}\right]\end{array}\right]$$

(4) a. Terry became quite mad

b. Terry became a complete maniac

c. *Terry became out of his mind

These examples show that 'became' may subcategorise for an NP or an AP, but crucially not a PP, and hence the possible values of the head attributes must be disjunctively specified.

If, however, we take the head features of a coordinate structure to be the generalisation of those of the conjuncts, then each of the examples in (5) will be licensed, including those that are ungrammatical.

(5) a. Terry became quite mad and a complete maniac

b. *Terry became quite mad and out of his mind

c. *Terry became a complete maniac and out of his mind

The real problem is that generalisation ignores conflicting values. Values that conflict are important and must not be ignored.

1.4.2 Unification$^+$

[Proudian & Goddeau 87] suggest an alternative whereby the head features of a coordinate structure are determined from the head features of the conjuncts via an operation they refer to as *unification*$^+$. Unification$^+$ is defined as:

> unification$^+(f_1, f_2)$, where f_1 and f_2 are feature structures, is the feature structure whose attributes are the union of the attributes of f_1 and f_2 such that for any attribute a
>
> - if $f_1(a) \sqcup f_2(a)$ is defined then unification$^+(f_1, f_2)(a) = f_1(a) \sqcup f_2(a)$
> - if $f_1(a)$ is defined and $f_2(a)$ is defined but $f_1(a) \sqcup f_2(a)$ is undefined (i.e., $f_1(a)$ and $f_2(a)$ conflict) then unification$^+(f_1, f_2)(a) = \text{CONFL}$
> - otherwise unification$^+(f_1, f_2)(a)$ is undefined.

Unification$^+$ is similar to unification except that it cannot fail. If the values for a given attribute of the arguments to unification$^+$ conflict, the result of unification$^+$ for that attribute is the distinguished value CONFL (mnemonic for "conflict"). CONFL thus marks which attributes would cause unification failure, rather than leaving them undefined as in the case with generalisation. CONFL is an atomic value and as such cannot unify with other atomic values or complex values. It is only introduced in coordinate structures, and then only via unification$^+$. Hence, if, for example, a verb is specified as taking a noun phrase subject, then the SYN|LOC|HEAD|MAJ attribute of the description of a possible subject must have value *Noun*, it cannot be CONFL, so the use of unification$^+$ only allows *Noun* type things to coordinate in the verb's subject position. If however, the verb is unspecified for the syntactic category of object it takes (as in the case of 'is'), i.e., the SYN|LOC|HEAD|MAJ attribute of the object in the verb's complement set is undefined, then it can take an object, such as 'a large bouncy kitten and proud of it', whose SYN|LOC|HEAD|MAJ feature has value CONFL.

Proudian and Goddeau's solution is more descriptively adequate than that using generalisation, but it too suffers from the difficulty illustrated in (5). Proudian and Goddeau do not indicate how they would treat 'become', which appears to require a disjunctive specification for the syntactic category of its object. This disjunctive value cannot be specified as CONFL (as then none of the examples in (4) would be licensed). If it is specified as *Adjective* $\vee$ *Noun*, then CONFL, being an atomic value, will not be consistent with it, and so the legitimate coordinated examples will not be licensed.

1.5 An Algebra of Syntactic Categories

The problem with each of these suggested solutions is that they do not accurately represent the condition on coordinate structures expressed by [Sag *et al.* 84]. The failing arises from the fact that the feature structure descriptions constructed for coordinate phrases from the individual conjuncts are not sufficiently rich to allow the distinctions to be drawn which the condition requires. The examples above with disjunction demonstrate that values for the syntactic attributes of the description of a coordinate structure must not conceal the values for the syntactic attributes of the individual conjuncts. This, together with the fact that feature structures are simply descriptions of tokens and types,

suggests that in the case of coordinated phrases it may be reasonable to employ a kind of "conjunctive" value: in some sense the syntactic category of 'famous and a complete snob' is a composite category consisting of *Adjective and Noun*.

As explained in section 5 of Chapter 1, we cannot simply add the classical connective '$\wedge$' to the semantic domain of feature structures, as this will interact with the definition of unification, allowing conflicting values to unify ($\alpha \sqcup \beta$ will just be $\alpha \wedge \beta$, regardless of whether α and β conflict or not). Rather, we look more closely at the logic of feature structures.

1.5.1 Composite Atomic Values

We have taken feature structures to describe deterministic finite automata. In the original formulation of [Rounds & Kasper 86], [Kasper & Rounds 86] and [Kasper & Rounds 90], these automata have atomic values assigned to (some of) their terminal states. The coordination data suggest that these values need not be atomic, and that there is structure on the domain of "atomic" values. To model this structure we introduce the *conjunctive composition* operator, "$\oplus$", which is an operator on the domain of atomic values such that if α and β are atomic values, then $\alpha \oplus \beta$ is also an atomic value. Informally, if 'a large bouncy kitten' is described by the pair [CATEGORY NP] and 'proud of it' is described by the pair [CATEGORY AP], then any conjunction of those constituents, such as 'neither a large bouncy kitten nor proud of it' will be described by the pair [CATEGORY $NP \oplus AP$].

For full generality we take $\oplus$ to be an operator on non-empty finite subsets of atomic values, though in the binary case we shall use the usual infix notation and write it as a binary operator. The use of such subsets yields several immediate properties of $\oplus$ which are not obvious when it is written as a binary infix operator. $\oplus$ is idempotent, corresponding to the fact that the conjunction of two (or more) noun phrases is still a noun phrase, and so we have for all atomic values α,

$$\alpha \oplus \alpha = \oplus\{\alpha\} = \alpha$$

$\oplus$ is also symmetric: a noun phrase coordinated with an adjectival phrase is of the same category as an adjectival phrase coordinated with a noun phrase. Thus for all atomic

values α and β,

$$\alpha \oplus \beta = \oplus\{\alpha, \beta\} = \beta \oplus \alpha$$

Finally, $\oplus$ is associative:

$$(\alpha \oplus \beta) \oplus \gamma = \oplus\{\alpha, \beta, \gamma\} = \alpha \oplus (\beta \oplus \gamma)$$

We further require that for any α,

$$\oplus\{\alpha\} = \alpha$$

Given this structure on the domain of atomic values, we modify the satisfiability requirements. Firstly,

- $\mathcal{A} \models \oplus\{\alpha_1, \ldots, \alpha_n\}$ iff $\mathcal{A} = \langle\{q_0\}, q_0, L, \delta, A, \pi\rangle$ where $\delta(q_0, l)$ is undefined for each l in L and $\pi(q_0) = \oplus\{\alpha_1, \ldots, \alpha_n\}$.

This is really just the same clause as for all atomic values:[1]

- $\mathcal{A} \models \alpha$ iff $\mathcal{A} = \langle\{q_0\}, q_0, L, \delta, A, \pi\rangle$ where $\delta(q_0, l)$ is undefined for each l in L and $\pi(q_0) = \alpha$.

Most importantly though, we replace previous occurrences of disjunction with an operator for *disjunctive composition*, "$\otimes$", which is similar to disjunction, but interacts appropriately with $\oplus$. A single state DFA satisfies a disjunctive composite description iff it satisfies one of the disjuncts *or* it satisfies the composite:

- $\mathcal{A} \models (\alpha \otimes \beta)$ where $\alpha, \beta \in A$ iff $\mathcal{A} \models \alpha$ or $\mathcal{A} \models \beta$ or $\mathcal{A} \models (\alpha \oplus \beta)$.

That is, $\mathcal{A} \models (\alpha \otimes \beta)$ iff $\mathcal{A} = \langle\{q_0\}, q_0, L, \delta, A, \pi\rangle$ where $\delta(q_0, l)$ is undefined for each l in L and $\pi(q_0) \in \{\alpha, \beta, (\alpha \oplus \beta)\}$.

This may be generalised to disjunction of non-empty finite subsets:

- $\mathcal{A} \models \otimes\Phi$ where $\Phi \subset A$ iff $\mathcal{A} \models \oplus\Psi$ for some non-empty subset Ψ of Φ.

[1] By an "atomic value", in this section we mean an element of the domain A. The structure which we have introduced on A means that strictly speaking these values are not atomic. They are, however, "atomic" in the feature structure sense: they have no attributes.

The intuition behind this modification stems from the fact that if a constituent has a disjunctive subcategorisation requirement, then that requirement can be met by any of the disjuncts, *or* some composite of those disjuncts. Thus by replacing disjunction (which is only in the syntax) with $\otimes$ (which is also only in the syntax) we are saying that such a description can describe any of the disjuncts (which are semantic objects) or some composite of those disjuncts (which is also a semantic object). For a more concrete example, if a verb subcategorises for something which is either an NP or an AP, then that subcategorisation requirement may be legitimately met by either an NP or an AP or a composite NP$\oplus$AP.

1.5.2 Composite Feature Structures

This use of an algebra of atomic values has assumed that composites may only be formed at the atomic level. That is, whilst we may form $\alpha \oplus \beta$ for α, β atomic, we may not form

$$\left[\textsc{cat}\ \ \alpha\right] \oplus \left[\textsc{cat}\ \ \beta\right]$$

However, such composites do appear to be necessary. In particular, in CPSG we must be able to form the composite of the head features of all conjuncts in a coordinate structure, and for this reason the previous subsection was really only illustrative. Unfortunately that development does not generalise in a straightforward manner to allow composite feature structures. In particular, whilst the intuitive behaviour of the connective remains as above, we must revise the semantic domain before defining satisfiability.

The generalisation of satisfiability of disjunctive composite formulae holds:

$\mathcal{A} \models \otimes\Phi$ iff $\mathcal{A} \models \oplus\Psi$ for some subset Ψ of Φ.

We must alter the semantic domain, the domain of deterministic finite automata, however, to allow a sensible rendering of satisfaction of conjunctive composite formulae — at first gloss it would seem that we need composite states, rather than composite atomic values, however, it is not clear how the state transition function, δ, should behave for such composite states. Defining

$$\delta(q_1 \oplus q_2, l) = \delta(q_1, l) \oplus \delta(q_2, l)$$

(perhaps with restrictions depending on when $\delta(q_1, l)$ and $\delta(q_2, l)$ are defined) is not quite right. This would effectively push all composite structure to atoms, resulting in the undesirable equivalence

$$\left[\text{ATT} \quad v_l\right] \oplus \left[\text{ATT} \quad v_2\right] = \left[\text{ATT} \quad v_l \oplus v_2\right]$$

Paralleling the treatments of sets in [Rounds 88], we might maintain atomic states and introduce a new type of transition, call it a ρ-transition, and allow δ to be relational on ρ-transitions (thus sacrificing determinism), so that for a given state q, there may be more than one q' such that $\delta(q, \rho) = q'$. (We abuse notation by maintaining the functional notation for the relation.) We further require that if for some state q, $\delta(q, \rho)$ is defined, then $\delta(q, l)$ is undefined for each l in L. This ensures that all non-final states have either "ordinary" transitions leaving them or ρ-transitions leaving them, but not both. A state with ρ-transitions leaving it corresponds to a composite state. We may then define:

- $\mathcal{A} \models \oplus\{\phi_1, \ldots, \phi_n\}$ where $\phi_i \in \mathcal{LF}$ and $\mathcal{A} = \langle Q, q_0, L, \delta, A, \pi \rangle$ iff for each ϕ_i there is a state q_i such that $\delta(q_0, \rho) = q_i$ and $\mathcal{A}_i \models \phi_i$, where $\mathcal{A}_i = \langle Q_i, q_i, L, \delta, A, \pi \rangle$, and Q_i is the subset of Q such that each state is connected to q_i by δ.

The principal undersirable aspect of this approach is that it distinguishes between ϕ and $\oplus\{\phi\}$: no automata can satisfy both, yet we would like them to be identical. This may be circumvented by requiring that for any state q, if $\delta(q, \rho)$ is defined then there are at least two distinct states q_1 and q_2, each related to q via a ρ arc, and adding an alternate clause for the satisfiability of $\oplus\{\phi\}$, such that it is satisfiable by an automaton iff ϕ is satisfiable by that automaton.

An alternative is to maintain determinism by generalising the type of automata employed in the semantic domain. This parallels our alternate treatment of set valued features. Basically, the trick is to think of all values as composite values, which may be given a semantics in terms of sets of states. This is only possible because we have the axiom $\alpha = \oplus\{\alpha\}$. We can't do this for normal sets because with normal sets we need to distinguish between an element and the singleton containing that element.

We define an *alternately generalised deterministic finite state automaton* (AGDFA) as a tuple $\langle Q, q_0, L, \delta, A, \pi \rangle$, where

- Q is a set of atoms known as states,
- $q_0 \in Pow(Q)$ is a distinguished set of states state known as the start state set,
- L is a set of atoms known as labels,
- δ is a partial function from $Q \times L$ to $Pow(Q)$,
- A is a set of atoms (in the sense of Chapter 1), and
- π is a partial assignment of atoms to final states.

Diagrammatically, an AGDFA may be represented as in Figure 5.1, where sets of states are indicated by circles.

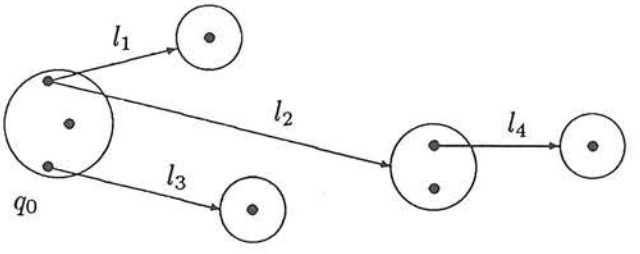


Figure 5.1: An Example AGDFA

Any DFSA $\mathcal{A} = \langle Q, q_0, L, \delta, A, \pi \rangle$ has a corresponding AGDFA $\mathcal{A}'$ where the state transition function maps states to singletons. $\mathcal{A}'$ is given by $\langle Q, \{q_0\}, L, \delta', A, \pi \rangle$ where $\delta'(q, l) = \{\delta(q, l)\}$.

Given an AGDFA $\mathcal{A}$ we define satisfiability of conjunctive, disjunctive and atomic formulae mostly as usual:

- $\mathcal{A} \models \top$ never,
- $\mathcal{A} \models \bot$ always,
- $\mathcal{A} \models a$ iff $q_0 = \{q\}$ and $\pi(q) = a$,
- $\mathcal{A} \models \phi \wedge \psi$ iff $\mathcal{A} \models \phi$ and $\mathcal{A} \models \psi$, and
- $\mathcal{A} \models \phi \vee \psi$ iff $\mathcal{A} \models \phi$ or $\mathcal{A} \models \psi$.

There is a significant difference in satisfiability for path equations:

- $\mathcal{A} \models l : \phi$ iff $\mathcal{A}/(l,q)$ is defined for each $q \in q_0$ and $\mathcal{A}/(l,q) \models \phi$, where if $\mathcal{A} = \langle Q, q_0, L, \delta, A, \pi \rangle$ and $q \in q_0$, then $\mathcal{A}/(l,q) = \langle Q, \delta(q,l), L, \delta, A, \pi \rangle$.

This clause has been altered to enforce the requirement that each element of q_0 must lead to a GDFSA that satisfies ϕ. The need for this is illustrated below.

The extensions for $\otimes$ and $\oplus$, for an AGDFA $\mathcal{A}$ $\langle Q, q_0, L, \delta, A, \pi \rangle$, are:

- $\mathcal{A} \models \otimes\Phi$ iff $\mathcal{A} \models \oplus\Psi$ for some subset Ψ of Φ.
- $\mathcal{A} \models \oplus\Phi$ iff there is a one-one mapping θ from Φ onto q_0 such that for each $\phi \in \Phi$, $\langle Q, \{\theta(\phi)\}, L, \delta, A, \pi \rangle \models \phi$.

Note that in the case of Φ a singleton, this last clause reduces to $\mathcal{A} \models \oplus\{\phi\}$ iff $\mathcal{A} \models \phi$.

The properties of composites may be summaried as follows:

- Disjunctive composite feature structures are a syntactic construction. Like disjunctive feature structures they exist in the language but have no direct correlation with objects in the world being modelled.
- Conjunctive composite feature structures describe composite objects which do exist in the world being modelled.
- A disjunctive composite feature structure describes an object just in case one of the disjuncts describes the object, or it describes a composite object.
- A disjunctive composite feature structure describes a composite object just in case each object in the composite is described by one of the disjuncts.
- A conjunctive composite feature structure describes an object just in case that object is a composite object and there is an isomorphism from the components of the feature structure to the components of the object such that each component feature structure describes the corresponding component object.

The crucial point here is that conjunctive composite objects exist in the described world whereas disjunctive composite objects do not.

As a brief example of the use of composites, consider the case where a noun phrase subcategorisation requirement is met by a coordinate structure consisting of two noun

phrases with different agreement features. Simplifying and departing from the CPSG formalism, the subcategorisation requirement may be specified as:

$$\left[\text{CAT}\ \ np\right]$$

or in the equational logic:

$$\text{(cat: np)}$$

A coordinate structure such as 'Tigger and the kittens' might be described as:

$$\left[\begin{array}{ll}\text{CAT} & np \\ \text{AGR} & sing\end{array}\right] \oplus \left[\begin{array}{ll}\text{CAT} & np \\ \text{AGR} & plural\end{array}\right]$$

In the equational logic this corresponds to:

$$\oplus\left\{\begin{array}{l}\text{(cat: np)} \wedge \text{(agr: sing)} \\ \text{(cat: np)} \wedge \text{(agr: plural)}\end{array}\right\}$$

Any AGDFA which satisfies the second of these equations must satisfy the first, because satisfiability of a formula of the form (cat: np) requires that each state in the initial state set has an arc labeled 'cat' leaving it and pointing to a terminal state with which the atomic value 'np' is associated. Consequently the first formula does subsume the second, as required.

This example illustrates why in the satisfiability clause for equations of the form $l : \phi$ each component state is required to lead via l to a feature structure which satisfies *phi*.

This formulation of AGDFAs does not allow for set valued attributes. To deal with such values we need to integrate this treatment with that of section 5.4 of Chapter 1, essentially allowing two different sorts of sets of states, sets representing composite values and sets representing normal set-valued attributes.

2 Subcategorisation and Binding

2.1 Subcategorisation

Only phrases whose subcategorisation requirements are consistent can be conjoined. 'Gives' and 'believes' cannot be conjoined because 'gives' requires an NP and a PP[to] complement whereas 'believes' requires an NP and a VP[+INF] complement. 'Gives a

bone to Tigger' and 'believes Tom to be watching', on the other hand can be conjoined, because each requires no complements and an NP[nom] specifier. The rule licensing coordinate phrases must thus specify that the SPEC and COMPS attributes of each of the conjuncts must not conflict. This may be ensured by requiring that the values of those attributes be shared by each of the conjuncts, and by the coordinate phrase as a whole. This structure sharing also effects the necessary semantic binding which we discuss in Chapter 7, ensuring that, for example, in a constituent such as 'Tigger is ferocious and bouncy', Tigger fills the argument roles associated with both 'ferocious' and 'bouncy'.

To illustrate, consider the partial descriptions for 'gives', 'believes', 'gives a bone to Tigger', 'believes Tom to be watching' and 'gives a bone to Tigger and believes Tom to be watching':

$$\begin{bmatrix} \text{PHON} & \langle \textit{gives} \rangle \\ \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{SPEC} & \text{NP} \\ \text{COMPS} & \{\text{NP,PP[to]}\} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \text{PHON} & \langle \textit{believes} \rangle \\ \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{SPEC} & \text{NP} \\ \text{COMPS} & \{\text{NP, VP[+INF]}\} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \text{PHON} & \langle \textit{gives a bone to Tigger} \rangle \\ \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{SPEC} & \text{NP} \\ \text{COMPS} & \textit{null} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \text{PHON} & \langle \textit{believes Tom to be watching} \rangle \\ \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{SPEC} & \text{NP} \\ \text{COMPS} & \textit{null} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \text{PHON} & \langle \textit{gives a bone to Tigger and believes Tom to be watching} \rangle \\ \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{SPEC} & \text{NP} \\ \text{COMPS} & \textit{null} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The values of the COMPS attributes of the partial descriptions for 'gives' and 'believes' conflict, corresponding to the fact that "gives and believes" is not a constituent. The values of the SPEC and COMPS attributes of 'gives Tigger a bone' and 'believes Tom to be watching', on the other hand, do not conflict. Thus as far as this subcategorisation requirement is concerned, 'gives Tigger a bone and believes Tom to be watching' is a potential constituent.

2.2 Binding Features

As in some phrase structure grammars (cf. [Gazdar 81], [Gazdar *et al.* 85]) and some categorial grammars (cf. [Steedman 85], [Morrill 88]), our treatment of unbounded dependencies allows a natural treatment of the interaction of gapping and coordination. We simply require that all BINDING attributes are shared by each of the conjuncts and by the coordinate phrase as a whole. Thus phrases such as 'believes _ to be ungrateful but gives _ a bone' are licensed:

$$\left[\begin{array}{ll} \text{PHON} & \langle \textit{believes to be ungrateful} \rangle \\ \text{SYN} & \left[\begin{array}{ll} \text{LOC} & \left[\begin{array}{ll} \text{SPEC} & \text{NP} \\ \text{COMPS} & \textit{null} \end{array}\right] \\ \text{BIND} & \left[\text{SLASH} \left[\begin{array}{ll} \text{BOUND} & \_ \\ \text{ALL} & \{\text{NP[acc]}\} \end{array}\right]\right] \end{array}\right] \end{array}\right]$$

$$\left[\begin{array}{ll} \text{PHON} & \langle \textit{gives a bone} \rangle \\ \text{SYN} & \left[\begin{array}{ll} \text{LOC} & \left[\begin{array}{ll} \text{SPEC} & \text{NP} \\ \text{COMPS} & \textit{null} \end{array}\right] \\ \text{BIND} & \left[\text{SLASH} \left[\begin{array}{ll} \text{BOUND} & \_ \\ \text{ALL} & \{\text{NP[acc]}\} \end{array}\right]\right] \end{array}\right] \end{array}\right]$$

$$\left[\begin{array}{ll} \text{PHON} & \langle \textit{believes to be ungrateful but gives a bone} \rangle \\ \text{SYN} & \left[\begin{array}{ll} \text{LOC} & \left[\begin{array}{ll} \text{SPEC} & \text{NP} \\ \text{COMPS} & \textit{null} \end{array}\right] \\ \text{BIND} & \left[\text{SLASH} \left[\begin{array}{ll} \text{BOUND} & \_ \\ \text{ALL} & \{\text{NP[acc]}\} \end{array}\right]\right] \end{array}\right] \end{array}\right]$$

At the same time, phrases such as '*pictures of whom Tom throws darts at and whom does Tigger like', where a relative clause is conjoined with a question, are ruled out as their REL attributes and QUE attributes conflict.

3 Head/Conjunction Phrases

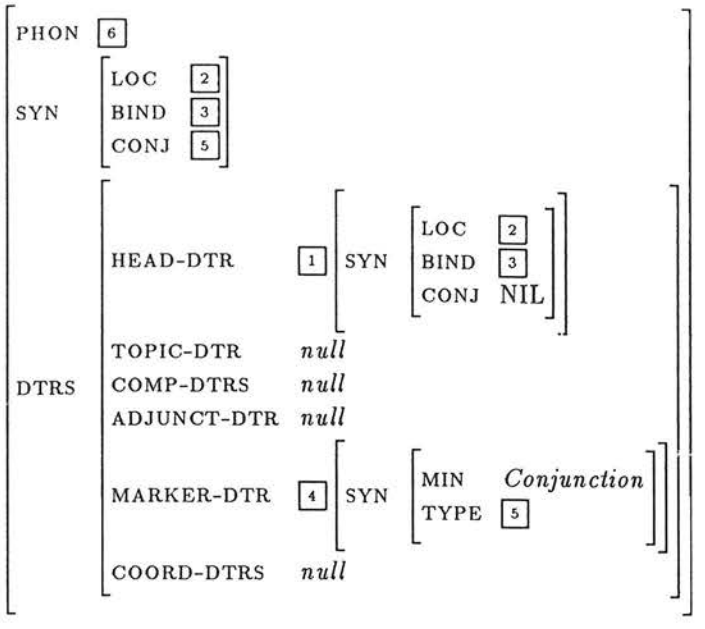
Head/conjunction phrases are those constituents consisting of a constituent (the head) preceded by a conjunction, i.e., phrases such as 'and Tigger', 'neither the stripped kitten', 'but bounced happily', etc.. To admit such phrases we introduce the SYNTAX|CONJ attribute, which takes atomic values from (at least) the set {*and*, *both*, *but*, *neither*, *nor*, *or*, NIL}. The value of this attribute is NIL for all major lexical signs and for all headed phrases apart from those of type head/conjunction phrase. The attribute is inappropriate for minor lexical signs.

Essentially, the head/conjunction rule below licences constituents of the GPSG form X[CONJ α], where $\alpha \neq$ NIL. No rule is necessary to license constituents of the GPSG form X[CONJ NIL]: all constituents which are not of type head/conjunction phrase are of this form (except minor lexical constituents, which cannot be coordinated). This is ensured by requiring that the descriptions of the types head/adjunct phrase, head/argument phrase, and major lexical constituent are all extensions of [SYN|CONJ NIL] .

The conjunctions 'both' and 'and' are described by:

$$\begin{bmatrix} \text{PHON} & \langle \textit{both} \rangle \\ \text{SYN} & \begin{bmatrix} \text{MIN} & \textit{Conjunction} \\ \text{TYPE} & \textit{both} \end{bmatrix} \end{bmatrix} \qquad \begin{bmatrix} \text{PHON} & \langle \textit{and} \rangle \\ \text{SYN} & \begin{bmatrix} \text{MIN} & \textit{Conjunction} \\ \text{TYPE} & \textit{and} \end{bmatrix} \end{bmatrix}$$

All constituents of type head/conjunction phrase are partially described by:



where *order-phonology*(6,{1,4})

and *constituent*(1)

constituent(4)

Head/conjunction phrases may be partitioned into subtypes according to the conjunction which marks the phrase. These subtypes include head/conjunction[and] phrase, head/conjunction[or] phrase, head/conjunction[both] phrase, head/conjunction[but] phrase, head/conjunction[either] phrase, head/conjunction[neither] phrase, etc..

4 Subtypes of Coordinate Phrase

Following [Gazdar *et al.* 85], we treat the phrase structure of coordinate structures in terms of binary and iterated coordination. Examples of binary coordination are given in (6) and examples of iterated coordination are given in (7).

(6) a. both α and β

b. either α or β

c. α but β

(7) a. α and β and γ and ...

b. $\alpha, \beta, \ldots, \gamma$ and δ

c. α or β or γ or ...

d. $\alpha, \beta, \ldots, \gamma$ or δ

e. neither α nor β nor γ nor ...

Note that iterated coordinate phrases may contain exactly two conjuncts: it is not "having exactly two conjuncts" which distinguishes binary and iterated coordinate phrases, it is the pattern of conjunctions which mark the conjuncts.

4.1 Binary Coordination

Constituents of type **binary coordinate phrase** are partially described by:

$$\boxed{1} = \begin{bmatrix} \text{PHON} & \boxed{2} \\ \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \textit{conjoin-head}(\boxed{3}) \end{bmatrix} \\ \text{CONJ} & \text{NIL} \end{bmatrix} \\ \text{DTRS} & \begin{bmatrix} \text{HEAD-DTR} & \textit{null} \\ \text{TOPIC-DTR} & \textit{null} \\ \text{COMP-DTRS} & \textit{null} \\ \text{ADJUNCT-DTR} & \textit{null} \\ \text{MARKER-DTR} & \textit{null} \\ \text{COORD-DTRS} & \boxed{3} \end{bmatrix} \end{bmatrix}$$

where *order-phonology*($\boxed{2}$,$\boxed{1}$)

and *share-comps*($\{\boxed{1}\} \uplus \boxed{3}$)

and *share-spec*($\{\boxed{1}\} \uplus \boxed{3}$)

and *share-bind*($\{\boxed{1}\} \uplus \boxed{3}$)

and *set-of-constituents*($\boxed{1}$)

The relations *share-spec*, *share-comps*, and *share-bind* take sets of feature structures, all of whose elements are defined on the SYN|LOC|SPEC, SYN|LOC|COMPS, and SYN|BIND attributes respectively, as their arguments. The relations hold if each element of the set shares the value of the relevant attribute. The function *conjoin-head* also takes a set

of feature structures as its argument. It returns the conjunctive composite of the head attributes of each feature structure in the argument set.

There are (at least) three subtypes of binary coordinate phrase: both-and, either-or, and NIL-but. Constituents of type both-and are partially described by:

$$\left[\text{DTRS|CONJ-DTRS} \left\{\left[\text{SYN|CONJ } \textit{both}\right], \left[\text{SYN|CONJ } \textit{and}\right]\right\}\right]$$

Similarly constituents of type either-or are partially described by:

$$\left[\text{DTRS|CONJ-DTRS} \left\{\left[\text{SYN|CONJ } \textit{either}\right], \left[\text{SYN|CONJ } \textit{or}\right]\right\}\right]$$

Finally, constituents of type NIL-but are partially described by:

$$\left[\text{DTRS|CONJ-DTRS} \left\{\left[\text{SYN|CONJ NIL}\right], \left[\text{SYN|CONJ } \textit{but}\right]\right\}\right]$$

The following English linear precedence rules are relevant for binary coordination:

$$\left[\text{SYN|CONJ } \textit{both}\right] \prec \left[\text{SYN|CONJ } \textit{and}\right]$$

$$\left[\text{SYN|CONJ } \textit{either}\right] \prec \left[\text{SYN|CONJ } \textit{or}\right]$$

$$\left[\text{SYN|CONJ NIL}\right] \prec \left[\text{SYN|CONJ } \textit{but}\right]$$

The relevant cases for the relation *order-phonology* may thus be specified:

$$\textit{order-phonology}\left(\boxed{1}\,\hat{}\,\boxed{2}, \left\{\begin{bmatrix}\text{PHON} & \boxed{1}\\ \text{SYN|CONJ} & \textit{both}\end{bmatrix}, \begin{bmatrix}\text{PHON} & \boxed{2}\\ \text{SYN|CONJ} & \textit{and}\end{bmatrix}\right\}\right)$$

$$\textit{order-phonology}\left(\boxed{1}\,\hat{}\,\boxed{2}, \left\{\begin{bmatrix}\text{PHON} & \boxed{1}\\ \text{SYN|CONJ} & \textit{either}\end{bmatrix}, \begin{bmatrix}\text{PHON} & \boxed{2}\\ \text{SYN|CONJ} & \textit{or}\end{bmatrix}\right\}\right)$$

$$\textit{order-phonology}\left(\boxed{1}\,\hat{}\,\boxed{2}, \left\{\begin{bmatrix}\text{PHON} & \boxed{1}\\ \text{SYN|CONJ} & \text{NIL}\end{bmatrix}, \begin{bmatrix}\text{PHON} & \boxed{2}\\ \text{SYN|CONJ} & \textit{but}\end{bmatrix}\right\}\right)$$

Note that this specification is independent of that required by *order-phonology* when ordering complements in constituents of type head/complement phrase. All complements necessarily bear the specification [SYN|CONJ NIL], and so are not ordered by the above LP rules and their interpretation by *order-phonology*.

4.2 Iterated Coordination

Constituents of type iterated coordinate phrase are partially described by:

$$\boxed{1} = \left[\begin{array}{ll} \textsc{phon} & \boxed{2} \\ \textsc{syn} & \left[\begin{array}{ll} \textsc{loc} & \left[\textsc{head} \quad \textit{conjoin-head}(\boxed{3})\right] \\ \textsc{conj} & \textsc{nil} \end{array}\right] \\ \textsc{dtrs} & \left[\begin{array}{ll} \textsc{head-dtr} & \textit{null} \\ \textsc{topic-dtr} & \textit{null} \\ \textsc{comp-dtrs} & \textit{null} \\ \textsc{adjunct-dtr} & \textit{null} \\ \textsc{marker-dtr} & \textit{null} \\ \textsc{coord-dtrs} & \boxed{3} \end{array}\right] \end{array}\right]$$

where *order-phonology*($\boxed{2}$,$\boxed{1}$)

and *share-comps*($\{\boxed{1}\} \uplus \boxed{3}$)

and *share-spec*($\{\boxed{1}\} \uplus \boxed{3}$)

and *share-bind*($\{\boxed{1}\} \uplus \boxed{3}$)

and *set-of-constituents*($\boxed{1}$)

In iterated coordination, one conjunct is distinguished: it is marked with a different conjunction from all the other conjuncts. There are (at least) five subtypes of iterated coordinate phrase, corresponding to the different conjunctions used to mark the conjuncts: and-NIL, NIL-and, or-NIL, NIL-or, and neither-nor. For each of these types there is a corresponding unary relation *iterated-conj-type*$_{x,y}$ for $(x, y) \in$ {(and,NIL), (NIL,and), (or,NIL), (NIL,or), (neither,nor)} such that

$$\left[\textsc{dtrs} \left[\textsc{coord-dtrs} \quad \boxed{1}\right]\right]$$

where *iterated-conj-type*$_{x,y}$($\boxed{1}$)

iterated-conj-type$_{x,y}$*/1* holds of a multi-set of feature structures iff that set contains at least two members and there is precisely one member of that set specified as [SYN|CONJ x] and all other members of the set are specified as [SYN|CONJ y].

The following linear precedence rules are relevant for English iterated coordination:

$$\left[\textsc{syn|conj} \quad \textsc{nil}\right] \prec \left[\textsc{syn|conj} \quad \textit{and}\right]$$

$$\left[\textsc{syn|conj} \quad \textsc{nil}\right] \prec \left[\textsc{syn|conj} \quad \textit{or}\right]$$

$$\left[\text{SYN|CONJ } \textit{neither}\right] \prec \left[\text{SYN|CONJ } \textit{nor}\right]$$

Again here the relational nature of *order-phonology* is important. The linear precedence rules only provide a partial ordering on the constituents, ensuring that the distinguished conjunct appears first in the case of constituents of type NIL-and, NIL-or, and neither-nor, and last in the case of constituents of type and-NIL and or-NIL. *Order-phonology* cannot, and does not, distinguish between the other conjuncts.

5 Summary

In this chapter we have presented the CPSG treatment of coordination. The main innovation is the use of the $\oplus$ operator to build composite feature structures, which are required to describe the head features of coordinate phrase. It is important to realise that this operator represents a real extension of the logic, increasing the expressibility, and cannot be replaced by some functional dependency.

Three rules are relevant to the phrase structure of coordinate structures. One rule licenses phrases consisting of a head preceded by a conjunction, which marks the phrase as a particular subtype of head/conjunction phrase. The other two rules license binary coordinate phrases and iterated coordinate phrases, following the GPSG approach to coordination.

Chapter 6

Background Semantic Theory

In this chapter, we present the semantic theory underlying the CPSG treatment of semantics. As in HPSG, this is a version of situation semantics. However, there are numerous points on which the two versions differ. Furthermore, there are numerous slightly different versions of situation semantics current in the literature. This chapter is thus necessary for a precise grounding of the semantic notions of the following two chapters: such a grounding cannot be assumed.

We begin by outlining some of the more important philosophical standpoints of situation semantics as they relate to the semantics of natural language. The bulk of the chapter then concerns the theory used to capture these standpoints. Finally we make many of the notions precise by giving details of how they may be mathematically modelled.

1 Situation Semantics

Situation semantics differs from most other semantic treatments of natural language (such as possible world semantics) on a number of important philosophical points. In this section some of these differences which are relevant to the CPSG treatment of semantics are outlined. It is worth noting that everything which is said in this section applies equally well to HPSG.

1.1 Meaning

Unlike most other approaches to meaning, the situation semantic approach is not just limited to natural language semantics. Its scope is much wider — linguistic meaning

is seen as just one instance of meaning in general. Meaning in situation semantics is analysed as a relation between different (types of) situations. (For present purposes a situation can be thought of as a partial possible world.) If whenever there is a situation of type S_1, there is a corresponding situation of type S_2, and this correspondence is not accidental (i.e., there is some law-like relationship behind the correlation), then S_1 is said to *mean* S_2. That is, the meaning relation holds of the situation types S_1 and S_2. Smoke *means* fire because whenever there is a smoky situation there is a corresponding fiery situation, and this correspondence arises from the causal relationship between fire and smoke (i.e., there is a causal relationship behind the correlation), and causal relationships are law-like. To know that smoke means fire is to be attuned to this systematic relationship between smoky situations and fiery situations. [Barwise 84] refers to the law-like relationship between smoke and fire as "situation type meaning" — it is a meaning relation that relates types of situations. This may be contrasted with the meaning of a particular instance of a smoky situation. The meaning of such an instance will be a particular instance of a fiery situation. Barwise refers to this, the particular fiery situation, as the "situation meaning" of the particular smoky situation. Alternately, the smoky situation carries with respect to the law-like dependency the information that there is fire.

This view of meaning may be contrasted with that of the set theoretic approach to possible worlds, as exemplified by [Montague 70, 73], where the meaning of an utterance is identified with the set of possible worlds in which the utterance is true, and with that of dynamic semantics, as exemplified by [Barwise 87a] and [Groenendijk & Stokhof 90], where the interpretation of an utterance is viewed in terms of its action on the world and is consequently taken as a function from worlds to worlds. In contrast to the first of these, meaning in situation semantics is independent of truth — it is not necessary to know when something is true to know what it means. Furthermore, things with necessarily equivalent truth values, such as tautologies or contradictions, need not have the same meaning. The view of meaning in situation semantics bears a much closer resemblance to that in dynamic semantics. One major difference is the role which context plays in situation semantics. This leads to a treatment of meaning in situation semantics as relational, rather than functional.

Linguistic meaning is a particular instance of the meaning relation which relates situation types in which an utterance occurs to situation types described by such utterances. An utterance which occurs in one situation describes another situation. Linguistic meaning is thus said to relate *utterance* situation types and *described* situation types. Truth is secondary to content, and only emerges when we ask how the described situation compares with the real world.

In the case of linguistic meaning, the distinction between situation type meaning and situation meaning, which is not paralleled in possible world approaches, allows a distinction to be drawn between a statement or sentence (the situation type meaning) and a particular utterance of that statement (the situation meaning). This distinction allows context, that which differs between particular utterances of the same sentence, to figure clearly and cleanly in the determination of linguistic meaning.

1.2 Constraints

Smoke means fire is one instance of the meaning relation. *A blue light flashing behind the car you are driving means to pull over* is another. A third is *"I", when spoken by me, means myself*. Meaning is thus a complex relation which arises from a number of separate, often independent, sources, or *constraints*, which partially specify the meaning relation. Linguistic constraints form only a part of this relation.

A (positive) constraint is a special sort of relation between two situation types which holds if whenever there is a situation of the first type there exists a situation of the second type. It is by being attuned to constraints that agents are able to pick up information about one situation from another. By being attuned to the constraint *smoke means fire*, an agent can pick up information about a fiery situation from a smoky situation. In this respect, situations carry information with respect to constraints. The same situation can carry different information with respect to different constraints. There are also negative constraints, which are again relations between situation types. Such constraints are responsible for the fact that certain situation types do not co-occur.

Constraints may be necessary, as in *Tom's being a cat means Tom is a mammal*, or conventional, as in *"I", when spoken by me, means myself*. A language specifies a number

of conventional constraints which determine part of the meaning relation. Conventional constraints are, as the name suggests, only constraints of convention, and can be broken. For successful communication, the hearer must be attuned to the constraints of a language and the speaker must adhere to those conventional constraints.

1.3 The Importance of Context

[Barwise & Perry 83] argue that the contribution of context in determining meaning is much greater than is usually realised, and that this contribution is reflected in the *efficiency of language* — that one sentence can be used in a variety of contexts and have different interpretations. Few, if any, sentences, they argue, are entirely independent of context for their interpretation. However, as [Barwise 87b] argues, in most possible world approaches to natural language semantics, context is treated in a somewhat *ad hoc* manner through the use of indices. When more contextual factors are discovered that influence the interpretation of an utterance, more indices are added to the interpretive apparatus of the logic. Thus, for any utterance, we end up with indices for the current world, time, speaker, hearer, etc.. In situation semantics, that an agent is "situated", i.e., exists in a context, is of central importance. This is reflected in Barwise's distinction between situation type meaning and situation meaning ([Barwise 84]). It is also reflected in situation theory, where many notions, including that of truth itself, have "situation relative" analogues.

The notion of situatedness is also important with respect to constraints. Many, if not all constraints, are not universally applicable. Their actual applicability depends on context, or background conditions. Such constraints are termed *conditional constraints*. If an agent never makes use of a conditional constraint outside of situations in which the background conditions hold, then this dependence on background conditions is not important to the agent, and it is unlikely that the agent would even be aware of the dependence. On the other hand, if an agent is placed in an environment where some background condition fails to hold, and the agent is unaware of the importance of this condition, then the agent may incorrectly employ the constraint and pick up misinformation from its environment (i.e., the situation in which it exists). As an illustration of this, consider how a bird from the wilderness which is attuned to the (conditional)

constraint that *being able to see clearly in direction x means that it is safe to fly in direction x* reacts when placed in an environment with transparent objects (such as glass windows). In such an environment the constraint is not applicable because one of the background conditions — that there be no transparent objects — is violated. Consequently the bird may incorrectly infer that it is safe to fly into a glass window.

Although a conditional constraint may be converted to a universal constraint by conjoining the background conditions with the antecedent (i.e., effectively making the background conditions explicit), this would be missing the idea of what it is for an agent to be situated. The efficiency of language arises from the fact that background conditions are implicit.

Context is obviously vitally important to the interpretation of utterances. Situation semantics divides context into two parts, the *resource situation* and the *speaker connections*. The resource situation contains background information which is (in general) common to both the speaker and hearer, and which the speaker may draw upon, often to establish reference. This information might include the gender or names of individuals, or some relationships between individuals. In a sentence such as 'Tigger chased the kitten', for example, resource situations will provide the context to resolve both the use of 'Tigger' and the use of 'the kitten'. We allow different referring expression in an utterance to draw on different resource situations, so that many different resource situations may be implicated in any one utterance. To illustrate why a single resource situation is insufficient, consider 'The kitten chases the other kitten'. Given that we take 'the' to indicate uniqueness, a single resource situation cannot resolve both referring expressions, as that resource situation must contain at least two kittens. A proper analysis of this phenomenon though should account for the relationship between the resource situations stemming from the use of 'other'.

In general we work in terms of parametric situation types. The described situation type, for example, will be parametric: it will contain parameters, or indeterminates, corresponding to, for example, the referents of pronouns or proper names. These parameters may be mapped, or anchored, to individuals in the particular context of the utterance. Speaker connections provide such a mapping. For example, speaker connec-

tions will map the parameter introduced via a pronoun to the referent of that pronoun. The resource situation will contain information about, for example, the gender of that referent. A conventional constraint will require that this gender be the same as that required by the pronoun used.

We thus have the general meaning "equation": situation type S_1 (in the context of a situation of type B) means situation type S_2. In the case of a linguistic meaning relation this can be phrased as "the utterance situation being of type S_1 (in the context of a resource situation of type B) means that the described situation is of type S_2". Speaker connections ground any parameters in these situation types to real objects in the domain of discourse.

2 Situation Theory

Before we can begin to give an account of the semantics of natural language, we must discuss some ontological issues. The ontology of situation theory, the theoretical underpinning of situation semantics, includes many different sorts of objects — the theory has been referred to as ontologically promiscuous. In this section we discuss some of the notions and objects of the theory.

2.1 Schemes of Individuation

Situation theory assumes that agents individuate various objects, including *situations*, *individuals*, *relations* and *infons*. These objects are taken to be real objects in the world (situation theory is thus ontologically realist), and the individuation is governed by a *scheme of individuation*. This is a theoretical notion related to the agent's perception of the world. A scheme of individuation specifies which of the many existing situations, individuals, relations, infons, etc. an agent individuates. Thus a general theory of situations may be explored, where the objects (individuals, relations, infons, etc.) are taken as abstract entities, without being specific about which objects are individuated. This general theory can then be grounded in a particular instance by specifying a scheme of individuation. (See the appendix of [Barwise 88a] for some notes on the modelling of schemes of individuation.)

This story is different to that told in [Pollard & Sag 87], where a scheme of individuation is placed at a higher level. They take a scheme of individuation to be "a way of classifying things in the world which is characteristic of a certain community" ([Pollard & Sag 87, p. 82]). Thus, according to Pollard and Sag, a scheme of individuation is prior to the notions of individual, relation, infon, etc., and the classification that we adopt of objects into individuals, relations, infons, etc., is really only one instance of a more general theory, a theory which does not involve individuals, relations, infons, etc.. We do not adhere to the this story. For the purposes of a general linguistic theory, such as CPSG (and surely HPSG) it is important that different communities employ the same sorts of semantic objects (individuals, relations, infons, etc.). Thus whilst different agents may individuate different individuals, relations, infons and situations, we take all to agents individuate these sorts of objects.

A simplifying assumption is that all agents within a linguistic community are governed by the same scheme of individuation. In reality this is probably only approximately true, as it is unlikely that, for example, all speakers of English individuate precisely the same relations. However, for successful communication, the schemes of individuation employed by different members of one linguistic community must be relatively similar, as otherwise people wouldn't be able to understand each other. For this reason, we shall adhere to the general assumption, although some interesting consequences may result if schemes of individuation are considered to be dependent on agents and perhaps also times.

2.2 Situations

Situations have been likened to partial possible worlds. The motivation for such partial objects comes from the fact that an agent is seldom (if ever) aware of everything in the world. In a given possible world, it is either true or false that Tom is miaowing, but whilst some situations may settle the issue of Tom miaowing (i.e., in some situations Tom will be miaowing, whilst in others he will not be), there are other situations which have nothing to say about this issue. This partiality of situations also makes them computationally more tractable than possible worlds. By characterising a situation by the things going on in it (in a way discussed below), a situation can be seen as (the

closure of) a small finite object.

Situations are partially ordered by a *part of* relation, $\trianglelefteq$: one situation may be part of another, larger, situation. We postpone discussion of this relation until section 2.5 where we discuss how situations may be characterised.

2.3 Individuals

Individuals are roughly "those persistent things that belong to the causal order of the world, the objects that we can track perceptually and affect by acting upon them" ([Pollard & Sag 87, p. 83]). This includes things such as yourself, or your cat. The domain of individuals may come with some primitive structure arising from grouping operations (such as those suggested by collectives) and "part of" relations which might hold between individuals. (A finger is part of a hand, which in turn is part of a limb, and so on.) Such structure is not required by situation theory and will largely be ignored.

2.4 Relations and Properties

Situation theoretic objects may have properties or participate in relations. For example, the *believes* relation might hold between a particular person and a particular proposition — a person might believe a proposition — or a particular object may have the property of *miaowing*. As part of the ontologically realist stance of situation semantics, relations and properties are taken to be real primitives in the world, rather than sets of ordered tuples (as in, for example, classical possible world semantics). This allows a great degree of intensionality — two relations may have the same extension (i.e., they may hold of the same tuples) without being the same relation.

Relations come with a set of argument roles and appropriateness conditions on those roles. Thus, for example, the relation *believe* has (at least) two argument roles, a believer and a belief. Only certain objects are appropriate for the first of these roles (probably all individuals, but certainly not situations), whilst only certain other things (probably only propositions, and certainly not individuals) are appropriate for the second. Thus it does not make sense for the fact that Tom is a cat to believe Sam: the object corresponding to "the fact that Tom is a cat" is inappropriate for the believer role and the object

corresponding to "Sam" is inappropriate for the belief role of the *believe* relation. A precise characterisation of appropriateness conditions is not attempted here.

There is some dispute over whether relations should be treated as having a fixed arity. Arguments are often optional, as in the case of the *bite* relation. In 'Tigger bites Fido', *bite* is a two-place relation, but 'bite' may also correspond to a one-place relation: Tigger can bite without actually biting anything. One approach to this problem is to allow two *bite* relations, $bite''$ and $bite'$, which just happen to correspond to the same English word, one binary and one unary. However, if Tigger bites Fido then Tigger does bite, so there is an inference from the binary case to the unary case. This can be captured via a meaning postulate, informally of the form:

$$bite''(x,y) \rightarrow bite'(x)$$

Clearly there is a large class of such relations. [Barwise 88a] suggests that such relations might be accommodated via a mechanism which licenses the "projection" of argument roles, so that the argument roles of a relation should be relative to the situation supporting the infon of which the relation is a part. We do not investigate this possibility here.

For HPSG, [Pollard & Sag 87] take each relation as having specific argument roles: *walk* has a walker role, which is different from the runner role of *run*. Alternately, we might have a fixed set of argument roles, such as {agent, patient, theme, experiencer, goal, ...}, from which each relation draws its argument roles. A third approach is to take such thematic roles as generalisations across argument roles. In the feature structure descriptions we employ in Chapter 7, thematic roles label the argument roles of relations. This does not carry any commitment to any particular approach. All that is really of importance is that argument roles are not ordered: associated with each relation is a set, not a list, of argument roles (and appropriateness conditions on those argument roles, though note that the same argument roles may have different appropriateness conditions when appearing with different relations).

The last point that needs to be made about relations is that many (perhaps all) relations take a spatio-temporal location as an argument. For example, tense markers on verbs introduce space-time locations at which the corresponding relation holds. Noun-like

properties, such as "kittenhood", may also be seen as only holding at certain spatio-temporal locations.

2.5 Basic Infons

By assigning appropriate objects to the argument roles of a relation, an issue — the issue of whether or not the objects stand in the relation — is formed. Such an issue could be resolved positively or negatively. Correspondingly, there are two *infons* (or *circumstances* or *states of affairs* or *possible facts*) that may be formed from every issue. Tom miaowing at a certain space-time corresponds to one infon, and Tom not miaowing at the same space-time corresponds to another. Both of these infons concern the issue of whether Tom miaows at that space-time. An infon is thus made up of an issue (which in itself consists of a relation and an appropriate assignment of objects to the argument roles of that relation), and a *polarity*, which may be positive or negative. The infon of Tom miaowing at l_d is denoted:

$$\langle\langle \text{miaow, miaower: Tom, location: } l_d; + \rangle\rangle$$

Similarly, the infon of Tom not miaowing at l_d is denoted:

$$\langle\langle \text{miaow, miaower: Tom, location: } l_d; - \rangle\rangle$$

Here we have taken the relation *miaows* to have two arguments roles: *miaower* and *location*.

For each (basic) infon σ, the dual of σ, written $\overline{\sigma}$ is the infon formed from the same issue as σ but resolved with the opposite polarity. The above two infons are duals. Polarities are a kind of formal annotation. They are not situation theoretic objects like individuals and relations.

For the most part we shall be concerned with cases where the assignment of objects to the argument roles of a relation is total, although there is room for partial assignments. An issue or infon involving only a partial assignment is termed *unsaturated*. Such an issue is that of Tom eating at l_d, expressed as:

$$\langle\langle \text{eat, eater: Tom, location: } l_d; + \rangle\rangle$$

This is unsaturated as the *eat* relation has an argument role corresponding to the thing eaten, and in this example no object is assigned to that argument role, although if Tom is eating, he must be eating something. Note that unsaturated infons are distinct from infons formed from a relation of less than maximal arity: in the "bite" example, 'Tigger bites' does not imply that there is anything that is bitten by Tigger.

Infons are not in themselves true or false, but are *made* true by a situation. Whether the positive resolution of the issue of Tom miaowing at a certain space-time location is true, for example, depends on the situation in question: in one situation it may be the case that Tom is miaowing at the relevant space-time location, whereas in another it may not be the case that Tom is miaowing at that space-time (and this is distinct from it being the case that Tom is not miaowing). If a situation S makes an infon σ true, then the infon is said to *hold* in the situation, or the situation is said to *support* the infon, and we write

$$S \models \sigma$$

Turning this around, a situation may be characterised extensionally in terms of the infons which it supports.

From the perspective of a situation making an infon true, the polarity is necessary because of the partial nature of situations. Because a situation does not resolve every issue, there is a difference between a situation making a negative infon true and the situation not supporting the corresponding positive infon. The first is written, for example:

$$S \models \langle\langle \mathrm{R}, \mathrm{a}, \mathrm{b}; - \rangle\rangle$$

whilst the second is written:

$$S \not\models \langle\langle \mathrm{R}, \mathrm{a}, \mathrm{b}; + \rangle\rangle$$

Situations are coherent: they cannot support both an infon and its dual. Consequently we may infer from $S \models \sigma$ that $S \not\models \overline{\sigma}$. However, because situations are partial, we cannot make the reverse inference: we cannot infer from $S \not\models \sigma$ that $S \models \overline{\sigma}$.

Situations may be partially ordered by the *part-of* relation $\trianglelefteq$. In contrast to some theorists (cf. [Barwise 88b]), who take *part-of* to be a primitive relation between situations, we take it to be a purely extensional relation that holds between two situations iff the class of infons supported by the first is a subclass of the class of infons supported by the second. Tied up in this notion of *part-of* is the notion of *persistence*: infons which hold in one situation hold in all situations of which that situation is a part — they are *upwardly persistent*. We argue for our extensional definition of *part-of* in section 1.3 of Chapter 8, where we treat it as a structurally determined type. See also [Richard Cooper 90a].

2.6 Parameters

Parameters in situation theory are usually thought of as theoretical constructs used to link argument roles to real objects. In some sense they do the work of unbound variables in the predicate calculus. They are mapped, or *anchored*, to real objects in the situation theoretic domain in much the same way as an assignment function maps unbound variables in the predicate calculus to objects of that domain. We write parameters with a dot above them to distinguish them from other situation theoretic objects.

Many situation theoretic objects have parametric analogues. A parametric infon is an infon with a parametric relation (formed by abstraction, see section 2.7) or one or more parametric objects as arguments, where the simplest kind of parametric object is just a parameter itself. A situation can only support non-parametric infons, but from a parametric infon and a situation a parametric proposition, the proposition that the situation supports the parametric infon can be formed. Parametric propositions, like parametric infons, are just propositions where the type is parametric or one argument role is filled by a parametric object. Given a parametric object τ and an anchor a for the parameters in τ, the object that results from anchoring the parameters in τ with a is written $\tau[a]$. Thus we have as an example of a parametric infon:

$$\langle\!\langle \text{naming, name: "Kim", named: } \dot{x}; + \rangle\!\rangle$$

and the (non-parametric) infon that results from anchoring it with the anchoring function a:

$$\langle\langle \text{naming, name: "Kim", named: } \dot{x}; + \rangle\rangle[a]$$

Many versions of situation theory, including that of [Pollard & Sag 87], allow *restricted parameters*, parameters that are allowed only to be anchored to a restricted domain of objects. Quoting the example of [Pollard & Sag 87, p. 94], the restricted parameter

$$\dot{x}_{\langle\langle \text{naming, name: "Kim", named: } \dot{x}; + \rangle\rangle}$$

can only be anchored to an object named Kim. This is achieved by requiring that any anchor a for $\dot{x}$ must map $\dot{x}$ to an object named "Kim" for the anchor to be well-defined. There is some imprecision in the way this is often stated. What we need is that for the anchor a to be well-defined some situation must support

$$\langle\langle \text{naming, name: "Kim", named: } \dot{x}; + \rangle\rangle[a]$$

To be precise we need to say which situation is required to support this infon. There are several options. Either we may require that every situation which supports a fact obtained by anchoring the parameter must also support the restriction, or we may postulate a maximal situation (sometimes called "the world") that must support the restriction (it seems that this is what is required by HPSG), or we may require a specified resource situation to support the restriction. To avoid confusion we choose instead to make the relevant situation explicit by writing

$$\dot{x}_{S \models \langle\langle \text{naming, name: "Kim", named: } \dot{x}; + \rangle\rangle}$$

for some particular situation S.

By assigning one parameter to several argument roles of a relation, the argument roles may be linked, so that they must each be anchored to the same object. Thus we have the parametric infon of $\dot{x}$ *loving itself*:

$$\langle\langle \text{loving, lovee: } \dot{x}, \text{ lover: } \dot{x}; + \rangle\rangle$$

The status of parameters in situation theory is open to question. They do not correspond directly to objects in the real world, and must be distinguished from the other "real" objects (such as situations, individuals and relations) of the theory. This has led to an

alternative conception of parameters known as *structural parameters* ([Robin Cooper 89]).

Within the version of situation theory employed by [Pollard & Sag 87], there are several different sorts of parameters. These sorts mirror the syntactic agreement differences in noun phrases which introduce parameters. Parameters are treated (in English) as having PERSON, NUMBER and GENDER features. Linking a treatment of parameters to agreement features is clearly language specific, so that this area might be seen as one where language specific schemes of individuation are involved. CPSG likewise attaches features to parameters. These should be seen as restrictions: a parameter bearing the features *3rd, sing, fem* is one which must be anchored to an object which also bears the features *3rd, sing, fem*. That is, the attachment of features to parameters in CPSG is a short hand for the use of restricted parameters. Such restrictions are all stated with respect to relevant resource situations.

2.7 Abstraction

Parameters in parametric infons may be abstracted over in much the same way as λ-abstraction may abstract variables in the λ-calculus. For example, given the parametric infon:

$$\langle\langle \text{naming, name: "Kim", named: } \dot{x}; + \rangle\rangle$$

we may abstract over the parameter $\dot{x}$ to form a property — the property of being named Kim. This property is denoted:

$$[\dot{x} \mid \langle\langle \text{naming, name: "Kim", named: } \dot{x}; + \rangle\rangle]$$

Complex properties formed in this manner may participate in infons in the same way as basic properties. Furthermore, for any situation S and property $[\dot{x} \mid C(\dot{x})]$, we have the following closure properties.

If

$$S \models \langle\langle [\dot{x} \mid C(\dot{x})], \text{arg: } z; + \rangle\rangle$$

(where z is a metavariable ranging over objects) then

$$S \models C(z)$$

and if

$$S \models \langle\langle [\dot{x} \mid C(\dot{x})], \text{arg: } z; - \rangle\rangle$$

then

$$S \not\models C(z)$$

We resist making the stronger claim:

$$\text{If } S \models \langle\langle [\dot{x} \mid C(\dot{x})], \text{arg: } z; - \rangle\rangle \text{ then } S \models \overline{C(z)}$$

Such a claim has in the past lead to problems of persistence, though see [Barwise 88a, p. 235], who suggests that this claim can be made coherently, and should in fact be made.

Whilst property abstraction is not employed by the fragment of HPSG presented in [Pollard & Sag 87], we employ abstraction for our treatment of the semantics of adjuncts. [Gawron & Peters 90a] also employ property abstraction in their treatment of quantification and anaphora. Abstraction here is crucial to their analysis.

Simultaneous abstraction of more than one parameter is allowed, so that we may form n-place relations by simultaneously abstracting over n parameters. With respect to the above closure properties, the above behaviour extends to such complex relations of any arity. Abstraction interacts in an interesting way when an abstracted parameter is assigned to more then one argument role of the infon in question. In such cases, argument roles linked by parameters remain linked after the abstraction. This allows the formation of complex properties and relations, such as the property (or unary relation) of *loving oneself*:

$$[\dot{x} \mid \langle\langle \text{loving, lovee: } \dot{x}, \text{ lover: } \dot{x}; + \rangle\rangle]$$

2.8 Variables

[Pollard & Sag 87] distinguish between (bound) variables and parameters. Standard situation theory does not employ variables (bound or otherwise), which arise solely out of Pollard and Sag's non-standard use of quantificational infons (see below). The variables of Pollard and Sag correspond closely to bound variables in predicate calculus. They are intended to range over objects of the theory, and this range can be restricted in the same way as the range of parameters. Although Pollard and Sag say nothing about the status of quantifiers, it seems that they are meant to bind variables. The way the theory is developed also suggests that variables must carry the same sort of agreement features as parameters. CPSG does not employ variables in its treatment of semantics.

2.9 Compound Infons

As well as the basic infons described above, HPSG makes use of *conjunctive*, *disjunctive*, and *quantificational* infons. Such infons are not used by the version of situation theory employed by CPSG but we include a discussion of them here for completeness.

2.9.1 Conjunctive and Disjunctive Infons

Conjunctive and disjunctive infons correspond to a set of conjoined or disjoined infons. Note that this is not logical conjunction or disjunction definable by a truth table, as infons are not, in themselves, true or false. However, a situation supports a conjunction of infons just in case it supports each individual infon, and a situation supports a disjunction of infons just in case it supports at least one of the disjuncts. Conjunctive and disjunctive infons are normally expressed in terms of a connective ($\wedge$ or $\vee$) followed by a set of infons:

$$\wedge\{\sigma_1, \sigma_2\}$$

$$\vee\{\tau_1, \tau_2, \tau_3\}$$

The interaction of abstraction with multiple occurrences of parameters discussed above extends to conjunctive and disjunctive infons, resulting in the formation of more complex

relations and properties, such as the *unrequited love* relation:

$$\left[\dot{x}, \dot{y} \;\middle|\; \bigwedge \left\{ \begin{array}{l} \langle\langle \text{loving, lovee: } \dot{x}, \text{ lover: } \dot{y}; + \rangle\rangle \\ \langle\langle \text{loving, lovee: } \dot{y}, \text{ lover: } \dot{x}; - \rangle\rangle \end{array} \right\} \right]$$

The notion of the dual of an infon can be extended to conjunctive and disjunctive infons via de Morgan's laws. If $\{C_i \mid i \in I\}$ is a set of infons, all of whose duals exist (this clause is explained when we look at quantificational infons), then

$$\overline{\vee\{C_i \mid i \in I\}} \stackrel{\text{def}}{=} \wedge\{\overline{C_i} \mid i \in I\}$$

$$\overline{\wedge\{C_i \mid i \in I\}} \stackrel{\text{def}}{=} \vee\{\overline{C_i} \mid i \in I\}$$

2.9.2 Quantificational Infons

The quantificational infons of HPSG are composed of a determiner, a restricted variable, and an infon containing that variable:

$$(\text{most } x \mid S \models \langle\langle \text{cat, instance: } x; + \rangle\rangle)\ \langle\langle \text{miaow, miaower: } x; + \rangle\rangle$$

The variable x in $\langle\langle$miaow, miaower: x; $+\rangle\rangle$ is said to be bound by the quantifier expression.

There are a number of points to be made about this treatment of quantification. Typically, workers in situation semantics treat determiners as relations between properties — most things having the property of cathood have the property of miaowing. Thus quantification is usually treated via basic infons. The basic infon which would normally be associated with *most cats miaow* is:

$$\langle\langle \text{most, restriction: } [\dot{x} \mid \langle\langle \text{cat, instance: } \dot{x}; + \rangle\rangle], \text{ scope: } [\dot{x} \mid \langle\langle \text{miaow, miaower: } \dot{x}; + \rangle\rangle]; + \rangle\rangle$$

There are points in favour of each treatment. Under the HPSG account, the syntax of quantificational infons is similar to that of sentences in the logic of generalised quantifiers developed in [Barwise & Cooper 81]. The syntax of this logic was designed to parallel that of English, and the contribution of the determiner, noun, and predicate to the syntactic formula is clear. On the other hand, it is model theoretic considerations, and not syntactic ones, which concern situation theorists. There are also possible similarities

between the HPSG approach and that of dynamic predicate logic ([Groenendijk & Stokhof 90]), in that it is possible to imagine how Pollard & Sag might treat intersentential anaphora, as in:

[Several men]$_i$ howled. They$_i$ were drunk.

However, [Pollard & Sag 87] do not discuss the details of variable binding and scope, and it is far from clear just how the situation S (if it is even a situation) described below should be thought of.

$S \models (\text{several } x_{S \models \langle\langle \text{man, instance: } x; + \rangle\rangle})\ \langle\langle \text{howl, howler: } x; + \rangle\rangle$

$S \models \langle\langle \text{drunk, instance: } x; + \rangle\rangle$

To further pursue this approach to intersentential anaphora would require a precise theory of variables, which [Pollard & Sag 87] do not give. We do not attempt to explore this here because, as mentioned above, CPSG does not employ such variables.

One final point about HPSG's use of quantificational infons is that it is not clear how, if it all, the dual of quantificational infons should be defined. Such a notion is clearly useful (in the treatment of the negation of quantified sentences, for example), but Pollard & Sag do not broach the subject. If duals are not defined for quantificational infons, then other compound infons involving quantificational infons (such as conjunctive or disjunctive infons) cannot have duals either.

In Chapter 8 we discuss various treatments of quantification, and the use of compound infons, in detail. Pre-empting this discussion, we argue that quantification and coordination can be treated with other tools available in situation theory, and that compound infons are thus not necessary: in our version of situation theory, all infons are basic.

2.10 Propositions

Given a situation S and an infon σ, we can form the proposition that S supports σ. This proposition is written:

$$(S \models \sigma)$$

Propositions have absolute truth values: they are true or false. The proposition $(S \models \sigma)$ is true if and only if S does indeed support σ. Essentially propositions are the absolute analogue of infons, the truth of infons being relative to a situation.

There is no partiality in the domain of propositions. Whilst a situation may not positively or negatively resolve an issue, all propositions are necessarily either true or false. If a situation S does not resolve an issue formed by an assignment a to the argument roles of a relation R, then the two propositions

$$(S \models \langle\langle R, a; + \rangle\rangle)$$

and

$$(S \models \langle\langle R, a; - \rangle\rangle)$$

are simply both false. There is no partiality.

The use of parameters in the propositional domain parallels their use in the infon domain: just as we have parametric infons, we have parametric propositions. Parametric propositions may have a parameter corresponding to (or perhaps embedded in) an argument role of the infon involved or a parameter corresponding to the situation which it is claimed supports the infon. Thus we may have the following parametric proposition:

$$(\dot{s} \models \langle\langle R, a; + \rangle\rangle)$$

Parametric propositions do not have truth values. They are underdetermined objects whose truth can only be determined once all of their parameters have been anchored.

2.11 Types

Just as the ontology of situation theory includes propositions, which are the absolute analogue of infons, situation theory includes *types*, which are the absolute analogue of relations (including unary relations, or properties). The truth of a statment of the form "the object x has the property P" is situation dependent: in one situation the object may have the property whereas in another it may not. The truth of the analogous statment "the object x is of type T" is situation independent: the object is, or is not, of the type.

Types have argument roles and appropriateness conditions on those argument roles in exactly the same way as relations do. Furthermore, just as an n-place relation and n appropriate objects may be put together to form an issue, an n-place type and n appropriate objects may be put together to form a proposition, the proposition that the objects are of the type. In the same way that we are able to abstract over parameters in parametric infons, we may abstract over parameters in parametric propositions.

One important 2-place type which we have already seen is '$\models$'. Appropriate arguments for this type are a situation and an infon, and a situation-infon pair (S,σ) is of the type '$\models$' if and only if the proposition $(S \models \sigma)$ is true, i.e., if and only if S does indeed support σ.

There are two distinct sorts of unary type which are of special interest: *object types* and *situation types*. In going from a parametric proposition to a type, if we abstract over a parameter filling an argument role of the infon, we are lead to an object type. On the other hand, if we abstract over a parameter corresponding to the situation which the proposition claims supports the infon, we are lead to a situation type.

Situation types lead to a second method for characterising situations. As indicated above we may characterise a situation directly by the infons which it supports. Alternatively, we may characterise a situation as being of various types. The correspondence between these two characterisations comes from the fact that for any (non-parametric) infon there is a corresponding situation type — the type of situation which supports that infon. If σ is an infon, then the type of situation which supports σ is denoted:

$$[\dot{s} \mid (\dot{s} \models \sigma)]$$

and

$$S : [\dot{s} \mid (\dot{s} \models \sigma)] \text{ iff } S \models \sigma$$

Here the ':' notation is read as "is of type".

An object can have a property in one situation but not in another (corresponding to Tom miaowing in one situation but not in another), whereas an object being of a certain type is independent of any situation. The object either is or is not of the type (corresponding to an object being, or not being, a cat) no matter what situation the object finds itself

in. This suggests that it might be useful to postulate a maximal situation W (thought of as "the world") which object types might be stated with respect to, so the fact that an object c is of type "cat" might be expressed in terms of the property "cathood" as:

$$c : [\dot{x} \mid (W \models \langle\langle \text{cat, instance: } \dot{x}; + \rangle\rangle)]$$

The importance of types will become clear in Chapter 8, where we make crucial use of the situation independence of types in treating quantification.

2.12 Constraints

A constraint is a special sort of relation between situation types. A situation type S_1 *involves* the situation type S_2 if and only if whenever a situation is of type S_1 there exists an extension of that situation which is of type S_2. Such a constraint might be written officially as:

$$\langle\langle \text{involves, arg1: } S_1, \text{ arg2: } S_2; + \rangle\rangle$$

To emphasis the special nature of constraints an alternate notation is often used:

$$S_1 \Rightarrow S_2$$

Constraints need not be "positive", like the *involves* constraint. A situation type S_1 *precludes* the situation type S_2 if and only if whenever there is a situation of type S_1 that situation is not also of type S_2. In the alternate notation this is written:

$$S_1 \perp S_2$$

Agents may be *attuned* to constraints. An agent that is attuned to a positive constraint of the form:

$$[\dot{s} \mid (\dot{s} \models \sigma)] \Rightarrow [\dot{t} \mid (\dot{t} \models \tau)]$$

may infer from a situation of type $[\dot{s} : (\dot{s} \models \sigma)]$ that there exists a larger situation of type $[\dot{t} : (\dot{t} \models \tau)]$. That is, from a situation supporting σ, the agent may infer that there exists a larger situation also supporting τ.

Similarly an agent that is attuned to a negative constraint of the form:

$$[\dot{s} \mid (\dot{s} \models \sigma)] \perp [\dot{t} \mid (\dot{t} \models \tau)]$$

may infer from a situation of type $[\dot{s} : (\dot{s} \models \sigma)]$ that that situation is not of type $[\dot{t} : (\dot{t} \models \tau)]$. That is, from a situation supporting σ, the agent may infer that the situation does not support τ.

Conditional constraints are ternary relations between three situation types. An agent attuned to a conditional constraint of the form

$$S_1 \Rightarrow S_2 \mid B$$

may infer from a situation of type S_1 that there exists a larger situation of type S_2, provided that the original situation (of type S_1) is also of type B. B is known as the background situation type. As explained in section 1.3, it is important that the background situation type is not incorporated into the antecedent.

2.13 The Information Carried by a Situation

Situations can carry information without actually supporting it in virtue of constraints. A situation S carries the information σ with respect to the constraint C, written:

$$S \mid\!\vdash_C \sigma$$

if and only if either $S \models \sigma$, in which case the information is carried trivially, or C is $T_1 \Rightarrow T_2$ and S is of type T_1 and every larger situation S' of type T_2 is such that $S' \models \sigma$.

To illustrate this use of constraints, consider the situation meaning of a particular utterance of "I like Tigger". This utterance might be taken to yield the following constraint, C:

$$\left[\dot{S}_1 \left| \left(\left\{ \begin{array}{c} \dot{S}_1 \models \langle\langle \text{name}, t, \text{'Tigger'}; + \rangle\rangle \\ \dot{S}_1 \models \langle\langle \text{speaker}, \text{"I like Tigger"}, r; + \rangle\rangle \end{array} \right\} : \wedge \right) \right] \Rightarrow \left[\dot{S}_2 \left| \left(\dot{S}_2 \models \langle\langle \text{like}, r, t; + \rangle\rangle \right) \right.\right]$$

(Here $\wedge$ is a one place type which is true of a set of propositions if and only if each proposition in the set is true.) A particular situation S which is of the type given in the antecedent will carry information with respect to C:

$$S \mid\!\vdash_C \langle\langle \text{like}, r, t; + \rangle\rangle$$

The infon carried by S is the situation meaning of the utterance.

To get at the situation type meaning of a sentence, we need to consider parametric constraints, constraints relating parametric situation types. The situation type meaning of the sentence 'I like Tigger' is the parametric constraint

$$\left[\dot{S}_1 \left| \left(\left\{ \begin{array}{c} \dot{S}_1 \models \langle\langle \text{name}, \dot{t}, \text{'Tigger'}; + \rangle\rangle \\ \dot{S}_1 \models \langle\langle \text{speaker}, \text{"I like Tigger"}, \dot{s}; + \rangle\rangle \end{array} \right\} : \wedge \right) \right] \Rightarrow \left[\dot{S}_2 \left| \left(\dot{S}_2 \models \langle\langle \text{like}, \dot{s}, \dot{t}; + \rangle\rangle \right) \right.\right]$$

In a particular utterance situation S_u, which is such that:

$$S_u \models \langle\langle \text{name}, t, \text{'Tigger'}; + \rangle\rangle$$

$$S_u \models \langle\langle \text{speaker}, \text{"I like Tigger"}, r; + \rangle\rangle$$

there is an anchoring of the parameters in the antecedent, namely that mapping $\dot{t}$ to t and $\dot{s}$ to r, such that S_u is of the type given in the antecedent. Anchoring the parameters as such leads to the non-parametric constraint above, and as above the information conveyed with respect to this constraint is $\langle\langle \text{like}, r, t; + \rangle\rangle$, as required.

3 Some Mathematical Models of Situation Theoretic Objects

In this section we give some brief notes on how some situation theoretic objects might be modelled. This is intended to help make precise the notions discussed in the previous sections of this chapter.

3.1 Objects

The class OBJ of non-parametric situation theoretic objects is the disjoint union of the set IND of individuals, the class REL of relations, the class INF of infons, the class SIT of situations, the class TYPE of types, and the class PROP of propositions. Each of these sets/classes is described below.

The class PAROBJ of (possibly) parametric situation theoretic objects is the disjoint union of the analogous parametric classes (where such classes exist) — the set of individuals, the class of parametric relations, the class of parametric infons, the class of situations, the class of parametric types, and the class of parametric propositions — together with the set of parameters.

All classes and the subclass relationships between them are shown in Figure 6.1. All classes with convex rounded boundaries are subclasses of the classes with which they share those boundaries.

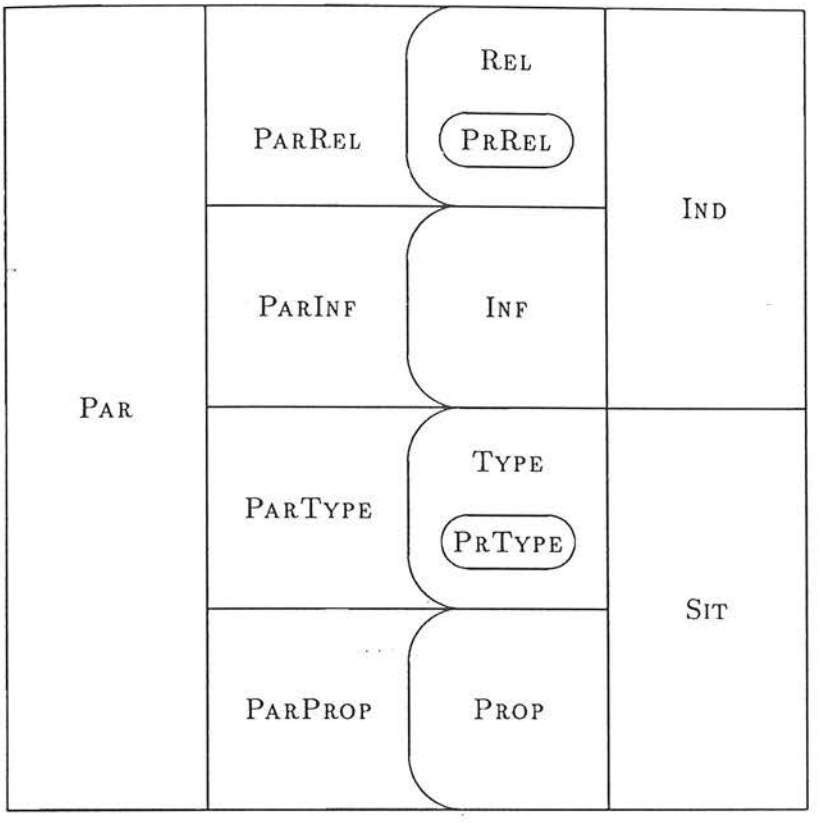


Figure 6.1: Situation Theoretic Objects

PRREL, the set of primitive relations, is a subset of REL, the class of (non-parametric) relations, which is a subclass of PARREL, the class of possibly parametric relations. Similarly, PRTYPE, the set of primitive types, is a subset of TYPE, the class of (non-parametric) types, which is a subclass of PARTYPE, the class of possibly parametric types. INF, the class of infons is a subclass of PARINF, the class of possibly parametric infons, and PROP, the class of propositions is a subclass of PARPROP, the class of possibly parametric propositions. Note that we take PRREL and PRTYPE to be proper sets, rather than arbitrary classes. Whilst it is possible to take the primitives as sets, the operations for forming complex relations and types lead to classes of each.

3.2 Individuals

We model the structured domain of individuals by a set IND. This set may be structured by, for example, *part of* relationships, though such structure is beyond that required by situation theory. Grouping operations, such as pairing or set formation, may also by responsible for structure on IND. Again these grouping operations are over and above the structure required by situation theory.

3.3 Primitive Properties and Relations

The apparatus of (primitive) relations (minus appropriateness conditions) may be modelled via a triple $\langle \text{PRREL}, X, \nu \rangle$, where

- PRREL is a set of atoms corresponding to the relation symbols,
- X is the set of atoms corresponding to the argument roles, and
- $\nu : [\text{PRREL} \rightarrow Pow(X)]$ assigns an arity (i.e., a set of argument roles) to each $\omega \in \text{PRREL}$.

For example, the triple defined by:

- PRREL = {miaow, give, appear, believe},
- X = {agent, patient, theme, goal}, and
- $\nu : [\text{PRREL} \rightarrow Pow(X)]$ such that

$$\begin{aligned} \nu : \text{miaow} &\mapsto \{\text{agent}\} \\ \nu : \text{give} &\mapsto \{\text{agent, theme, goal}\} \\ \nu : \text{appear} &\mapsto \{\text{theme}\} \\ \nu : \text{believe} &\mapsto \{\text{agent, theme}\} \end{aligned}$$

might model the relations corresponding to the English words 'miaow', 'give', 'appear', and 'believe'.

With respect to appropriateness conditions, we might informally require that, for example, only propositions are appropriate for the theme-role of 'appear', whilst only individuals are appropriate for the theme-role of 'give'. Thus, appropriateness conditions are a function of the argument role and the relation that argument role is an argument role of, not just a function of the argument role.

3.4 Basic Infons

A domain of (basic) infons, INF, is given by a quadruple $\langle \text{PRREL}, X, \nu, \text{OBJ} \rangle$, where the triple $\langle \text{PRREL}, X, \nu \rangle$ defines the relations and OBJ are the objects which may fill the argument roles of those relations. Relations are objects, and so we therefore require that $\text{PRREL} \subset \text{OBJ}$.

An issue is modelled by a pair (ω, θ), where

- $\omega \in \text{PRREL}$, and
- $\theta : [\nu\omega \rightarrow \text{OBJ}]$ is an assignment of (appropriate) objects to the argument roles of ω.

The class of all such pairs is ISS, the domain of issues.

A (basic) infon may then be modelled by a pair (ι, ρ), where

- $\iota \in \text{ISS}$, and
- $\rho \in \text{POL}$, where POL is the set $\{+, -\}$.

INF is the class of all such pairs.

We also have that $\text{INF} \subset \text{OBJ}$. Given that an issue involves an assignment of objects to argument roles, we therefore admit circular objects. For example, given the tuple $\langle \{R\}, \{\alpha\}, \nu \rangle$ defining a domain of relations, where $\nu(R) = \{\alpha\}$, we may form the basic infon $((R, \theta), +)$ where $\theta : \alpha \mapsto ((R, \theta), +)$. We cannot therefore assume an underlying well-founded set theory. [Aczel 88] details a suitable theory of non-well-founded sets.

3.5 Situations

We may model a situation by the class of basic infons which it supports. The "part of" relation $\trianglelefteq$ which holds between situations may then be characterised in our model as class inclusion. If we model the situations S and T by the classes S' and T' respectively, then $S \trianglelefteq T$ iff $S' \subseteq T'$.

This approach to modelling situations allows incoherent situations: the tools allows us to model situations which support both an infon and its dual. An alternate approach

to modelling situations stems from the view of a situation as a partial possible world (see [Perry 87]). On this view, a situation resolves a subset of all possible issues. Consequently we can see a situation as being a partial function from the class of issues to the set of polarities. This will only allow the modelling of basic infons. As we do not admit compound infons, this is of no concern here. If compound infons are to be admitted, then a situation may be modelled in terms of the basic infons which it supports, with the support of compound infons being determined in terms of the basic infons which constituent those compound infons. An alternate model of a situation S is thus a partial function S' from ISS, the class of issues, into POL such that

$$S \models \sigma \text{ iff } S'(\iota) = \rho$$

where (ι, ρ) models σ

Within such a model, if S is modelled by S' and T by T', then $S \trianglelefteq T$ iff T' is an extension of S', i.e., for all ι on which S' is defined, T' is defined and $S'(\iota) = T'(\iota)$.

3.6 Primitive Types

Types are the absolute, i.e., non-situated, analogues of properties and relations. They may thus be modelled in a roughly analogous way with a set of atoms called primitive type symbols, a set of argument roles, and an assignment of argument roles to primitive type symbols. In being absolute, however, types also have fixed extensions. We thus model the apparatus of types with a quintuple $\langle \text{PRTYPE}, X, \nu, \text{OBJ}, \mu \rangle$, where

- PRTYPE is a set of atoms called primitive type symbols,
- X is a set of atoms known as argument roles,
- ν assigns an arity, i.e., a set of argument roles, to each primitive type symbol,
- OBJ is a set, the elements of which may fill the argument roles of a primitive type symbol, and
- μ assigns an extension, a subset of $\{f \mid f : \nu\omega \rightarrow \text{OBJ}\}$, to each $\omega \in \text{PRTYPE}$.

Of course, $\text{PRTYPE} \subset \text{OBJ}$.

One primitive type of special importance is '$\models$'. This type has two argument roles, *sit* and *inf*, and the extension of $\models$ is the set of all functions mapping *sit* to a situation S

and *inf* to an infon σ such that S supports σ. The extension of $\models$ is thus *structurally determined*. Other structurally determined types, such as logical conjunction, disjunction and negation, as well as unary types such as "is an individual" and "is a relation", may also be included in PRTYPE.

3.7 Propositions

A proposition is modelled by a pair (ω, θ) consisting of a type symbol and an (appropriate) assignment of objects to the argument roles of that type symbol. The proposition evaluates to T, written $[(\omega, \theta)] = \mathbf{T}$, if θ is in the extension of ω, i.e., if $\theta \in \mu\omega$, and F otherwise. A proposition of the form $(\models, \theta)$ is thus true if θ maps *sit* to a situation S and *inf* to an infon σ and S supports σ.

3.8 Parameters and Parametric Objects

We may model parameters by a set PAR of atoms. From this and the situation theoretic objects discussed above we may construct classes of parametric relations, parametric infons, parametric types, and parametric propositions.

One variety of parametric infon may be constructed in much the same way as non-parametric infons by allowing parametric objects in the range of the assignment to argument roles of primitive relations. Similarly a variety of parametric proposition may be constructed by allowing parametric objects in the range of the assignment to argument roles of primitive types. The other variety of parametric infon/proposition involves complex relations/types which themselves contain parameters. To construct such infons/proposition we need first to construct complex relations/types by abstracting over parameters in other infons/propositions.

Before we do this we introduce a function *params* from the class of parametric objects to the power set of parameters. This function maps any (parametric) object to the set of parameters in that object, and is defined by cases:

$$params(x) = \begin{cases} \{\} & \text{if } x \in \textsc{Obj} \\ \{x\} & \text{if } x \in \textsc{Par} \\ params_{\textsc{ParRel}}(x) & \text{if } x \in \textsc{ParRel} \\ params_{\textsc{ParInf}}(x) & \text{if } x \in \textsc{ParInf} \\ params_{\textsc{ParType}}(x) & \text{if } x \in \textsc{ParType} \\ params_{\textsc{ParProp}}(x) & \text{if } x \in \textsc{ParProp} \end{cases}$$

where

$$params_{\textsc{ParInf}}(((\omega,\theta),\iota)) = params(\omega) \cup \bigcup_{x \in \nu\omega} params(\theta(x))$$

$$params_{\textsc{ParProp}}((\omega,\theta)) = params(\omega) \cup \bigcup_{x \in \nu\omega} params(\theta(x))$$

The clauses for $params_{\textsc{ParRel}}$ and $params_{\textsc{ParType}}$ are given below.

An anchor is a function Anc from the set PAR of parameters to the set OBJ of (non-parametric) objects. Any anchor Anc may be uniquely extended to a function Anc^* which maps the class of all parametric objects to the class of (non-parametric) objects. Anc^* is defined by cases:

$$Anc^*(x) = \begin{cases} x & \text{if } x \in \textsc{Obj} \\ Anc(x) & \text{if } x \in \textsc{Par} \\ ((Anc^*(\omega), Anc^* \circ \theta), \rho) & \text{if } x = ((\omega,\theta),\rho) \in \textsc{ParInf} \\ (Anc^*(\omega), Anc^* \circ \theta) & \text{if } x = (\omega,\theta) \in \textsc{ParProp} \end{cases}$$

The other cases ($x \in \textsc{ParRel} \cup \textsc{ParType}$) are given below.

3.9 Abstraction

Given a parametric object, we may abstract over one or more of the parameters to form an object with fewer parameters.

3.9.1 Complex Relations

Given a parametric infon $((\omega,\theta),\rho)$, where $params(((\omega,\theta),\rho)) = \phi$, we may form the complex relation $(\psi,((\omega,\theta),\rho))$, where ψ is some non-empty subset of ϕ. This relation

has one argument role for each member of ψ and has as parameters those elements of ϕ not in ψ. If $\phi = \psi$ then the complex relation is non-parametric. PARREL is the closure of the set PRREL under this operation of complex relation formation.

We still need to specify the argument roles of such complex relations. This is not straightforward, primarily because in general the parameters we abstract over may be buried indefinitely far in the parametric infon — the arguments of the parametric infon may include not just parameters but other more complex parametric objects whose parameters may legitimately be abstracted. We leave this as an incomplete aspect of our model.

Lastly we must define *params* and Anc^* for the domain of parametric relations. These functions are not a problem:

$$params_{\text{ParRel}}(x) = \begin{cases} \{\} \\ \qquad \text{if } x \in \text{PRREL} \\ params(((\omega,\theta),\rho)) \setminus \psi \\ \qquad \text{if } x \text{ is of the form } (\psi,((\omega,\theta),\rho)) \end{cases}$$

REL is the class of all members x of PARREL for which $params_{\text{ParRel}}(x) = \{\}$.

$$Anc^*(x) = \begin{cases} x \\ \qquad \text{if } x \in \text{REL} \\ (\psi,((Anc^*_\psi(\omega), Anc^*_\psi \circ \theta),\rho)) \\ \qquad \text{if } x \text{ is of the form } (\psi,((\omega,\theta),\rho)) \end{cases}$$

where

$$Anc^*_\psi(x) = \begin{cases} x & \text{if } x \in \psi \\ Anc^*(x) & \text{if } x \notin \psi \end{cases}$$

We note that the two clauses agree if x is both in REL and of the form $(\psi,((\omega,\theta),\rho))$, ensuring that Anc^* is well defined.

3.9.2 Complex Types

Complex types may be formed from parametric propositions in an analogous way to complex relations. Given a parametric proposition (ω,θ), where $params((\omega,\theta)) = \phi$, we may form the complex type $(\psi,(\omega,\theta))$, where ψ is some non-empty subset of ϕ. This type has one argument role for each member of ψ and has as parameters those elements

of ϕ not in ψ. If $\phi = \psi$ then the complex type is non-parametric. PARTYPE is the closure of the set PRTYPE under this operation of complex type formation. Again we really need to specify the argument roles of complex types. As with complex relations, we leave this as incomplete.

The definitions of *params* and Anc^* for complex/parametric types follow that for complex/parametric relations:

$$params_{\text{ParType}}(x) = \begin{cases} \{\} \\ \qquad \text{if } x \in \text{PRTYPE} \\ params((\omega, \theta)) \setminus \psi \\ \qquad \text{if } x \text{ is of the form } (\psi, (\omega, \theta)) \end{cases}$$

TYPE is the class of all members x of PARTYPE for which $params_{\text{ParType}}(x) = \{\}$.

$$Anc^*(x) = \begin{cases} x \\ \qquad \text{if } x \in \text{TYPE} \\ (\psi, (Anc^*_\psi(\omega), Anc^*_\psi \circ \theta)) \\ \qquad \text{if } x \text{ is of the form } (\psi, (\omega, \theta)) \end{cases}$$

We note that the two clauses agree if x is both in TYPE and of the form $(\psi, (\omega, \theta))$, ensuring that Anc^* is well defined.

4 Summary

In this chapter we have presented the semantic theory which CPSG employs in its treatment of natural language semantics. This theory, a version of situation theory, differs markedly from possible world approaches to semantics, as explained in section 1, and takes meaning to be a relation between situation types, rather than identifying it with a set of possible worlds. The objects of the theory are briefly described in prose in section 2, and much of this prose is made more precise in section 3 by the use of mathematical models of many situation theoretic objects. We now move on to the application of this theory, with a comprehensive presentation of the SEMANTICS attribute of CPSG signs.

Chapter 7

The SEMANTICS Attribute

In this chapter we present much of the CPSG treatment of semantics, complementing the three earlier chapters on syntax. We begin in section 1 with a discussion of the use of feature structures to model situation theoretic objects. In section 2 we look at the issue of unification versus predication, which arises from the combination of the ontology of situation theory with a unification based formalism. Section 3 introduces the semantics of most of the lexical types previously mentioned. We do not here discuss the semantics of determiners, leaving that, and the treatment of quantification to Chapter 8. Sections 4, 5 and 6 discuss the semantics of various phrasal types: head/argument phrases, head/adjunct phrases and coordinate phrases respectively. An extended example is then given in section 7, illustrating the treatment of tense and auxiliaries, to further clarify the mechanisms presented throughout the chapter.

1 Describing Situation Theoretic Objects with Feature Structures

In the previous chapter we introduced a number of different situation theoretic objects. The use of a single language to represent syntactic and semantic information (i.e., the language of feature structures) requires that we be able to model those situation theoretic objects required by our semantic treatment with feature structures. In this section we consider how this modelling is carried out, both by HPSG and by CPSG. We thus discuss the modelling of atomic relations and atomic types, parameters (which, following HPSG, we use in the treatment of agreement), infons, situations, propositions, restricted

parameters and complex relations and complex types. Broadly speaking, parameters (possibly restricted) are associated with nominal constituents and parametric propositions are associated with verbal constituents. In modelling these objects, however, we must also model their constituents, and hence we must also model situations, relations, types and infons. Note that we do not need to be able to model individuals. The semantic representations that we build are parametric. The connection to individuals is achieved through the utterance specific speaker connections, which anchor the parameters involved in the representations to individuals in the resource and described situations.

1.1 Content, Context and Speaker Connections

The value of the SEMANTICS attribute of any constituent is a feature structure which is defined on the attributes CONTENT, CONTEXT and PARAMETERS. The value of the CONTENT attribute for any constituent describes a situation theoretic object corresponding to that constituent. In general, for saturated nominal constituents the CONTENT describes a parameter, and for saturated verbal constituents the CONTENT describes a parametric proposition. The value of the CONTEXT attribute describes a set of parametric propositions. The elements of this set normally represent restrictions on the parameters in the CONTENT. The value of the PARAMETERS attribute describes a set of parameters, the set of all parameters occurring in the CONTENT and CONTEXT. On any particular occasion of use, the speaker connections must anchor each parameter in this set so that each parametric proposition described in the CONTEXT set is mapped to a true proposition. These speaker connections will also anchor any parameters in the CONTENT, and thus map the CONTENT to a proposition, the proposition expressed by the utterance. The general form of the SEMANTICS attribute of a feature structure description of a constituent is thus:

$$\begin{bmatrix} \text{SEMANTICS} & \begin{bmatrix} \text{CONTENT} & [\ ] \\ \text{CONTEXT} & \{\ldots\} \\ \text{PARAMETERS} & \{\ldots\} \end{bmatrix} \end{bmatrix}$$

This may be compared to the SEMANTICS attribute of an HPSG sign, which bears only the attributes CONTENT and CONTEXT. There are two important differences. Firstly,

HPSG's CONTEXT attribute describes restricted parameters — parameters together with restrictions on those parameters. In CPSG parameters are described by the PARAMETERS attribute and restrictions on those parameters are described by the CONTEXT attribute. The second differences involves the objects that the values of the CONTENT and CONTEXT attributes describe. Using propositions rather than infons allows more control over the precise interpretation of the values of the semantic attributes by making explicit the situations which the infons of HPSG are making claims about.

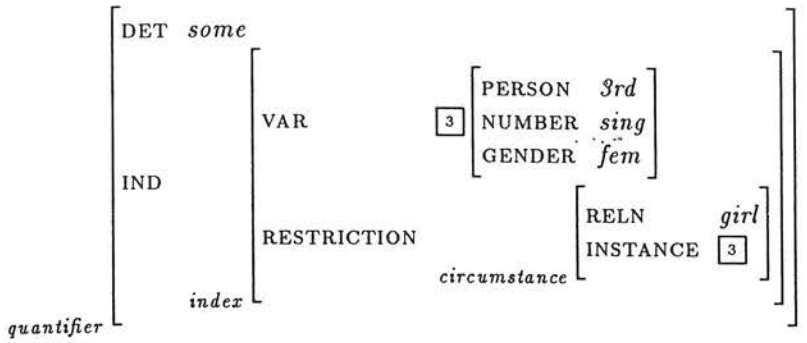
To summarise then,

> The speaker connections must anchor all parameters in the PARAMETERS set such that all propositions specified in the CONTEXT set are true. The claim made by the utterance is then given by the proposition specified by the CONTENT attribute.

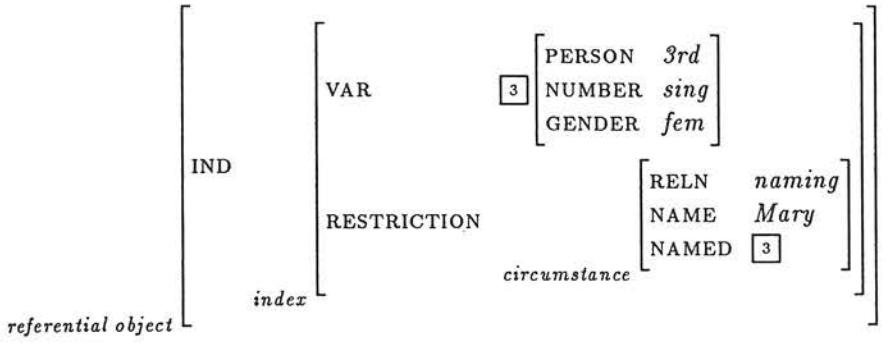
1.2 Parameters

Within HPSG, the semantic content of any noun phrase is modelled by a sorted feature structure of sort *indexed object*. There are two subsorts of *indexed object*: *referential object* and *quantifier*. Feature structures of sort *referential object* model parameters, which correspond to the semantic content of proper nouns. Feature structures of sort *quantifier* model bound variables (i.e., variables bound by a determiner), which are taken to be the semantic content of noun phrases involving quantifier expressions. CPSG entertains only one sort of parameter, similar to that modelled in HPSG by a feature structure of sort *referential object*. This is possible because CPSG employs a significantly different treatment of quantification. The difference means that much of the complexity in the modelling of parameters in HPSG can be avoided.

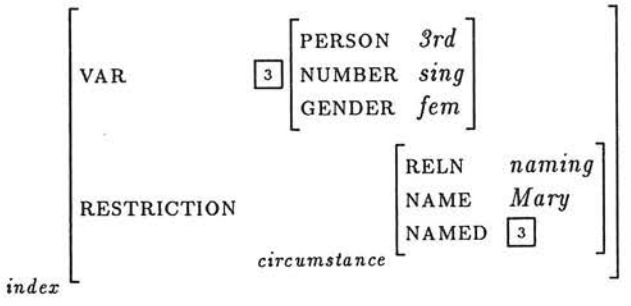
A third person singular feminine variable bound by the determiner *some* is modelled in HPSG by the following sorted feature structure:



A third person singular feminine parameter, on the other hand, such as one introduced by an utterance of 'Mary', is modelled by:



Within HPSG it seems that the need to model referential objects and quantifiers by feature structures which are subsorts of a common sort leads to unnecessary complications in the case of referential objects. In short, it seems strange that it is necessary to embed the *index* one level deep in the feature structure modelling a referential object as is done here. It is unclear why a parameter (of sort referential object) should not be modelled directly with a feature structure of sort *index*, such as:



Restrictions in CPSG are stated as part of the context, and not as part of the content, and so, as discussed below, even for restricted parameters, the RESTRICTION attribute is not required.

A further criticism of the HPSG approach to variables is that the VAR attribute of a *quantifier* is interpreted as modelling a variable, whereas the VAR attribute of a *referential object* is interpreted as modelling a parameter. This has the unfortunate effect of encouraging the distinction between (bound) variables and parameters to be blurred. Recall that CPSG avoids variables all together, and it is thus unnecessary to employ a VAR attribute. Parameters in CPSG are modelled simply by feature structures bearing the three agreement attributes. Thus we have, for example:

$$\begin{bmatrix} \text{PERSON} & \textit{3rd} \\ \text{NUMBER} & \textit{sing} \\ \text{GENDER} & \textit{fem} \end{bmatrix}$$

As suggested by this feature structure, agreement in CPSG is treated much as in HPSG, via features associated with parameters. The treatment of agreement as a semantic phenomenon follows [Pollard & Sag 88], but they are careful to avoid equating agreement features with features of real world objects. In particular, [Pollard & Sag 88, p. 18] stress that they

> are not saying that the world is simply divided into singular, plural, masculine, feminine and neuter objects, and that pronouns are restricted as to what sorts of things they may refer to.

Rather, their account "localizes agreement features within referential parameters" ([Pollard & Sag 88, p. 18]). The use of features on parameters must not be taken to be a convenient shorthand for restrictions which those features encode. Agreement features do not correspond to special restrictions which might have been treated via the CONTEXT attribute, but which have instead been elevated to this special role because of their importance in language. Such an approach would suffer from grave difficulties in treating differences between natural and grammatical gender.

Incorporating parameters into a feature-based framework raises some particular problems concerning abstraction, binding and scope. Abstracting over a parameter should only bind occurrences of the parameter in the scope of the abstraction, but it is not clear how this can be effected in a feature structure framework. We do not offer a solution to this problem, which has arisen in previous work (see, for example, [Pollard 89b]).

1.3 Infons and Propositions

In both HPSG and CPSG atomic relations and atomic types are modelled by atoms. Relations and types rarely occur in isolation, and normally the relation or type will be part of an infon or proposition. Unlike HPSG, CPSG also makes use of complex relations and complex types. These are discussed in detail below.

HPSG admits three sorts of infons, or circumstances: *basic circumstance*, *quantified circumstance*, and *compound circumstance*. In HPSG each of these must be modelled by distinct sorts of feature structures. The version of situation semantics employed by CPSG only admits a single sort of infon, that corresponding to a basic circumstance. The role played by quantified circumstances and compound circumstances is instead taken by propositions, as explained in Chapter 8.

All infons thus consist of a relation and an assignment of objects to the argument roles of that relation (forming an issue) together with a polarity. Propositions are similar in that they consist of a type and an assignment of objects to the argument roles of that type.

Two possibilities suggest themselves for the modelling of infons. We might employ a feature structure with an attribute for the relation, an attribute for each argument role of the relation, and a binary valued attribute for the polarity. Thus, a (parametric) infon such as:

$$\langle\langle \text{naming, name: "Kim", named: } \dot{x}; + \rangle\rangle$$

might be modelled by the feature structure:

$$\begin{bmatrix} \text{RELATION} & \textit{naming} \\ \text{NAME} & \textit{"Kim"} \\ \text{NAMED} & \boxed{1} \\ \text{POLARITY} & + \end{bmatrix}$$

where $\boxed{1}$ models the parameter $\dot{x}$.

Alternately, we might model an infon in terms of an issue and a polarity, with the issue itself modelled by a separate feature structure. That is, the above might be modelled as:

$$\begin{bmatrix} \text{ISSUE} & \begin{bmatrix} \text{RELATION} & \textit{naming} \\ \text{NAME} & \text{“}\textit{Kim}\text{”} \\ \text{NAMED} & \boxed{1} \end{bmatrix} \\ \text{POLARITY} & + \end{bmatrix}$$

This division of an infon into an issue and a polarity is preferred as it has certain advantages when treating negation: to form the description of an infon's dual from the description of the infon the issue can be copied directly from the model of the infon and the polarity reversed.

Following the modelling of infons, propositions may be modelled by feature structures bearing an attribute for the type and one attribute for each argument role of that type. Thus a proposition such as:

$$S \models \sigma$$

might be modelled by the feature structure:

$$\begin{bmatrix} \text{TYPE} & \textit{supports} \\ \text{SIT} & S' \\ \text{INF} & \sigma' \end{bmatrix}$$

where S' models the situation S and σ' models the infon σ.

In the hierarchically sorted system of feature structures employed within HPSG, relations and types may be described via the sorting mechanism. This is motivated by the fact that the relation or type determines the argument roles of the corresponding issue or proposition, and hence the attributes relevant to the description. Thus, rather than describing an infon as:

$$\begin{bmatrix} & & \begin{bmatrix} \text{RELN} & \textit{Believe} \\ \text{BELIEVER} & \boxed{1} \\ \text{BELIEF} & \boxed{2} \\ \text{LOCATION} & \boxed{3} \end{bmatrix} \\ \textit{infon} & \begin{bmatrix} \text{ISSUE} & \textit{issue} \\ \text{POLARITY} & + \end{bmatrix} \end{bmatrix}$$

It may be described as:

$$\textit{infon}\left[\begin{array}{ll} \text{ISSUE} & \textit{believe}\left[\begin{array}{ll}\text{BELIEVER} & \boxed{1}\\ \text{BELIEF} & \boxed{2}\\ \text{LOCATION} & \boxed{3}\end{array}\right]\\ \text{POLARITY} & + \end{array}\right]$$

where *believe* is a subsort of *issue*. This is similar to the approach adopted by Pollard in more recent work ([Pollard 88], [Pollard & Moshier 89]).

Polarities may also be incorporated into the sort system, with two subsorts of *infon*, *positive infon* and *negative infon*. Given this we may further alter the description to:

$$\textit{positive infon}\left[\begin{array}{ll} \text{ISSUE} & \textit{believe}\left[\begin{array}{ll}\text{BELIEVER} & \boxed{1}\\ \text{BELIEF} & \boxed{2}\\ \text{LOCATION} & \boxed{3}\end{array}\right]\end{array}\right]$$

This again raises the issue of where the line between information described in terms of attribute-value pairs and information described via the sort hierarchy in HPSG should be drawn. In individual cases it is difficult to argue either way, so CPSG is not losing anything by not having this option.

There are two issues regarding the argument roles of relations and types and the attributes used to describe those argument roles. Firstly note that the use of atoms to describe (atomic) relations/types means that the description of a relation/type contains no information about the argument roles of that relation/type. Only when the relation/type appears in an infon/proposition are the argument roles represented as part of the description of the infon/proposition. Likewise, in the descriptions of complex relations/types presented below, argument roles are not included. This deficiency in our descriptions does not cause any problems, but one should be aware of it. The second issue concerns the question of whether there is a (small) set of universal argument roles from which individual relations/type choose some subset, or whether individual relations/types have disjoint sets of argument roles. Our use of attributes as corresponding to argument roles in the descriptions of infons/propositions does not require us to make a stand on this ontological issue. We often use attributes such as AGENT and PATIENT

for argument roles, but these attributes need not be seen as being in one-one correspondence with argument roles. In particular, AGENT might correspond to the "runner" role of the relation *run* and the "chaser" role the relation *chase*. The principal advantage of taking attributes from a small set as corresponding to argument roles is that it allows the generalisations which those attributes encode to be efficiently represented in the constituent hierarchy. Given such a system, all intransitive verbs, for example, can be partially described by:

$$\begin{bmatrix} \text{SYN|LOC|SPEC} & \text{NP}\boxed{1} \\ \text{SEM|CONTENT|AGENT} & \boxed{1} \end{bmatrix}$$

where $\text{NP}\boxed{1}$ is an abbreviation for:

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{Noun} \end{bmatrix} \\ \text{SPEC} & \textit{null} \\ \text{COMPS} & \textit{null} \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{CONTENT} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

In HPSG, which employs relation specific attributes, such generalisations cannot be easily made, leading to redundancy in the lexical hierarchy.

1.4 Restricted Parameters

Restricted parameters are modelled as normal parameters (i.e., by variables) except that the CONTEXT attribute is used to encode the restriction. Recall that the value of the CONTEXT attribute is a set of parametric propositions which, when anchored by the speaker connections, must hold. Thus the value of the SEMANTICS attribute for a particular use of the proper name "Kim", corresponding to the parameter:

$$\dot{x} \mid S \models \langle\langle \text{naming, name: "Kim", named: } \dot{x}; + \rangle\rangle$$

where S is the resource situation associated with 'Kim' employed on the particular occasion of use, is:

$$\begin{bmatrix} \text{CONTENT} & \boxed{1} \\ \text{CONTEXT} & \left\{ \begin{bmatrix} \text{TYPE} & \textit{supports} \\ \text{SIT} & S \\ \text{INF} & \begin{bmatrix} \text{ISSUE} & \begin{bmatrix} \text{RELN} & \textit{naming} \\ \text{NAME} & \text{“}\textit{Kim}\text{”} \\ \text{NAMED} & \boxed{1} \end{bmatrix} \\ \text{POLARITY} & + \end{bmatrix} \end{bmatrix} \right\} \end{bmatrix}$$

Further restrictions may be applied to the parameter by adding further propositions to the CONTEXT set. This representation of restrictions bears a close resemblance to the use of *environments* by Robin Cooper ([Robin Cooper 90], [Robin Cooper ∞]). He argues on several grounds for the use of environments rather than restricted parameters, and the status of restricted parameters within situation theory currently appears to be changing.

It is worth noting once again that in CPSG restrictions on parameters are given as propositions, and not as infons. Thus context is taken to be a set of propositions, and not a set of infons. This allows more control over the precise restrictions which can be stated. In particular, all restrictions do not have to be stated with respect to a single resource situation.

1.5 Abstraction: Complex Relations and Complex Types

Complex relations and complex types are formed by abstraction over parametric infons and parametric propositions respectively. Such complex objects are required by our treatment of quantification and modification.

A complex relation or type consists of two parts, a body, consisting of a parametric infon or proposition respectively, and a set of abstracted parameters. If the infon/proposition contains parameters that are not abstracted over, then the resultant relation/type will be parametric, having as parameters those parameters in the infon/proposition not abstracted over. We thus model complex relations and types by feature structures with three attributes: BODY, ARG-ROLES and PARAMS. For a complex relation, the value of BODY describes a parametric infon, and for a complex type, the value of BODY describes a parametric proposition.

The complex relation:

$$[\{\dot{x}\ \dot{y}\} \mid \langle\langle \text{rel}, \text{arg}_1\colon \dot{x}, \text{arg}_2\colon \dot{y}, \text{arg}_3\colon \dot{z}, \text{arg}_4\colon \dot{x}; +\rangle\rangle]$$

is described by the feature structure:

$$\begin{bmatrix} \text{BODY} & \begin{bmatrix} \text{ISSUE} & \begin{bmatrix} \text{RELN} & \textit{rel} \\ \text{ARG}_1 & \boxed{1} \\ \text{ARG}_2 & \boxed{2} \\ \text{ARG}_3 & \boxed{3} \\ \text{ARG}_4 & \boxed{1} \end{bmatrix} \\ \text{POLARITY} & + \end{bmatrix} \\ \text{ARG-ROLES} & \{\boxed{1}, \boxed{2}\} \\ \text{PARAMS} & \{\boxed{3}\} \end{bmatrix}$$

where $\boxed{1}$ represents the parameter $\dot{x}$, $\boxed{2}$ represents $\dot{y}$ and $\boxed{3}$ represents $\dot{z}$.

Note that the argument roles of this relation are not explicitly modelled. This follows the case of atomic relations, whose argument roles are only described when the relation is part of an infon.

One serious problem with this, and it would seem any, treatment of abstraction via feature structure descriptions is that feature structures allow no way of binding the abstracted parameter. Our description of the property

$$[\dot{x} \mid \sigma(\dot{x})]$$

does not capture the fact that $\dot{x}$ is bound by the abstraction, and that any other occurrence of $\dot{x}$ should be treated as a different parameter. We avoid this problem by always choosing parameters which are otherwise unused.

1.6 Resource Situations

Many of the constraints on restricted parameters take the form of propositions requiring that some resource situation supports some infon. Different utterances, even if they are of the same words, generally exploit different resource situations. Furthermore, different parts of the same utterance exploit different parts of the resource situation associated with the whole utterance, or to put it another way, each part of an utterance exploits its own resource situation, and these resource situations are all part of a larger global

resource situation for the utterance. We assume that the resource situation of any constituent is a part of the resource situation of any larger constituent. That is, the resource situation of any N is a part of the resource situation of the dominating $\overline{\text{N}}$, and the resource situation of any $\overline{\text{V}}$ is part of the resource situation of the dominating $\overline{\overline{\text{V}}}$, etc..

Resource situations are expressed in terms of parameters which are mapped by speaker connections to the particular resource situation on the particular occasion of use. Thus the parameter corresponding to the resource situation must also be restricted such that it can only be mapped to the relevant resource situation. Given the circularity admitted by situation theory, there is no problem in giving this restriction in terms of the resource situation itself, so that, for a particular utterance of 'Tigger', the parameter $\dot{s}$ corresponding to the resource situation might be restricted by the proposition:

$$\dot{s} \models \langle\langle \text{resource-situation}, \dot{s}, \underline{\text{'Tigger'}}; + \rangle\rangle$$

where the underlining indicates a particular use.

Adopting this approach, resource situations are required to support infons about themselves: infons stating that they are resource situations for particular utterances.

Alternately, the restriction might be stated in terms of a type: $\dot{s}$ and the particular utterance must stand in the binary type *is-resource-situation-of*. That is, the relevant restriction might be given by the proposition:

$$((\dot{s}, \underline{\textit{'Tigger'}}) \ : \ \textit{is-resource-situation-of})$$

These solutions may be made equivalent by taking *is-resource-situation-of* to be a structurally determined type (see section 1 of Chapter 8).

2 Unification and Predication

Adopting a compositional approach to semantics we may view each subconstituent of a constituent as providing partial information about the semantics of the constituent as a whole. Within situation theory there are two distinct ways of talking about partial

information. Firstly there are parametric objects, such as infons some of whose argument roles are filled by restricted parameters. A second handle on partial information is given by thinking of properties, relations and types as partially specifying infons or propositions. Embedding situation theory in a feature-based formalism yields another way of talking about partial information: via uninstantiated values within the feature structure corresponding to the situation theoretic object. Each of these treatments of partial information yields its own mechanism for combining partial information. Firstly, all partial information might be treated in terms of restrictions on parameters (possibly with the parameters themselves being introduced by the head subconstituent), so that the result of combining this partial information is just a set of restrictions (or constraints) on the final object. This approach is somewhat counter to the usual use of feature-based systems. Secondly, the semantics of a constituent might be constructed from the semantics of its subconstituents via some sort of structure building, putting together a relation with its arguments to build an infon, for example. This approach we see as a generalisation of the usual notion of *predication*, where a function, representing partial information, is combined with its arguments to yield another object. Thirdly, we may use unification so that one subconstituent (presumably the head) may be taken to provide a frame for the semantics of the constituent with the semantics of the other subconstituents being unified into that frame.

To illustrate the last two options, consider the verb 'miaow', which might on the one hand be taken to express a unary relation, or on the other hand it might be taken to express an infon with uninstantiated arguments (or even a proposition with uninstantiated arguments). If it expresses just a relation, then putting that relation together with an argument is *predication*. If the verb expresses an infon/proposition with uninstantiated arguments, then putting that infon together with an argument involves only instantiating the relevant variable with the argument. This is *unification*.

Given these two mechanisms for combining partial information, the question of which (if not both) should be used in CPSG arises. It is difficult to argue either way, and what is really at issue is the type of objects which constituents denote. If verb phrases are taken to denote properties or types and proper names are taken to denote individuals, then predication is required to put these objects together to form an infon or proposi-

tion. CPSG takes verb phrases to denote parametric propositions, and does not employ predication, neither for filling the argument roles of verbs nor anywhere else. Instead, CPSG employs unification (which is specified in terms of shared structure) to build the semantics of constituents from their subconstituents. This is not to say that unification is sufficient for all natural language semantics. The need for predication may become apparent when larger fragments are considered, and as such it should not be ruled out entirely.

On a related issue, abstraction, which also involves structure building, is employed by CPSG in a number of places. These include the treatment of quantification, the treatment of coordination of adjectives and the treatment of relative clauses. Following [Barwise & Cooper 81], determiners are treated as denoting binary relations. These relations are taken to be relations between types, and thus, given our treatment of verb phrases as denoting parametric propositions, quantifier scoping requires the conversion of parametric propositions to types via abstraction. Abstraction is also required in the treatment of relative clauses. The semantic content of a relative clause is a parametric proposition, where the parameter corresponding to the relative pronoun is distinguished. Given our treatment of nouns as denoting types, attachment of a relative clause to a noun involves abstracting over the distinguished parameter in the semantic content of the relative clause to form a type, and forming a type corresponding to the conjunction of that type and the type contributed by the noun (which again involves abstraction: see section 5.3).

Given that CPSG employs abstraction without predication, abstraction can only be employed where relations or types can fill argument roles. Each of the above instances of abstraction yields a type which may fill an argument role of the relation denoted by the determiner.

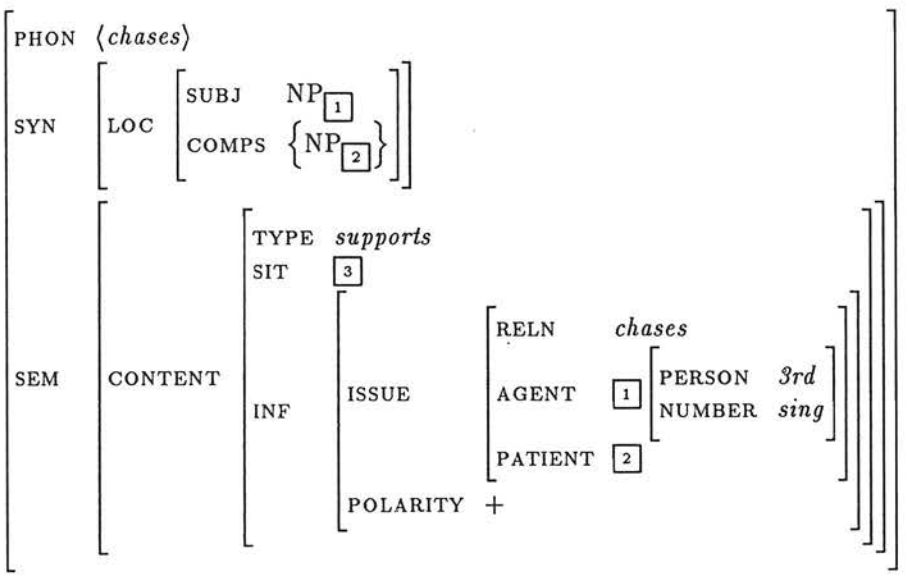
3 Lexical Types

The value of the SEMANTICS attribute of phrasal constituents is determined from the lexical subconstituents according to the partial descriptions of phrasal types. In this section we sketch the appropriate values for some types of lexical constituents.

3.1 Verbs

As discussed in the previous section, we adopt unification as the principal mechanism for combining semantic information. This unification is effected by structure sharing in the lexicon. We take verbs as denoting partially specified propositions, where the semantic contents of the subcategorised for arguments of the verb are identical with the uninstantiated structure of the proposition. When the subcategorisation requirements of the verb are met, the description of the proposition is made more complete due to the instantiation of the subcategorised arguments (see section 4). This is very similar to the approach of HPSG, except that verbs are taken to denote propositions rather than infons.

The finite verb 'chases' is partially described by:



Recall that subscripted tags indicate the value of the SEMANTIC|CONTENT of the element they subscript. Thus $NP_{[1]}$ abbreviates a noun phrase which bears the feature specification [SEMANTICS|CONTENT [1]]. Given a description like this, when the verb combines with its complements to form a head/complement phrase, the SEMANTIC|CONTENT of each complement will be unified with the appropriate argument role of the verb, by virtue of the structure sharing in the lexical entry for the verb. Similarly when the verb combines with its subject, the SEMANTIC|CONTENT of the subject will be unified with the argument role of the verb which corresponds to the subject (in this case the agent of the embedded infon). Note that in this respect subjects and complements are treated identically.

The value of the CONTEXT attribute must reflect the tense of the verb (see section 7) and the fact that [3] must be anchored to the described situation. Furthermore, features on the agent parameter [1] are included to reflect the agreement properties of the verb: in this case [1] must bear the features associated with third person singular.

In the case of control verbs, which subcategorise for unsaturated complements, the situation is slightly more complex, though again the CPSG treatment mostly follows that of HPSG. A partial feature structure description of (one form of) the intransitive raising verb 'appear' is:

$$\left[\begin{array}{ll}
\text{PHON} & \langle \textit{appears} \rangle \\
\text{SYN} & \left[\begin{array}{ll}
\text{LOC} & \left[\begin{array}{ll}
\text{HEAD} & \left[\begin{array}{ll} \text{MAJ} & \textit{Verb} \\ \text{FORM} & \textit{fin} \\ \text{AUX} & - \\ \text{INV} & - \end{array}\right] \\
\text{SPEC} & \boxed{1}\,\text{NP[nom]} \\
\text{COMPS} & \left\{\text{VP}[\textit{inf}, \text{SPEC } \boxed{1}]\boxed{2}\right\}
\end{array}\right]
\end{array}\right] \\
\text{SEM} & \left[\begin{array}{ll}
\text{CONTENT} & \left[\begin{array}{ll}
\text{TYPE} & \textit{supports} \\
\text{SIT} & \boxed{3} \\
\text{INF} & \left[\begin{array}{ll}
\text{ISSUE} & \left[\begin{array}{ll} \text{RELN} & \textit{appear} \\ \text{THEME} & \boxed{2} \end{array}\right] \\
\text{POLARITY} & + \end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]$$

The semantic content of the embedded verb phrase is shared with the THEME of 'appear', with the subject of the embedded verb phrase being shared with the subject of 'appear', ensuring the correct semantic binding.

The semantics of auxiliary verbs, which HPSG also treats as control verbs, are discussed in detail in section 7.

3.2 Saturated Nouns

The semantic content of lexically saturated nouns is taken to be a restricted parameter, with the type of restriction depending on the type of noun. The anchor given by the speaker connections must map this parameter to an individual which satisfies all restrictions.

3.2.1 Proper Names

Rather than assuming that a name names a unique object, as is assumed by most other semantic theories, situation semantics attempts to capture the fact that different things can have the same name, and that with different resource situations a given name will refer to a different object. The semantic content of a proper name is therefore taken to be a parameter (rather than an individual), which is restricted such that it can only be anchored to an object which is named by the required name in the relevant resource situation. With different speaker connections, different objects may be anchored to the parameter, capturing the fact that the same name may refer to different individuals in different contexts.

Under this conception of names, the situation described by an utterance of 'Tigger chased Fido', S_d, must be such that (under a suitable anchoring a for $\dot{x}$ and $\dot{y}$):

$$S_d \models \langle\langle \text{chase, chaser: } \dot{x}, \text{ chased: } \dot{y};\ + \rangle\rangle[a]$$

The restrictions on the parameters $\dot{x}$ and $\dot{y}$ are stated in resource situations, S_r and $S_{r'}$, so we also require that:

$$S_r \models \langle\langle \text{naming, name: "Tigger", named: } \dot{x};\ + \rangle\rangle[a]$$

$$S_{r'} \models \langle\langle \text{naming, name: "Fido", named: } \dot{y};\ + \rangle\rangle[a]$$

The arrangement of semantic attributes in CPSG leads to a fairly straightforward treatment of proper names. 'Tigger', for example, is partially described by:

$$\begin{bmatrix} \text{PHON} & \langle \textit{Tigger} \rangle \\ \text{SEM} & \begin{bmatrix} \text{CONTENT} & \boxed{1} \\ \text{CONTEXT} & \left\{ \begin{bmatrix} \text{TYPE} & \textit{supports} \\ \text{SIT} & \boxed{2} \\ \text{INF} & \begin{bmatrix} \text{ISSUE} & \begin{bmatrix} \text{RELN} & \textit{naming} \\ \text{NAME} & \text{"Tigger"} \\ \text{NAMED} & \boxed{1} \end{bmatrix} \\ \text{POLARITY} & + \end{bmatrix} \end{bmatrix}, \ldots \right\} \\ \text{PARAMETERS} & \{\boxed{1}, \boxed{2}\} \end{bmatrix} \end{bmatrix}$$

Not shown here is the restriction requiring the parameter [2] to be anchored to the relevant resource situation.

Note that this is vastly simpler than the treatment of proper names in HPSG (see section 1.2), primarily because of the simplifications made in CPSG's treatment of parameters.

3.2.2 Pronouns

Pronouns are treated almost exactly as in HPSG. The only difference stems from the treatment of parameters. The pronoun 'she' is partially described by:

$$\begin{bmatrix} \text{PHON} & \langle she \rangle \\ \text{SEM} & \begin{bmatrix} \text{CONTENT} & \boxed{1}\begin{bmatrix} \text{PERSON} & 3rd \\ \text{NUMBER} & singular \\ \text{GENDER} & feminine \end{bmatrix} \\ \text{CONTEXT} & \{\,\} \\ \text{PARAMETERS} & \{\boxed{1}\} \end{bmatrix} \end{bmatrix}$$

3.2.3 Expletives

Expletives differ from other saturated lexical nouns in that they do not refer to an individual. In 'it rains', for example, the 'it' does not refer to an individual: *rains* is a 0-place relation (ignoring tense). Similarly, in *it appears that Tigger is miaowing*, *it* does not refer and *appears* is a 1-place relation whose sole argument role is filled by the proposition denoted by *Tigger is miaowing*. Similar comments hold for the expletive 'there'.

Verbs which subcategorise for expletives do not refer to the semantic attributes of that expletive, and it is therefore irrelevant what we take the semantics of such elements to be. Given the general principles employed in building the semantics of head/argument phrases, we take expletives to be partially described by:

$$\begin{bmatrix} \text{SEMANTICS} & \begin{bmatrix} \text{CONTENT} & null \\ \text{CONTEXT} & \{\,\} \\ \text{PARAMETERS} & \{\,\} \end{bmatrix} \end{bmatrix}$$

In recent work ([Pollard 89b], [Pollard & Sag ∞a]), Pollard has suggested that expletives should be treated wholly semantically, and not in terms of the syntactic NFORM attribute.

3.3 Unsaturated Nouns

The value of the SEMANTICS|CONTENT attribute for unsaturated nouns is a type, the property denoted by the noun with respect to the relevant resource situation. Thus the CONTENT for the noun 'kitten' is the property of kittenhood with respect to a resource situation. A partial description of 'kitten' is:

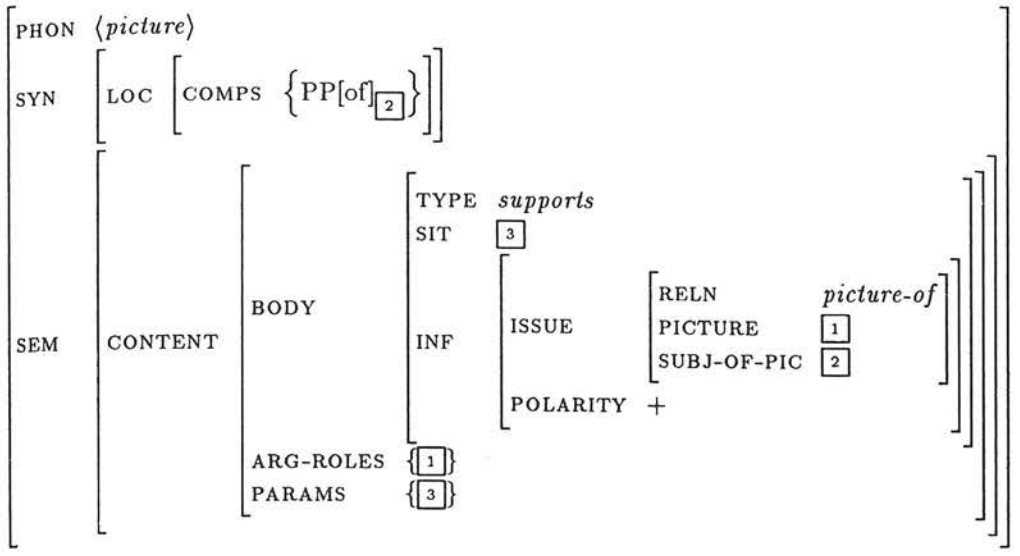
$$\left[\begin{array}{ll}
\text{PHON} & \langle \textit{kitten} \rangle \\
\text{SYN} & \left[\text{LOC}\left[\text{COMPS}\ \{\,\}\right]\right] \\
\text{SEM} & \left[\text{CONTENT}\left[\begin{array}{ll}
\text{BODY} & \left[\begin{array}{ll}
\text{TYPE} & \textit{supports} \\
\text{SIT} & \boxed{2} \\
\text{INF} & \left[\begin{array}{ll}
\text{ISSUE} & \left[\begin{array}{ll}\text{RELN} & \textit{kitten} \\ \text{AGENT} & \boxed{1}\end{array}\right] \\
\text{POLARITY} & +
\end{array}\right]
\end{array}\right] \\
\text{ARG-ROLES} & \{\boxed{1}\} \\
\text{PARAMS} & \{\boxed{2}\}
\end{array}\right]\right]
\end{array}\right]$$

The value of the SEMANTICS|CONTEXT attribute contains a restriction on the parameter described by $\boxed{2}$ such that the parameter may only be anchored to the relevant resource situation. The semantic content of the description represents the type:

$$[\dot{x} \mid \dot{s}_r \models \langle\langle \text{kitten}, \dot{x}; + \rangle\rangle]$$

where $\dot{s}_r$ is a parameter restricted such that it must be anchored to the resource situation on which the noun draws on the particular occasion of use.

In the case of "picture" nouns, the type is only partially specified, with the semantic content of the missing complement completing the specification. To be precise, 'picture' is partially described by:



The semantic content of this represents the partially specified type:

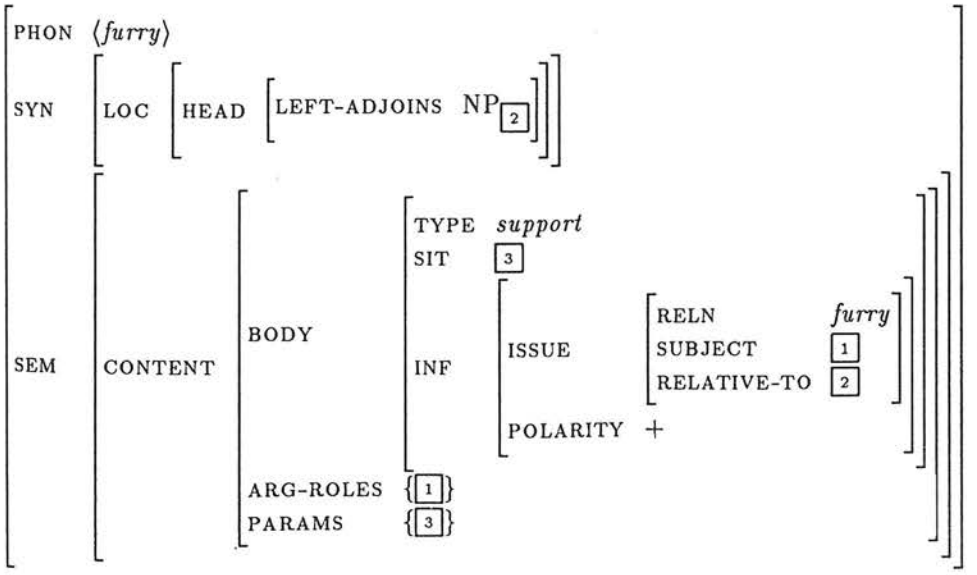
$$[\dot{x} \mid S_\tau \models \langle\langle \text{picture-of}, \dot{x}, y; + \rangle\rangle]$$

where y is identical to the semantic content of the subcategorised for complement.

We leave until Chapter 8 a discussion of how the semantics of determiners and unsaturated nouns interact.

3.4 Adjectives

An adjective combines with an $\overline{\text{N}}$ to yield an $\overline{\text{N}}$. As discussed in section 4 and Chapter 8, the semantic content of all $\overline{\text{N}}$s is a type. Adjectives must thus effectively map types to types. A partial description of 'furry' is:



The semantic content of this represents the partially specified type:

$$[\dot{x} \mid \dot{s} \models \langle\langle \text{furry}, \dot{x}, y; + \rangle\rangle]$$

where y is identical to the semantic content of the head which the adjective modifies. The semantic content of 'furry kitten' is thus:

$$[\dot{x} \mid \dot{s} \models \langle\langle \text{furry}, \dot{x}, [\dot{z} \mid \dot{s}' \models \langle\langle \text{kitten}, \dot{z}; + \rangle\rangle]; + \rangle\rangle]$$

All adjectives are taken to correspond to types which are relative to the noun they modify. If desired, situations may be taken to be closed under meaning postulates to capture the usual consequences of adjectival modification: a fake gun is not a gun, but a furry kitten is a kitten, etc..

3.5 Adverbs

Syntactically, adverbs may modify $\overline{\text{V}}$s or $\overline{\overline{\text{V}}}$s. Semantically, each of these corresponds to a proposition. Adverbs may thus be treated as denoting partial propositions consisting of a unary type. A partial description of 'loudly' is:

$$\begin{bmatrix} \text{PHON} & \langle \textit{loudly} \rangle \\ \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{LEFT-ADJOINS} & \text{V}\boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{CONTENT} & \begin{bmatrix} \text{TYPE} & \textit{loudly} \\ \text{PROP} & \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The semantic content of this represents the partially specified proposition:

$$(y \ : \ \textit{loudly})$$

where y is identical to the semantic content of the head which the adverb modifies.

4 Head/Argument Phrases

The value of the semantics attribute of phrases of type head/argument phrase is determined from the values of the semantics attributes of the phrase's daughters in much

the same way as in HPSG. Ignoring issues of quantifier scope, the Semantics Principle of HPSG may be stated as:

$$\left[\text{DTRS } \textit{headed-structure}\left[\ \right]\right] \Rightarrow \left[\begin{array}{l}\text{SEM} \left[\begin{array}{l}\text{CONTENT } \boxed{1} \\ \text{CONTEXT collect-indices}(\boxed{2})\end{array}\right] \\ \text{DTRS } \boxed{2}\left[\text{HEAD-DTR } \left[\text{SEM } \left[\text{CONTENT } \boxed{1}\right]\right]\right]\end{array}\right]$$

This, combined with the semantic linking in the lexicon discussed in the previous section, ensures that the semantic content of, for example, a verb phrase, is the semantic content of the head verb, which, in forming the verb phrase will have had its complement arguments instantiated. The principle also ensures that all context indices (i.e., parameters) are inherited.

The use of a hierarchical classificatory system in CPSG allows this principle to be stated as part of the definition of the type **headed phrase** in much the same way as HPSG's Head Feature Principle was stated in section 3.3 of Chapter 2. However, in extending the coverage to include further subtypes of **headed phrase**, it is difficult to see how this principle can be maintained for all of those subtypes. In particular, the principle is not suited for phrases of type **head/adjunct phrase**: the semantic content of such phrases is not the semantic content of the head. We thus effectively restrict the applicability of the principle to head/argument phrases by stating it as part of the feature structure description of the type **head/argument phrase**:

$$\Delta(\textbf{head/argument phrase}) \sqsupseteq \left[\begin{array}{l}\text{SEM} \left[\begin{array}{l}\text{CONTENT } \boxed{1} \\ \text{CONTEXT } \boxed{3} \\ \text{PARAMETERS } \boxed{4}\end{array}\right] \\ \text{DTRS } \boxed{2}\left[\text{HEAD-DTR } \left[\text{SEM } \left[\text{CONTENT } \boxed{1}\right]\right]\right]\end{array}\right]$$

where *collect-context*(2,3)

and *collect-parameters*(2,4)

The generalisation over head argument phrases captured by this specification motivates the type **head/argument phrase**, subsuming **head/complement phrase** and **head/topic phrase**, in the constituent hierarchy. This generalisation is revised slightly in section 4 of Chapter 8, where quantifier scoping is considered.

Note then that the semantics of head/complement phrases, head/specifier phrases and head/filler phrases are all specified in terms of their subconstituents according to this one specification. In particular, nothing special needs to be said about head/filler phrases — elements in the SLASH set have their semantic content linked to the appropriate argument role of the verb just as other non-extracted arguments.

5 Head/Adjunct Phrases

5.1 Adjectives and Adverbs

Given the lexical signs for adjectives and adverbs in section 3, the semantic content of a head/adjunct phrase is just the semantic content of the adjunct, with the context and parameters being the set union of the context and parameters of the subconstituents:

$$\Delta(\text{head/adjunct phrase}) \sqsupseteq \left[\begin{array}{ll} \text{SEM} & \left[\begin{array}{ll} \text{CONTENT} & \boxed{1} \\ \text{CONTEXT} & \boxed{2} \\ \text{PARAMETERS} & \boxed{3} \end{array}\right] \\ \text{DTRS} & \left[\begin{array}{ll} \text{HEAD-DTR} & \left[\text{SEM} \left[\begin{array}{ll} \text{CONTEXT} & \boxed{4} \\ \text{PARAMETERS} & \boxed{5} \end{array}\right]\right] \\ \text{ADJUNCT-DTR} & \left[\text{SEM} \left[\begin{array}{ll} \text{CONTENT} & \boxed{1} \\ \text{CONTEXT} & \boxed{6} \\ \text{PARAMETERS} & \boxed{7} \end{array}\right]\right] \end{array}\right] \end{array}\right]$$

where $\boxed{2} = \boxed{4} \cup \boxed{6}$

and $\boxed{3} = \boxed{5} \cup \boxed{7}$

To illustrate, consider the case of 'furry kitten'. For convenience we use situation theoretic notation rather than feature structure descriptions, and do not include the details of resource situations. For 'furry', we have:

Content:	$[\dot{y} \mid S_r \models \langle\langle \text{furry}, \dot{y}, x; + \rangle\rangle]$
Context:	$\{\}$
Parameters:	$\{\}$

where x is the semantic content of the noun that 'furry' modifies.

For 'kitten', we have:

Content: $[\dot{z} \mid S_r \models \langle\langle \text{kitten}, \dot{z}; + \rangle\rangle]$

Context: $\{\}$

Parameters: $\{\}$

Instantiating x in the content of 'furry' with the content of 'kitten', together with the necessary percolation of context and parameters, yields as the semantics for 'furry kitten':

Content: $[\dot{y} \mid S_r \models \langle\langle \text{furry}, \dot{y}, [\dot{z} \mid S_r \models \langle\langle \text{kitten}, \dot{z}; + \rangle\rangle]; + \rangle\rangle]$

Context: $\{\}$

Parameters: $\{\}$

5.2 Prepositional Phrases I: Verb Phrase Modifiers

Locative prepositional phrases which act as verb phrase modifiers are normally taken to express a restriction on the location argument of the infon corresponding to the verb phrase which is modified (see, for example, [Colban & Fenstad 87]). Our treatment is slightly different from this, involving a binary type for each preposition.

As an illustrative example, the prepositional phrase 'in Rome' has semantics:

Content: $(p, \dot{r}) : in$

Context: $\{S_r \models \langle\langle \text{named}, \dot{r}, \text{Rome}; + \rangle\rangle\}$

Parameters: $\{\dot{r}\}$

where p is the semantic content of the verb phrase which the prepositional phrase modifies. Thus for 'Tigger miaows in Rome', we have:

Content: $(S_d \models \langle\langle \text{miaows}, \dot{t}; + \rangle\rangle, \dot{r}) : in$

Context: $\left\{ \begin{array}{l} S_r \models \langle\langle \text{named}, \dot{r}, \text{Rome}; + \rangle\rangle \\ S_r \models \langle\langle \text{named}, \dot{t}, \text{Tigger}; + \rangle\rangle \end{array} \right\}$

Parameters: $\{\dot{r}, \dot{t}\}$

The reason why we adopt this approach rather than the more traditional one stems from our use of propositions rather than infons as the semantic content of verb phrases. The approaches are related by the structurally determined definition of the types associated with prepositions. We assume a function *loc* mapping propositions to associated locations. In the simple case of a proposition of the form $S \models \sigma$, the location of the proposition is just the value of the location argument of σ. This function is also used

in our treatment of tense and aspect in section 7, where its value for more complex propositions is defined by cases.

Assuming space-time is structured with some sort of inclusion relation $\subset$, a proposition P and a location l are of type *in* iff $loc(P) \subset l$. Similar types may be employed for other prepositions.

5.3 Restrictive Relative Clauses

Whilst we have left open the question of just how relative clauses are syntactically attached to nouns, given our treatment of nouns as denoting types, the semantics of restrictive relative clauses is fairly straightforward. The description of the relative clause will include:

$$\left[\begin{array}{ll} \text{SYN} & \left[\text{BIND}\ \left[\text{REL}\ \left[\text{ALL}\ \left\{\text{NP}_{\boxed{1}}\right\}\right]\right]\right] \\ \text{SEM} & \left[\begin{array}{ll} \text{CONTENT} & \boxed{2} \\ \text{CONTEXT} & \ldots \\ \text{PARAMETERS} & \ldots \end{array}\right] \end{array}\right]$$

The element on the REL|ALL set corresponds to the relative pronoun in the relative clause. [2] is a parametric proposition containing the parameter [1] corresponding to the relative pronoun. If the semantic content of the noun is the type [3], then the semantic content of the noun/relative clause complex will be the type

$$[\boxed{1} \mid (\{(\boxed{1} : \boxed{3}), \boxed{2}\} : \mathit{and})]$$

i.e., the type formed by first forming the conjunction of the propositions $(\boxed{1} : \boxed{3})$ and [2], and then abstracting over the parameter [1], which occurs in each of the conjoined propositions.

As an example consider the relative clause 'which miaows', modifying the noun 'kitten'. The relative clause will be analysed syntactically as a sentence with a singleton set as the value of its SYN|BIND|REL|ALL attribute. The content of the sentence will correspond to the parametric proposition:

$$S \models \langle\langle \text{miaows}, \dot{x}; + \rangle\rangle$$

where the content of the single element in the SYN|BIND|REL|ALL set corresponds to $\dot{x}$. Furthermore, the content of the noun 'kitten' will be the object type:

$$[\dot{y} \mid S' \models \langle\langle \text{kitten}, \dot{y}; + \rangle\rangle]$$

Following the above the content of the modified noun is the complex type:

$$\left[\dot{x} \;\middle|\; \left(\left\{ \begin{array}{c} (\dot{x} : [\dot{y} \mid S' \models \langle\langle \text{kitten}, \dot{y}; + \rangle\rangle]) \\ S \models \langle\langle \text{miaows}, \dot{x}; + \rangle\rangle \end{array} \right\} : \mathit{and} \right) \right]$$

Note that in building this semantic content we must abstract over the parameter $\boxed{1}$. If we take nouns to denote types, as we do, then it does not seem possible to avoid this abstraction

Note also that the lexical entry for a relative pronoun such as 'who' might include in its semantic context a proposition restricting the parameter it denotes to an individual which is human. Upon abstraction of the parameter, this restriction remains, becoming an appropriateness condition on the corresponding argument role (see [Gawron & Peters 90b]).

5.4 Prepositional Phrases II: Noun Modifiers

The semantic treatment of noun modifying prepositional phrases is similar to that of relative clauses. The semantics of 'in Rome' is:

Content: $$\left[\dot{x} \;\middle|\; \left(\left\{ \begin{array}{c} (\dot{x}, \dot{r}) : \mathit{in} \\ \dot{x} : \boxed{1} \end{array} \right\} : \mathit{and} \right) \right]$$

Context: $S_r \models \langle\langle \text{named}, \dot{r}, \text{Rome}; + \rangle\rangle$

Parameters: $\{\dot{r}\}$

where $\boxed{1}$ is the semantic content of the head noun being modified.

Note that this treatment requires different lexical entries for prepositions acting as noun modifiers and prepositions acting as verb phrase modifiers. Further lexical entries are required for prepositions acting as noun phrase case markers.

6 Coordination

As described in Chapter 5, coordination is treated syntactically in terms of head/conjunction phrases, which in English consist of a head preceded by a conjunction,

and coordinate phrases, which consist of two or more conjuncts, at least one of which is a head/conjunction phrase.

6.1 Head/Conjunction Phrases

Head/conjunction phrases consist of a head marked with a conjunction. The conjunction is treated as a syntactic marker and does not alter the semantics of the phrase — only when forming coordinate phrases from head/conjunction phrases does the marking on the daughters determine whether the semantics of the whole is a conjunction or a disjunction. All semantic attributes are therefore inherited by the phrase from its head:

$$\Delta(\text{head/conjunction phrase}) \sqsupseteq \begin{bmatrix} \text{SEM} & \boxed{1} \\ \text{DTRS} & \left[\text{HEAD-DTR} \left[\text{SEM} \boxed{1}\right]\right] \end{bmatrix}$$

6.2 Coordinate Phrases

Two principal issues arise in specifying the semantics of coordinate phrases in terms of their subconstituents. The semantic content of the phrase as a whole must be specified in terms of its parts, and if the conjuncts are unsaturated, unfilled argument roles must be linked appropriately, so that in, for example, 'Tigger miaows and chases Fido', the semantic content of 'Tigger' is both the agent of 'miaows' and the agent of 'chases Fido'.

6.2.1 Coordination and Compositionality

Verb Phrase Coordination

The semantics of verb, verb phrase and sentence coordination is relatively straightforward. In each case the semantic content of each of the conjuncts is a proposition, ϕ_i, and the semantic content of the coordinated phrase is just the proposition

$$(\{\phi_i \mid \phi_i \text{ is a conjunct}\} : T)$$

where T is the relevant unary type. T might be the type *and*, in which case the proposition is true iff every element in the set is true, the type *or*, in which case the proposition is true iff at least one element in the set is true, the type *xor*, in which case the proposition is true iff exactly one element in the set is true, or the type *nor*, in which case the proposition is true iff no element in the set is true.

Thus, recalling the subtypes of coordinate phrase from Chapter 5, we have, for example:

$$\Delta(\text{neither-nor}) \sqsupseteq \left[\begin{array}{ll} \text{SEM} & \left[\begin{array}{ll} \text{CONTENT} & \left[\begin{array}{ll} \text{TYPE} & \textit{nor} \\ \text{PROPS} & \boxed{1} \end{array}\right] \\ \text{CONTEXT} & \boxed{2} \\ \text{PARAMETERS} & \boxed{3} \end{array}\right] \\ \text{DTRS} & \left[\text{COORD-DTRS} \quad \boxed{4}\right] \end{array}\right]$$

where *collect-contents*(4,1)

and *collect-context*(4,2)

and *collect-parameters*(4,3)

The relation *collect-contents/2* holds of a set of constituent descriptions and a set of feature structures if the second argument is the set of the values of SEMANTICS|CONTENT attributes of the elements of the first. The relation *collect-context/2* holds of a set of constituent descriptions and a set of feature structures if the second argument is the set union of the values of SEMANTICS|CONTEXT attributes of the elements of the first. The relation *collect-parameters/2* holds of a set of constituent descriptions and a set of uninstantiated variables if the second argument is the set union of the values of SEMANTICS|PARAMETERS attributes of the elements of the first.

Noun Phrase Coordination

The situation is far more complex in the case of noun phrase coordination. Consider the examples in (1):

(1) a. Either Tigger or Fido miaows.

b. Neither Tigger nor Fido miaows.

c. Both Tigger and Fido miaow.

One possibility for the semantics of (1a) is:

Content: $S_d \models \langle\langle \text{miaows}, \dot{x}; + \rangle\rangle$

Context: $\left\{ \left(\left\{ \begin{array}{l} S_r \models \langle\langle \text{named}, \dot{x}, \text{Tigger}; + \rangle\rangle \\ S_r \models \langle\langle \text{named}, \dot{x}, \text{Fido}; + \rangle\rangle \end{array} \right\} : \textit{or} \right) \right\}$

Parameters: $\{\dot{x}\}$

That is, something miaows, and that something is named Tigger or named Fido. One could imagine building this semantics compositionally by taking the semantics of the noun phrase to be a parameter with a disjunctive restriction. This is not quite right, however. In particular note that reference is only made to one parameter. What we really require is:

Content: $\left\{\left(\left\{\begin{array}{l} S_d \models \langle\langle \text{miaows}, \dot{x}; + \rangle\rangle \\ S_d \models \langle\langle \text{miaows}, \dot{y}; + \rangle\rangle \end{array}\right\} : or\right)\right\}$

Context: $\left\{\begin{array}{l} S_r \models \langle\langle \text{named}, \dot{x}, \text{Tigger}; + \rangle\rangle \\ S_r \models \langle\langle \text{named}, \dot{y}, \text{Fido}; + \rangle\rangle \end{array}\right\}$

Parameters: $\{\dot{x}, \dot{y}\}$

It is more difficult to see how this semantics could be built compositionally from the semantics of the subconstituents. In particular, the noun phrase disjunction is cashed out in terms of disjunction of propositions in the described situation.

Example (1b) provides an even more persuasive argument for the second approach. If we were to adopt the first approach above, replacing *or* with *nor*, we would have:

Content: $S_d \models \langle\langle \text{miaows}, \dot{x}; + \rangle\rangle$

Context: $\left\{\left(\left\{\begin{array}{l} S_r \models \langle\langle \text{named}, \dot{x}, \text{Tigger}; + \rangle\rangle \\ S_r \models \langle\langle \text{named}, \dot{x}, \text{Fido}; + \rangle\rangle \end{array}\right\} : nor\right)\right\}$

Parameters: $\{\dot{x}\}$

This is clearly not a true reflection of the semantics of (1b). According to this, (1b) could be paraphrased as "something miaows and that something is neither Tigger nor Fido". Again, it is the second semantics which is correct:

Content: $\left\{\left(\left\{\begin{array}{l} S_d \models \langle\langle \text{miaows}, \dot{x}; + \rangle\rangle \\ S_d \models \langle\langle \text{miaows}, \dot{y}; + \rangle\rangle \end{array}\right\} : nor\right)\right\}$

Context: $\left\{\begin{array}{l} S_r \models \langle\langle \text{named}, \dot{x}, \text{Tigger}; + \rangle\rangle \\ S_r \models \langle\langle \text{named}, \dot{y}, \text{Fido}; + \rangle\rangle \end{array}\right\}$

Parameters: $\{\dot{x}, \dot{y}\}$

Given the structurally determined definition of the type *nor* above, this means that "it is not the case that Tigger miaows, and nor is it the case that Fido miaows", which is a true paraphrase of (1b).

These observations also apply to (1c), where the first approach yields a semantics corresponding to "there is something that miaows and that something is named both Tigger and Fido", and the second approach yields a semantics corresponding to "Tigger miaows and Fido miaows". In functional approaches, this kind of coordination might be treated via "type-raising": making noun phrases denote functions from predicates to propositions. If we take the noun phrase 'Tigger' to denote the function $\lambda P.P(t)$ and 'Fido' to denote the function $\lambda P.P(f)$, then 'Tigger or Fido' may be denoted by $\lambda P.(P(t) \vee P(f))$. Functional treatments of quantification also generally take noun phrases to denote such functions. As such, there are clear similarities between coordinated noun phrases and quantified noun phrases, and we therefore defer discussion of the semantics of coordinated noun phrases until section 5 of Chapter 8.

The case of noun phrase coordination involving 'and' is complicated by *non-distributive* readings: 'Tigger and Fido carry a piano' has two readings, one as above where the conjuncts distribute over the verb phrase, which might be paraphrased as "Tigger carries a piano, as does Fido", and one where "Tigger and Fido" corresponds to a composite individual — there is one piano and Tigger and Fido cooperate in carrying it. Such non-distributive readings are only available in the case of 'and', and are not possible for 'or' or 'nor'.

As discussed in section 2.3 of chapter 6, the situation theoretic domain of individuals may be structured by grouping operations. Non-distributive readings of conjoined noun phrases point out one such grouping operation: any subset of individuals can be considered as one composite individual (see [Link 83]). A non-distributive reading of 'Tigger and Fido' might have the following semantics:

$$\begin{array}{ll} \text{Content:} & \dot{z} \\ \text{Context:} & \left\{ \begin{array}{c} S_r \models \langle\langle \text{named}, \dot{x}, \text{Tigger}; + \rangle\rangle \\ S_r \models \langle\langle \text{named}, \dot{y}, \text{Fido}; + \rangle\rangle \\ (\dot{z}, \{\dot{x}, \dot{y}\} : \textit{consists-of}) \end{array} \right\} \\ \text{Parameters:} & \{\dot{x}, \dot{y}, \dot{z}\} \end{array}$$

where *consists-of* is a binary type which holds of a composite individual and a set of individuals iff the composite individual consists of exactly those individuals in the set. Such a semantics could easily be generated compositionally via a suitable rule from the

semantics of 'Tigger' and the semantics of 'Fido'.

Yet another semantic distinction in noun phrase coordination arises with examples such as 'Tigger and Fido quarrelled', where the coordinate phrase identifies a set, all of whose elements have a certain property, in this case the property of quarrelling with each other. We do not consider here how such readings may be generated.

Adjective and Adverb Coordination

In the cases of adjective and adverb coordination it is fairly clear what we want the semantic content of the coordinated phrases to be. Give two adjectives, 'dark' and 'furry', with semantic contents:

$$[\dot{x} \mid \dot{s} \models \langle\!\langle \text{dark}, \dot{x}, \dot{y}; + \rangle\!\rangle]$$

and

$$[\dot{x} \mid \dot{s} \models \langle\!\langle \text{furry}, \dot{x}, \dot{y}; + \rangle\!\rangle]$$

respectively, the required semantic content of 'dark and furry' is:

$$\left[\dot{x} \left| \left(\left\{ \begin{array}{l} [\dot{x} \mid \dot{s} \models \langle\!\langle \text{dark}, \dot{x}, \dot{y}; + \rangle\!\rangle] \\ [\dot{x} \mid \dot{s} \models \langle\!\langle \text{furry}, \dot{x}, \dot{y}; + \rangle\!\rangle] \end{array} \right\} : \textit{and} \right) \right.\right]$$

Similarly, for two adverbs 'loudly' and 'fiercely', with semantic contents:

$$(\dot{p} : \textit{loudly})$$

and

$$(\dot{p} : \textit{fiercely})$$

respectively, the required semantic content of 'loudly and fiercely' is:

$$\left(\left\{ \begin{array}{l} (\dot{p} : \textit{loudly}) \\ (\dot{p} : \textit{fiercely}) \end{array} \right\} : \textit{and} \right)$$

In each case the parameters involved are just the union of the parameters involved in the conjuncts, as are the restricting propositions of the CONTEXT attribute. The content, however, is clearly dependent on not just the contents of the conjuncts and the type of coordination, but also on the major syntactic category of each conjunct. In general, then,

the semantic content of a coordinate phrase is determined via a relational dependency involving the coordinate daughters (and specifically the semantic content of each such daughter), the coordinate type, the major syntactic category of the coordinate phrase, and the semantic content of the coordinate phrase. This and the previous subsections outline specific cases of that relational dependency.

6.2.2 Coordination and Subcategorisation

According to the feature structure descriptions of the subtypes of coordinate phrase, the subcategorisation requirements of each conjunct in a coordinate structure are identical. This is ensured by structure sharing of the values of all SPEC attributes, structure sharing of the values of all COMPS attributes, and structure sharing of the values of all BINDING attributes. The subcategorisation requirements of a coordinate phrase are given by these three groups of shared values. Structure sharing ensures that each conjunct requires compatible arguments and also effects binding of parameters involved in subcategorised for signs. For example, the relevant attributes for 'miaows' and 'and chases Fido' are:

$$\begin{bmatrix}
\text{PHON} & \langle \textit{miaows} \rangle \\
\text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{SPEC} & \text{NP}\boxed{1} \\ \text{COMPS} & \textit{null} \end{bmatrix} \end{bmatrix} \\
\text{SEM} & \begin{bmatrix} \text{CONTENT} & \begin{bmatrix} \text{TYPE} & \textit{supports} \\ \text{SIT} & S \\ \text{INF} & \begin{bmatrix} \text{ISSUE} & \begin{bmatrix} \text{RELN} & \textit{miaows} \\ \text{AGENT} & \boxed{1} \end{bmatrix} \\ \text{POLARITY} & + \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}
\end{bmatrix}$$

$$\begin{bmatrix}
\text{PHON} & \langle \textit{and chases Fido} \rangle \\
\text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{SPEC} & \text{NP}\boxed{2} \\ \text{COMPS} & \textit{null} \end{bmatrix} \end{bmatrix} \\
\text{SEM} & \begin{bmatrix} \text{CONTENT} & \begin{bmatrix} \text{TYPE} & \textit{supports} \\ \text{SIT} & S \\ \text{INF} & \begin{bmatrix} \text{ISSUE} & \begin{bmatrix} \text{RELN} & \textit{chases} \\ \text{AGENT} & \boxed{2} \\ \text{PATIENT} & \boxed{3} \end{bmatrix} \\ \text{POLARITY} & + \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}
\end{bmatrix}$$

where [3] is the parameter contributed by 'Fido'. When these constituents occur as conjuncts in a coordinate phrase, $NP_{[1]}$ and $NP_{[2]}$ are required to be identical, ensuring that [1] and [2] are identical and that the same individual fills the agent argument role of each embedded relation. The sharing of parameters corresponding to complements and extracted elements is analogous.

7 Tense, Aspect and Auxiliaries

In section 3, the value of the SEMANTICS attribute of verbs was discussed. That discussion included some comments on the treatment of control verbs, with the intransitive raising verb 'appears' being used to illustrate the treatment of the semantics of such verbs. Auxiliary verbs, which have similar subcategorisation requirements to intransitive raising verbs, may be treated similarly. There is much more that can be said about the semantics of auxiliaries, however, and in this section, as an extended example of the semantic component of CPSG, we provide a discussion of the tense and aspect system of English and the role auxiliaries play in the conveying of tense and aspect. Much of this material is orthogonal to the other material in this chapter, concerning tense and aspect in general. However, some discussion of this is necessary to provide the background necessary for the CPSG treatment.

7.1 A Reichenbachian Approach to Tense and Aspect

Following [Reichenbach 47] tense and aspect may be analysed in terms of three times: an event time (E), a speech time (S) and a reference time (R). Whilst two times are adequate for simple tenses, as in (2), perfect tenses involve a third time: a reference time which the event is presented with respect to. Thus in the examples of (3), the described event is in each case in the past with respect to the speech time, though in (3a) it is presented from the point of view of a point in time which is also in the past, whereas in (3b) it is presented from the point of view of the current time, and in (3c)

it is presented from the point of view of a time in the future.

(2) a. Tigger miaowed (Simple Past)

b. Tigger miaows (Simple Present)

c. Tigger shall miaow (Simple Future)

(3) a. Tigger had miaowed (Past Perfect)

b. Tigger has miaowed (Present Perfect)

c. Tigger shall have miaowed (Future Perfect)

Reichenbach's analysis is summarised via time lines in Table 7.1.

	Simple	Perfect
Past	R,E S	E R S
Present	S,R,E	E S,R
Future	S R,E	S E R

Table 7.1: Reichenbachian Tenses

Note that the entry for the simple future in this table differs from that of [Reichenbach 47]. In his simple future, the speech time and reference time are taken to be coincident, preceding the event time. We dispute this, and claim that in a simple future sentence such as 'I shall see John', the reference time and event time our coincident, with both following the speech time. (Though below we suggest that a reference time is not needed for the simple tenses.) This alteration is important as it allows us to treat tense and aspect via independent dimensions, as discussed below. It should also be noted that there is no real future tense in English: future tense is indicated solely by the appropriate modal verb in the auxiliary chain.

Given the above table, past, present and future tense can be seen to be independent from the aspectual dimension corresponding to the perfective. Tense concerns the relationship between the reference time and the speech time. For past tense, the reference

time precedes the speech time. For present tense the reference and speech times are coincident. For future tense the reference time follows the speech time. Aspect concerns the relationship between the event time and the reference time. In each of the simple tenses, the event time and reference time are coincident. In each of the perfect tenses, the event time precedes the reference time.

Reichenbach also considers extended, or progressive tenses. For each of the tenses in Table 7.1 there is an associated progressive tense. These progressive tenses are illustrated by the examples in (4) and (5).

(4) a. Tigger was miaowing (Simple Progressive Past)

b. Tigger is miaowing (Simple Progressive Present)

c. Tigger shall be miaowing (Simple Progressive Future)

(5) a. Tigger had been miaowing (Past Progressive Perfect)

b. Tigger has been miaowing (Present Progressive Perfect)

c. Tigger shall have been miaowing (Future Progressive Perfect)

The progressive, which is marked by the use of the progressive auxiliary (the appropriate form of 'be') and the progressive participle form of the main verb (i.e., the +'ing' form), indicates that the event is extended or repeated in time. This aspectual dimension is independent of tense or the perfective aspect in the above table: all time line diagrams may be altered to depict progressive aspect by making the event extend/repeat in time.

7.2 The English Auxiliary Chain

The structure of the English auxiliary chain is fairly straightforward and well-documented. Following [Burton-Roberts 86, pp. 118–126], a verb phrase consists of an optional modal auxiliary, followed by an optional perfective auxiliary, followed by an optional progressive auxiliary, followed by an optional passive auxiliary, followed by the main verb, followed by the complements of the main verb. The optionality of each of the auxiliaries is independent. We thus have the following syntax for the verbal chain:

(modal) (perfective) (progressive) (passive) main verb

where the parentheses indicate the optionality.

Tense is marked on the first verb, normally an auxiliary, in the chain. Each verb must agree in form with the following/preceding verb in the chain. With each verb form we may associate a semantic function mapping (parametric) propositions to (parametric) propositions corresponding to the appropriate tense/aspect of that form. Table 7.2 shows the various forms of 'miaow', together with their semantic content given in terms of such semantic functions.

Verb	Form	Semantic Content
miaow	*bse*	$S_d \models \langle\langle \text{miaow}, \dot{x}, \dot{l}; + \rangle\rangle$
miaowed	*fin*	$\text{past}(S_d \models \langle\langle \text{miaow}, \dot{x}, \dot{l}; + \rangle\rangle)$
miaows	*fin*	$\text{pres}(S_d \models \langle\langle \text{miaow}, \dot{x}, \dot{l}; + \rangle\rangle)$
miaowing	*prp*	$\text{prog}(S_d \models \langle\langle \text{miaow}, \dot{x}, \dot{l}; + \rangle\rangle)$
miaowed	*psp*	$\text{perf}(S_d \models \langle\langle \text{miaow}, \dot{x}, \dot{l}; + \rangle\rangle)$

Table 7.2: Main Verbal Forms

Auxiliary verbs may be thought of as mapping a verb phrase in one form to a verb phrase in another form. Given this, and the independence of tense, perfective aspect and progressive aspect, we may prepare a similar table for auxiliaries. Such a table is given as Table 7.3.

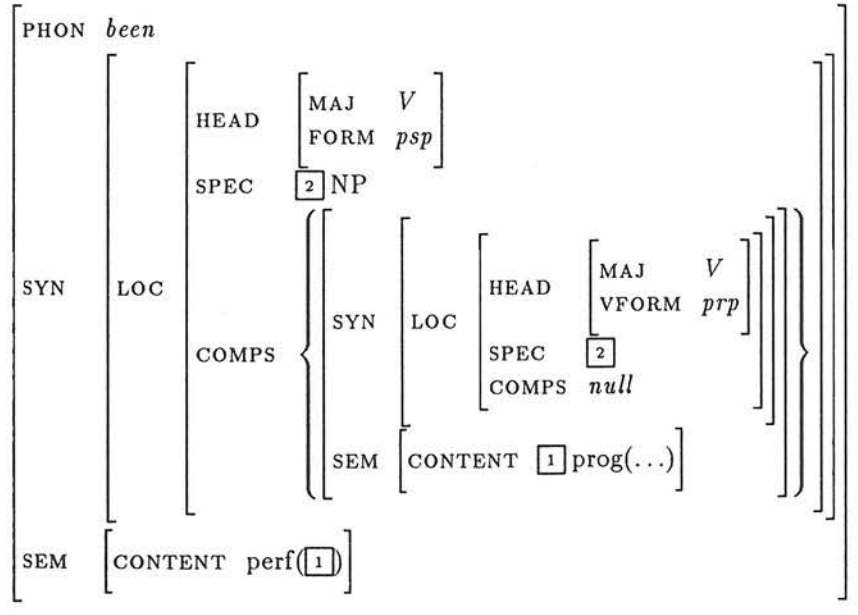
This treatment of auxiliaries suggests that the sequencing of verb forms within the auxiliary chain, and in fact verb form itself, may be treated as a semantic phenomenon: the syntactic FORM attribute is mostly redundant in the above tables, and if we could motivate a semantic function corresponding to the verb form *bse*, verb form could be treated wholly semantically. The auxiliary form 'had', for example, might be taken to subcategorise for a verb phrase whose semantic content is of the form "perf(...)", and whose semantic content is "past(perf(...))".

Verb	Form of Head	Form of Complement	Semantic Content of VP Complement	Semantic Content of Head
shall	*fin*	*bse*	[1]	shall([1])
had	*fin*	*psp*	[1] = perf(...)	past([1])
has	*fin*	*psp*	[1] = perf(...)	pres([1])
have	*bse*	*psp*	[1] = perf(...)	[1]
was	*fin*	*prp*	[1] = prog(...)	past([1])
is	*fin*	*prp*	[1] = prog(...)	pres([1])
be	*bse*	*prp*	[1] = prog(...)	[1]
been	*psp*	*prp*	[1] = prog(...)	perf([1])

Table 7.3: Auxiliary Verbal Forms

7.3 Auxiliaries in CPSG

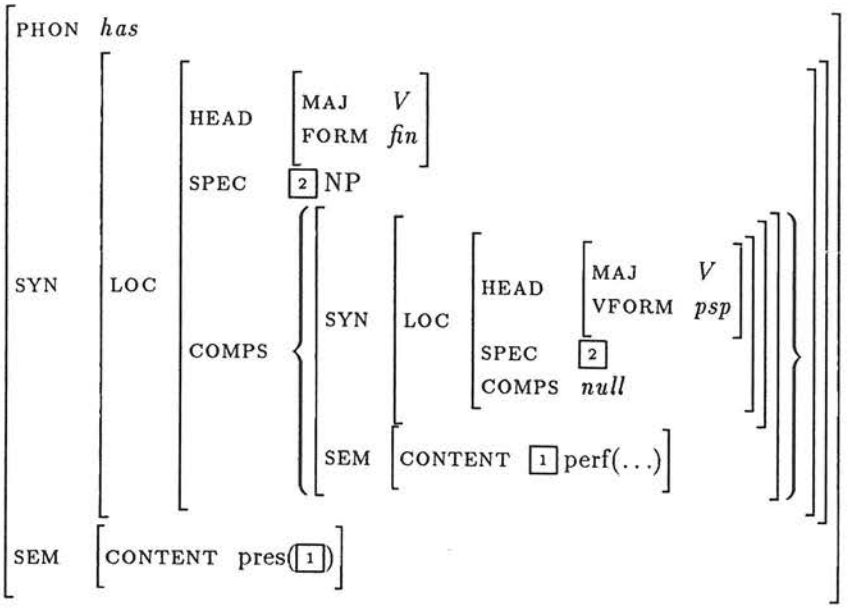
Syntactically, auxiliaries are treated in CPSG in much the same way as in HPSG and CG — as heads which combine with a verb phrase to yield a verb phrase, with the FORM attribute (as discussed above) and linear precedence rules combining to ensure the correct sequencing of the auxiliaries. Given the use of the semantic functions introduced above, the mechanics of the semantic treatment is very simple, as in, for example:



Here we have made the semantics of the whole phrase functionally dependent on the

semantics of the embedded verb phrase. (The function could be replaced by the equivalent relation if we wish to consistently use only relations.) The real task is to specify the function exactly.

Similarly, for a perfective auxiliary we may have:



Given this treatment, the semantic contents of the previous examples are:

(2) a′. past($S_d \models \langle\langle$miaow, t, l; +$\rangle\rangle$) (Simple Past)

b′. pres($S_d \models \langle\langle$miaow, t, l; +$\rangle\rangle$) (Simple Present)

c′. shall($S_d \models \langle\langle$miaow, t, l; +$\rangle\rangle$) (Simple Future)

(3) a′. past(perf($S_d \models \langle\langle$miaow, t, l; +$\rangle\rangle$)) (Past Perfect)

b′. pres(perf($S_d \models \langle\langle$miaow, t, l; +$\rangle\rangle$)) (Present Perfect)

c′. shall(perf($S_d \models \langle\langle$miaow, t, l; +$\rangle\rangle$)) (Future Perfect)

(4) a′. past(prog($S_d \models \langle\langle$miaow, t, l; +$\rangle\rangle$)) (Simple Progressive Past)

b′. pres(prog($S_d \models \langle\langle$miaow, t, l; +$\rangle\rangle$)) (Simple Progressive Present)

c′. shall(prog($S_d \models \langle\langle$miaow, t, l; +$\rangle\rangle$)) (Simple Progressive Future)

(5) a′. past(perf(prog($S_d \models \langle\langle$miaow, t, l; +$\rangle\rangle$))) (Past Progressive Perfect)

b′. pres(perf(prog($S_d \models \langle\langle$miaow, t, l; +$\rangle\rangle$))) (Present Progressive Perfect)

c′. shall(perf(prog($S_d \models \langle\langle$miaow, t, l; +$\rangle\rangle$))) (Future Progressive Perfect)

All functions are taken to map propositions to propositions.

7.4 The Tense and Aspect Functions

It remains then to specify the semantic functions we have employed above. These include functions for tense (*past* and *pres*), functions for aspect (*perf* and *prog*) and functions corresponding to the modal verbs (*shall*, *should*, etc.).

The perfective presents one potential difficulty given the above Reichenbachian approach. In the simple tenses, we have $E = R$, whereas in the perfect tenses we have $E < R$. Thus the requirement that $E = R$ appears to be a default: in the absence of the perfective this holds, but in the presence of the perfective it is overridden by $E < R$. Such defaults present difficulties, and our approach to this is to reconsider Reichenbach's appeal to three points in time. In particular, the three points are not required in the cases of simple tenses. We therefore propose to do without a reference time for simple tenses. As demonstrated below, this allows us to avoid the above problem with defaults.

Suppose we associate a location in time with each proposition. This time might be thought of as the time with respect to which the proposition is presented. For notational convenience we write this location as a left subscript of the proposition, so for example we have:

$$_{\alpha}\text{shall}(_{\beta}\text{perf}(_{\gamma}\text{prog}(_{\delta}\text{P})))$$

The idea is that the functions constrain the times which correspond to Reichenbach's locations in time. The semantic functions can be stated as:

$_{\alpha}\text{past}(_{\beta}\text{P})$ is true iff P is true and $\alpha > \beta$

$_{\alpha}\text{pres}(_{\beta}\text{P})$ is true iff P is true and $\alpha = \beta$

$_{\alpha}\text{shall}(_{\beta}\text{P})$ is true iff P is true and $\alpha < \beta$

$_{\alpha}\text{perf}(_{\beta}\text{P})$ is true iff P is true and $\alpha > \beta$

$_{\alpha}\text{prog}(_{\beta}\text{P})$ is true iff P is true and $\alpha = \beta$ is extended/repeated

These definitions are obviously naïve in places — modals for a start require some sort of intensionality — but given these functions, we have, for example:

$$_{\alpha}\text{shall}(_{\beta}\text{perf}(_{\gamma}\text{prog}(_{\delta}\text{P})))$$

iff

$$\text{P is true, } \alpha < \beta,\ \beta > \gamma,\ \gamma = \delta, \text{ and } \delta \text{ is extended/repeated.}$$

The time associated with the semantic content of a finite verb phrase corresponds to Reichenbach's speech time (S). The time associated with a bare proposition (the most deeply embedded argument) corresponds to Reichenbach's event time (E). The time corresponding to the result of applying the function *perf* corresponds to Reichenbach's reference time (R). Thus writing S for α, R for β and E for both γ and δ in the above gives $S < R$ and $E < R$ and E is extended/repeated, which is the future progressive perfect.

7.5 Tense and Aspect in CPSG

Having defined the above functions, we are now ready to formalise them in situation theory. The usual approach to tense in situation theory is to relate the location argument of an infon that characterises the described situation to the location of the utterance with the appropriate temporal type. Thus for an utterance of a simple past sentence such as 'Tigger miaowed' we might have:

$S_d \models \langle\langle$miaows, miaower: $\dot{t}$, location: $\dot{l}$; $+\rangle\rangle$

$S_r \models \langle\langle$utterance location, location: $\dot{l_u}$; $+\rangle\rangle$

$S_r \models \langle\langle$naming, name: "Tom", named: $\dot{t}$; $+\rangle\rangle$

$(\dot{l}, \dot{l_u})$: *temporally precedes*

[Pollard & Sag 87] hint at an approach to tense like this when discussing the lexical hierarchy. In particular, they claim that the signs for all past tense verbs share the following information:

$$\left[\begin{array}{l}\text{SEMANTICS} \left[\begin{array}{ll}\text{CONTENT} & \left[\text{LOCATION } \boxed{1}\right] \\ \text{CONTEXT} & \left\{\left[\begin{array}{ll}\text{VAR} & \boxed{1} \\ \text{RESTRICTION} & \left[\begin{array}{ll}\text{RELN} & \textit{temporally precedes} \\ \text{LOC1} & \boxed{1} \\ \text{LOC2} & l_d\end{array}\right]\end{array}\right]\right\}\end{array}\right]\end{array}\right]$$

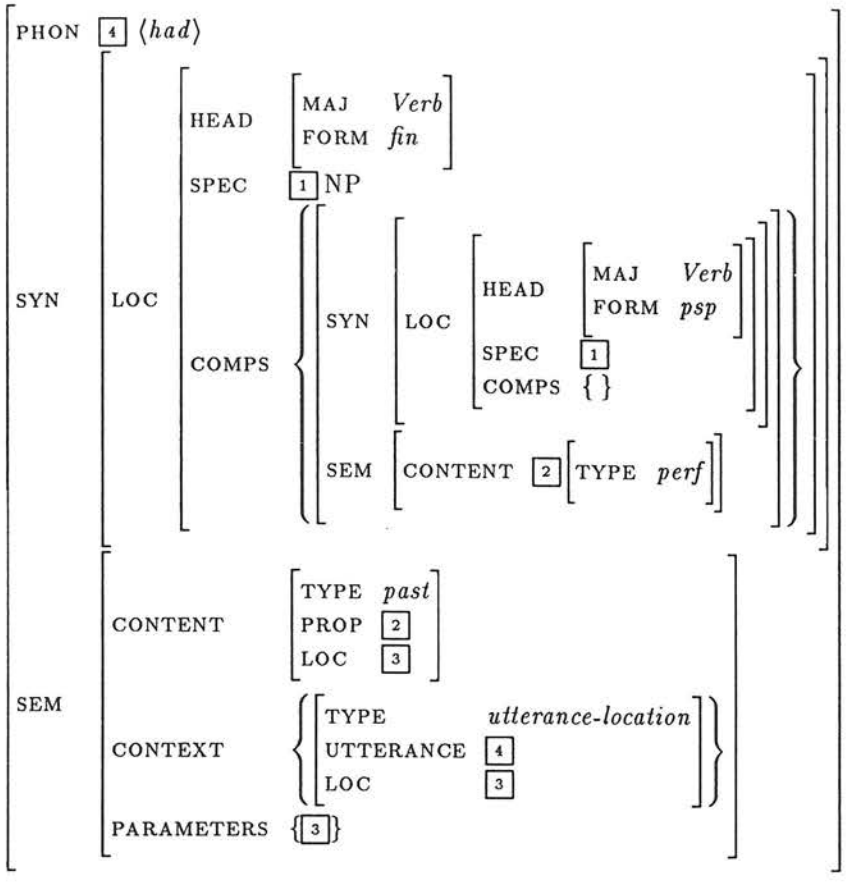
Rephrasing this in CPSG, we would have:

$$\left[\begin{array}{l}\text{SEMANTICS} \left[\begin{array}{ll}\text{CONTENT} & \left[\text{INF } \left[\text{ISSUE } \left[\text{LOCATION } \boxed{1}\right]\right]\right] \\ \text{CONTEXT} & \left\{\left[\begin{array}{ll}\text{TYPE} & \textit{temporally precedes} \\ \text{LOC1} & \boxed{1} \\ \text{LOC2} & \boxed{2}\end{array}\right]\right\}\end{array}\right]\end{array}\right]$$

where further restrictions are required to ensure that $\boxed{2}$ is anchored to the utterance location.

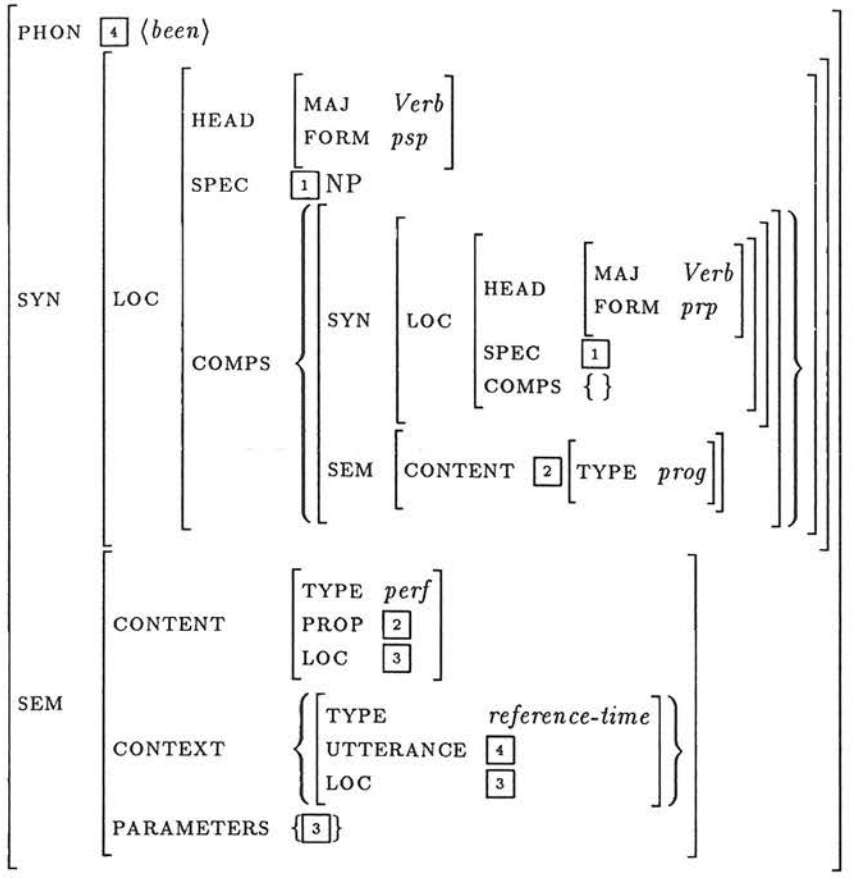
Given that CPSG takes the semantic content of a verb phrase to be a proposition, rather than an infon, the location of the event is not so accessible. Our treatment of tense and aspect thus relies on the function *loc* of section 5.2 which maps propositions to associated locations. Recall that in the simple case of a proposition of the form $(S \models \sigma)$, the location of the proposition is just the value of the location argument of σ. We define the function by cases for more complex propositions.

The semantics of tensed verbs is given in terms of binary types, such as *past* and *pres*, which holds of a location and a proposition. A location l and a proposition P are of type *past* iff the location of P precedes l. (This type may be expressed in terms of more primitive types such as *temporally-precedes*.) Given this type, we have the following partial description for the lexical constituent 'had':



We also define $loc((P, l : past)) = l$.

To treat perfectives, we employ a binary type *perf* which holds of a location l and a proposition P iff the location of P precedes l. With this type, we have for 'been':



We define $loc((P, l) : perf) = l$.

We treat the progressive via a unary type *prog*, which holds of a proposition just in case the location of the proposition is either extended or repeated. A partial description of 'miaowing' is:

$$\begin{bmatrix}
\text{PHON} & \boxed{4}\ \langle miaowing \rangle & & \\
\text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & Verb \\ \text{FORM} & prp \end{bmatrix} \\ \text{SPEC} & \text{NP}\boxed{1} \\ \text{COMPS} & \{\,\} \end{bmatrix} \end{bmatrix} \\
\text{SEM} & \begin{bmatrix} \text{CONTENT} & \begin{bmatrix} \text{TYPE} & prog \\ \text{PROP} & \begin{bmatrix} \text{TYPE} & supports \\ \text{SIT} & \boxed{3} \end{bmatrix} \\ \text{INF} & \begin{bmatrix} \text{ISSUE} & \begin{bmatrix} \text{RELN} & miaow \\ \text{AGENT} & \boxed{1} \end{bmatrix} \\ \text{POLARITY} & + \end{bmatrix} \end{bmatrix} \\ \text{CONTEXT} & \left\{ \begin{bmatrix} \text{TYPE} & utterance\text{-}situation \\ \text{UTTERANCE} & \boxed{4} \\ \text{SIT} & \boxed{3} \end{bmatrix} \right\} \\ \text{PARAMETERS} & \{\boxed{3}\} \end{bmatrix}
\end{bmatrix}$$

We define $loc(P : prog) = loc(P)$.

Taking these together gives as the value of the SEMANTICS attribute of the verb phrase 'had been miaowing':

Content: $(((\dot{s} \models \langle\langle \text{miaow}, x; + \rangle\rangle) : prog, \dot{y}) : perf, \dot{z}) : past$

Context: $\left\{ \begin{array}{c} (\langle \underline{had\ been\ miaowing} \rangle, \dot{s} : utterance\text{-}situation) \\ (\langle \underline{had\ been\ miaowing} \rangle, \dot{y} : reference\text{-}time) \\ (\langle \underline{had\ been\ miaowing} \rangle, \dot{z} : utterance\text{-}location) \end{array} \right\}$

Parameters: $\{\dot{s}, \dot{y}, \dot{z}\}$

Lastly, we define truth conditions for the types employed in the treatment. These follow the semantic functions of section 7.4:

$(P, l : past)$ is true iff $l > loc(P)$

$(P, l : pres)$ is true iff $l = loc(P)$

$(P, l : shall)$ is true iff $l < loc(P)$

$(P, l : perf)$ is true iff $l > loc(P)$

$(P : prog)$ is true iff $loc(P)$ is extended/repeated

8 Summary

In this chapter, which has described in detail the value taken by the SEMANTICS attribute for descriptions of various constituents, several major difference between the CPSG and the HPSG treatment of semantics have been highlighted. These include:

- the use of propositions, rather than infons, both in the content and the context, allowing distinctions between resource situations to be captured and allowing the use of types other than *supports*,
- a simplified approach to the use and modelling of parameters, with the CONTEXT attribute being used to encode restrictions on all parameters (rather than encoding restructed parameters) and a separate PARAMETERS attribute to encode the parameters themselves, and
- the treatment of nouns as denoting types, rather than as partial quantified infons, and the associated treatment of noun modifiers as type modifiers.

We have also extended the semantic phenomena under consideration by discussing the semantics of adjuncts and coordinated phrases, as well as giving a detailed discussion of the treatment of tense and aspect. Quantifier scoping and the semantics of distributive noun phrase coordination remain problematic. These areas are discussed in detail in Chapter 8.

Chapter 8

Quantification and Quantifier Scope

This chapter concerns the treatment of quantification in CPSG. Much work has been done on the treatment of quantification within various frameworks, and there remain many problems. We begin in section 1 with a discussion of the treatment of quantification in situation semantics. This section, which is based on [Richard Cooper 90a], argues for the situation theoretic treatment of quantification which CPSG adopts. It advocates the view, based on the strong distinction between the situation relative and the absolute, that quantification should be treated in terms of types, rather than relations. Quantification in HPSG is then considered briefly, with special emphasis on the problem of quantifier scoping. More time is spent on quantification in 2–14–89, a theory descended from HPSG and presented by Pollard in [Pollard 89b]. The treatment of quantification in CPSG incorporates several aspects of that of 2–14–89, but also includes the situation semantic treatment of quantification in terms of types advocated in section 1, the use of a global resource situation to determining scoping, and the use of a Principle of Semantic Interpretation, based on the scoping algorithm of [Lewin 90].

Also discussed in this chapter are some residual problems with the treatment of distributive readings of coordinated noun phrases. The treatment of these bears a striking resemblance to the treatment of quantification.

1 Quantification in Situation Semantics

1.1 Some Approaches to Quantification

Numerous approaches to the treatment of natural language quantification within situation semantics have been suggested. What these approaches have in common is that they treat the situation theoretic objects corresponding to the content of sentences containing quantified NPs as being *structurally determined* — as being in some way dependent on other infons supported by the described situation. In general, an infon or proposition is structurally determined if its holding in a situation (for an infon) or its truth (for a proposition) is determined by the truth or falsity of other propositions. In the case of quantification, quantificational sentences are generally taken to correspond to infons and conditions are given for when a situation will support such infons, and for what a situation must be like if it does support such infons. This seems a reasonable stance to adopt. The following subsections discuss some of the possible treatments.

1.1.1 A Relation Between Properties

Following [Robin Cooper 87], the treatment of determiners as generalised quantifiers ([Barwise & Cooper 81]) within set-theoretic semantic formalisms may be translated into situation semantics by treating determiners as binary relations between properties. Under such an analysis, a sentence such as 'most kittens are miaowing' describes a situation S_d such that

$$S_d \models \langle\langle \text{most}, [x \mid \langle\langle \text{kitten}, x; + \rangle\rangle], [x \mid \langle\langle \text{miaowing}, x; + \rangle\rangle]; + \rangle\rangle$$

where if

$$S \models \langle\langle det,\ P,\ Q;\ + \rangle\rangle$$

then

$$det'(\{x \mid S \models \langle\langle P,\ x\ ;\ + \rangle\rangle\},\ \{x \mid S \models \langle\langle Q,\ x;\ + \rangle\rangle\})$$

Here (and throughout this chapter) det' denotes the extensional set theoretic generalised quantifier relation that corresponds to the situation theoretic relation *det*.

In [Robin Cooper 87], the conditional is only required to go one way: if a situation S is such that $S \models \langle\langle det,\ P,\ Q;\ + \rangle\rangle$ then ..., but not vice versa. We adopt the view

that situations should be closed under some form of "infon equivalence", and hence that we are dealing with a bi-conditional. Thus if a situation supports various infons then it also supports related quantificational infons, and *vice versa*. This is consistent with the treatment of conjunctive, disjunctive, and quantified infons given in [Barwise 88a] and [Barwise 88b], where a bi-conditional relationship is required between related propositions.

1.1.2 A Relation Between Types

Treating quantifiers as relations between properties does not correctly account for the exploitation of resource situations. In particular, with referential uses of noun phrases (we shall not be concerned with attributive uses), the restriction property is understood to be relative to a resource situation, whilst the scope property is relative to the described situation. We might thus propose that determiners should be treated as relations between object types, so that 'most kittens are miaowing' describes a situation S_d such that

$$S_d \models \langle\langle \text{most}, [x \mid S_r \models \langle\langle \text{kitten}, x; + \rangle\rangle], [x \mid S_d \models \langle\langle \text{miaowing}, x; + \rangle\rangle]; + \rangle\rangle$$

where S_r is the resource situation exploited by the noun phrase and

$$S \models \langle\langle det, T_1, T_2; + \rangle\rangle$$

iff

$$det'(\{x \mid x : T_1\}, \{x \mid x : T_2\})$$

This is slightly worrying because, by building in the resource situation and described situation, the resulting infon losses the "situation relative" nature of infons. Under this definition, if any situation supports $\langle\langle det, T_1, T_2; + \rangle\rangle$, then every situation supports it. Note that of course this is a consequence of the bi-conditional in the structurally determined definition.

One way around the situation independence problem is to incorporate the requirement that we are only quantifying over objects of the situation in question. Thus we might have:

$$S \models \langle\langle det, T_1, T_2; + \rangle\rangle$$

iff

$$det'(\{x \mid x : T_1 \ \wedge \ x \in Obj(S)\}, \ \{x \mid x : T_2 \ \wedge \ x \in Obj(S)\})$$

1.1.3 A Relation Between a Type and a Property

An alternate solution to the problem of situation independence above is to treat determiners as relations between an object type and a property. With this treatment, 'most kittens are miaowing' describes a situation S_d such that

$$S_d \models \langle\langle \text{most}, [x \mid S_r \models \langle\langle \text{kitten}, x; +\rangle\rangle], [x \mid \langle\langle \text{miaowing}, x; +\rangle\rangle]; +\rangle\rangle$$

where again S_r is the resource situation exploited by the noun phrase and

$$S \models \langle\langle det, \ T, \ Q; \ +\rangle\rangle$$

iff

$$det'(\{x \mid x : T\}, \ \{x \mid S \models \langle\langle Q, \ x; \ +\rangle\rangle\})$$

Alternately we might require complete information about those objects which the quantification concerns. That is, we might define the conditions under which quantificational infons hold as:

$$S \models \langle\langle det, \ T, \ Q; \ +\rangle\rangle$$

iff for all x such that $(x : T)$, either $S \models \langle\langle Q, \ x; \ +\rangle\rangle$ or $S \models \langle\langle Q, \ x; \ -\rangle\rangle$, and furthermore

$$det'(\{x \mid x : T\}, \ \{x \mid S \models \langle\langle Q, \ x; \ +\rangle\rangle\})$$

1.1.4 Quantified Infons

Some versions of situation theory (e.g. [Barwise 88a], [Barwise 88b], [Gawron & Peters 90b], [Devlin ∞]) admit *quantified infons*, infons of the form:

$$\exists x^{\tau(x)}\sigma(x)$$

and

$$\forall x^{\tau(x)}\sigma(x)$$

In these infons the superscripted infon $\tau(x)$ acts as a restriction on the parameter x which is bound by the determiner (either $\exists$ or $\forall$). A situation S supports the existentially

quantified infon iff there is some object b in S such that $S \models \tau(b)$ and $S \models \sigma(b)$. Similarly a situation S supports the universally quantified infon iff for all objects b of S such that $S \models \tau(b)$, it is also the case that $S \models \sigma(b)$. Although these quantified infons are not normally intended to capture natural language quantification, [Pollard & Sag 87] extend the use of quantified infons for the HPSG treatment of natural language semantics. This is achieved by admitting quantifiers corresponding to the full range of natural language determiners. Their theory admits quantified infons of the form:

$$(\text{most } x^{\tau(x)})\ \sigma(x)$$

Pollard and Sag do not state structurally determined definitions giving the conditions under which a situation will support a quantified infon, though presumably such definitions follow the general pattern above:

$$S \models (det\ x^{\tau(x)})\ \sigma(x)$$

iff

$$det'(\{x \mid S \models \tau(x)\},\ \{x \mid S \models \sigma(x)\})$$

Note that such a definition takes the restriction to be with respect to the situation supporting the infon, not with respect to a resource situation. This proposal might be modified to correctly account for the exploitation of resource situations by treating restrictions in terms of propositions rather than infons. A further minor criticism concerns Pollard and Sag's use of terms of the form (most $x^{\tau(x)}$). It is not clear what situation theoretic objects such terms correspond to. These terms are, however, not essential to the approach.

1.1.5 A Property of a Property

[Gawron & Peters 90b] point out a potential problem for the above analyses concerning the binding of pronouns. Under the above approaches, and assuming the absorption principle of [Gawron & Peters 90a] (which requires that, in the instance below, because of the dependence of x on y, y be abstracted when x is), the sentence (from their 65):

> Every psychiatrist who dates a patient of hers will be sued by him.

would be required to have content:

$$\langle\!\langle \text{every}, [x, y \mid S_r \models \langle\!\langle \text{psych}, x_{S_r \models \tau(x,\ y)};\ + \rangle\!\rangle], [x \mid \langle\!\langle \text{will-sue}, y, x;\ + \rangle\!\rangle];\ + \rangle\!\rangle$$

where

$$\tau(x,\ y) = \wedge\{\langle\!\langle \text{patient}, y, x;\ + \rangle\!\rangle, \langle\!\langle \text{dates}, x, y;\ + \rangle\!\rangle\}$$

Note that, because of the absorption principle we are lead to a two place type rather than a one place type. The second argument role may be interpreted as unsaturated, effectively existentially quantifying over that argument. This technical detail is not relevant to the point in question.

With the above translation, 'him' cannot be anaphoric to 'a patient of hers', because y is absorbed when x is abstracted over to form the restriction type.

To overcome this problem, [Gawron & Peters 90b] suggest an alternate treatment of determiners where the determiner expresses a property of a property and minimal appropriateness conditions are employed to give a structurally determined definition. They take the sentence 'most kittens are miaowing' to describe a situation S_d such that

$$S_d \models \langle\!\langle \text{most}, [x_{S_r \models \langle\!\langle \text{kitten}, x;\ + \rangle\!\rangle} \mid \langle\!\langle \text{miaowing}, x;\ + \rangle\!\rangle];\ + \rangle\!\rangle$$

The accompanying structurally determined definition is

$$S \models \langle\!\langle det,\ P;\ + \rangle\!\rangle$$

iff

$$det'(\{x \mid x \text{ is appropriate for } P \text{ (in } S)\},\ \{x \mid S \models \langle\!\langle P,\ x;\ + \rangle\!\rangle\})$$

The restriction on the abstracted parameter in the property is treated as an appropriateness condition on the argument role of the resultant property. Thus in the case of 'most kittens are miaowing', this definition requires that the "most" relation hold between the set of things appropriate for "miaowing kittens" and the set of things that actually have the "miaowing" property in the described situation.

Whilst this approach makes elegant use of the notion of appropriateness conditions provided by situation theory, there are a number of difficulties which it presents. These have to do with the precise details of appropriateness conditions. For example, if a

restricted parameter is abstracted, then the restriction on the parameter becomes an appropriateness condition on the resultant argument role. In the first instance this is what is required — the restriction on the relevant parameter, when translated as an appropriateness condition, restricts the domain of quantification. However, parameters may be restricted by other factors. 'Every kitten is chasing its tail', for example, might be taken to correspond to "everything that is appropriate for P has P", where P is the property (or type) of thing that is a kitten chasing its tail. Although it is not clear, it seems reasonable to assume that manx kittens, kittens without tails, are not appropriate for P, and as such 'every kitten is chasing its tail' can still be supported by a situation even though that situation may contain manx kittens which are not chasing there tails. This seems counter-intuitive.

1.2 Persistence and Quantification

An infon σ is persistent if and only if whenever a situation S supports it, every situation S' of which S is a part also supports it. The issue of when an infon should be persistent is not clear cut, and it forms the basis of two of Barwise's branch points [Barwise 88b, branch points 6 and 18]. Persistence can also be seen as a special case of a kind of monotonicity in an argument role of a proposition: σ is persistent in S if and only if $(S', \sigma :\models)$ is true for all situations S' for which $S \trianglelefteq S'$. As such persistence can be generalised from infons to propositions. Persistence is clearly intimately related to the $\trianglelefteq$ relation that holds between situations. This relation cannot normally be defined in purely extensional terms as in section 2.5 of Chapter 6 as this would lead to all infons being persistent by definition. We return to this issue in section 1.3.

The reason for raising the issue of persistence is that there is at least a pre-theoretic view that the content of sentences containing quantificational noun phrases should not be persistent: if most kittens are miaowing in one situation there is no guarantee that the same can be said of a larger situation, especially one in which there are more kittens. It is thus of interest to consider how the issue of persistence relates to the above treatments of quantification. Not all of the above treatments are considered here. For the most part the details are fairly obvious. The following short survey, however, serves to highlight some relevant points.

1.2.1 A Relation Between Properties

Treating quantification in terms of a relation between properties leads to a non-persistent infon. In one situation, it may be the case that most of the objects having the property of kittenhood also have the property of miaowing, but there is no guarantee that in a larger situation there may be further kittens not miaowing. Alternately a larger situation may resolve some of the issues regarding miaowing kittens which were unresolved in the original situation, and thereby cause the quantificational infon not to hold.

1.2.2 A Relation Between Types

The original formulation of quantification as a relation between types leads to an infon which, if supported by one situation, is supported by all situations. As such, the infon is trivially persistent. We consider this situation independence further below.

The alteration to this treatment, restricting attention only to those objects in the situation in question, leads to a non-persistent infon. A larger situation may involve further objects which may prevent the quantificational infon holding in the larger situation.

1.2.3 A Relation Between a Type and a Property

The original treatment of quantification in terms of a relation between a type and a property need not be persistent. Although the set of objects under consideration is fixed by the restriction type, in the case of monotone decreasing determiners, such as 'no', a larger situation may reveal further information about those objects preventing the quantificational infon from being persistent. This might be side-stepped by giving alternate structurally determined definitions for monotone increasing and monotone decreasing quantifiers, so that monotone decreasing quantifiers are determined in terms of negative information. We don't investigate this possibility here.

The alteration to the treatment, requiring that all relevant information is known about the objects in question, does lead to a persistent infon. The objects under consideration are determined by the type contributed by the restriction. This set is thus fixed, and the requirement that everything relevant be known about those objects means that taking a larger situation cannot reveal anything more about those objects. Note though that

this treatment does require complete information about all objects under consideration, although in general there will not be many such objects: only those objects having the property of kittenhood in the resource situation.

1.3 Absoluteness and Situation Dependence

The various treatments of quantification and the consideration of persistence raise two principal issues: structural determination and completeness of information. Recall that in the theory presented in Chapter 6, a sharp distinction was drawn between the situation dependent and the situation independent. Infons and relations are situation dependent, whereas propositions and types are situation independent, or absolute: propositions are true or false, whereas infons hold in a situation. Types are situation independent analogues of relations, and cannot participate in infons in the same way as relations. Given this distinction, we can view structurally determined "facts" as corresponding to propositions, rather than infons. That is, we may treat structurally determined "facts" as being on the logical plane, rather than on the informational plane. This is consistent with the fact that the logic reflects our ability to stand back and look at situations from the "outside". Our ability to entertain structurally determined facts depends crucially on our ability to adopt this perspective.

1.3.1 Determiners as Types

Each of the above accounts of quantification can thus be revised to treat determiners as corresponding to types rather than relations. The first account, where the determiner corresponds to a relation between two properties, for example, might be revised so that the determiner corresponds to a three place type that holds between a situation and two properties if and only if the corresponding generalised quantifier relation holds between the set of things having the first property in the situation and the set of things having the second property in the situation. That is,

$$(S, P, Q) : det$$

iff

$$det'(\{x \mid S \models \langle\langle P,\ x;\ +\rangle\rangle\},\ \{x \mid S \models \langle\langle Q,\ x;\ +\rangle\rangle\})$$

Similarly, the other accounts may be revised. Revising the property of property account,

determiners could correspond to two place types that hold between a situation and a property iff the corresponding generalised quantifier relation holds between the set of individuals appropriate for the property (in the situation) and the set of individuals that actually have the property in the situation. That is

$$(S, P) : det$$

iff

$$det'(\{x \mid x \text{ is appropriate for } P \text{ (in } S)\},\ \{x \mid S \models \langle\!\langle P,\ x;\ + \rangle\!\rangle\})$$

Proceeding along these lines, the revision of the treatment of determiners as binary relations between types leads to perhaps the most natural formulation: that determiners are two place types which hold of two object types iff the corresponding set-theoretic relation holds of the sets of objects of those types. The use of a type rather than a relation exactly captures the situation independence of this formulation, and maintains the intuition that determiners are like two place relations, the difference being that determiners are the situation *independent* analogues of two-place relations, where relations are situation *dependent*.

1.3.2 Other Structurally Determined Types

Quantification provides one source of structurally determined proposition. In many versions of situation theory an algebra is constructed on the domain of infons with the logical operators of conjunction, disjunction, and negation. Structurally determined definitions are then given for when a situation supports conjunctive, disjunctive or negative infons:

Conjunctive Infons:

$$S \models \wedge\Sigma \quad \text{iff} \quad \forall \sigma \in \Sigma,\ S \models \sigma$$

Disjunctive Infons:

$$S \models \vee\Sigma \quad \text{iff} \quad \exists \sigma \in \Sigma,\ S \models \sigma$$

External Negation:

$$S \models \neg\sigma \quad \text{iff} \quad S \not\models \sigma$$

Note that $\neg\sigma$ is not persistent, where as $\wedge\Sigma$ and $\vee\Sigma$ are.

In the version of situation theory developed here, we take the structural determination of these infons, which is a consequence of their logical character, as evidence that they should be treated as propositions, rather than infons. Rather than admitting compound infons, we take all infons to be basic, and treat all logic (i.e., situation independent, or absolute, things) at the level of types and propositions.

Amongst the types of our theory, we include the types *and* and *or*, which are structurally determined two place types that hold between a situation and a set of infons:

$$(S, \Sigma) : \text{and} \quad \text{iff} \quad \forall \sigma \in \Sigma \;\; S \models \sigma$$

and

$$(S, \Sigma) : \text{or} \quad \text{iff} \quad \exists \sigma \in \Sigma \;\; S \models \sigma$$

Persistence of the types *and* and *or* arises from a kind of upward monotonicity in their first argument role: if $S \trianglelefteq S'$, then

$$(S, \Sigma) : \text{and} \;\Rightarrow\; (S', \Sigma) : \text{and}$$

and likewise

$$(S, \Sigma) : \text{or} \;\Rightarrow\; (S', \Sigma) : \text{or}$$

We also have

$$(S, \Sigma) : \text{and} \;\Rightarrow\; (S, \Sigma) : \text{or}$$

From a form of upward monotonicity in the second argument role of *or*, we have:

$$(S, \Sigma) : \text{or} \;\Rightarrow\; (S, \Sigma') : \text{or}$$

for any superset Σ' of Σ,

and from a form of downward monotonicity in the second argument role of *and*, we have:

$$(S, \Sigma) : \text{and} \;\Rightarrow\; (S, \Sigma') : \text{and}$$

for any subset Σ' of Σ.

We also include a type corresponding to external negation. The type *not* is a two place type that holds of a situation and an infon:

$$(S,\sigma) : \text{not} \quad \text{iff} \quad (S,\sigma) \not\models$$

$$\text{iff} \quad S \not\models \sigma$$

In this version of situation theory then, all infons are persistent and the part of relation between situations can be defined purely extensionally:

$$(S_1, S_2) : \trianglelefteq \quad \text{iff} \quad \{\sigma \mid S_1 \models \sigma\} \subseteq \{\sigma \mid S_2 \models \sigma\}$$

1.4 Quantification and Natural Language Semantics

The treatment of quantification in terms of types and propositions may easily be incorporated into the treatment of semantics of CPSG and, if we interpret the semantic content of an HPSG sign not as describing an infon but as describing a situation type, the type of situation with supports that infon, HPSG.

In HPSG the value of the CONTENT attribute may be interpreted as a situation type. Generally this situation type will be given in terms of an infon, and the situation type will be the type of situation which supports that infon. Quantification yields a different sort of situation type. Given that determiners correspond to binary types that hold of two object types, abstraction allows a sentence such as 'most kittens are miaowing' to be taken to correspond to the situation type:

$$[S \mid (([x \mid S_r \models \langle\langle \text{kitten}, x; + \rangle\rangle], [x \mid S \models \langle\langle \text{miaows}, x; + \rangle\rangle]) : \text{most})]$$

'Most' may thus be associated with the type:

$$[S \mid (([x \mid S_r \models \langle\langle P, x; + \rangle\rangle], [x \mid S \models \langle\langle Q, x; + \rangle\rangle]) : \text{most})]$$

where P is to be unified with the property contributed by the following noun and Q is a property contributed by the rest of the sentence. Some further mechanism is required to determine this property.

In CPSG the value of the CONTENT attribute is interpreted as a proposition. Thus the abstraction required by HPSG is not required by CPSG. Determiners may be translated directly as propositions with uninstantiated arguments. *Most* might therefore be translated as:

$$(T_1, T_2 : \text{most})$$

where T_1 is unified with the type contributed by the noun phrase and T_2 is constructed from the proposition expressed by the rest of the sentence by abstracting over the relevant argument role. The details of this approach are spelled out further in section 4, where the account is complicated by the issue of quantifier scoping.

2 Quantification in HPSG

As mentioned above, HPSG treats quantification in terms of quantified infons. The content of a quantified noun phrase is a bound variable, such as

$$(\text{most } x \mid \langle\langle \text{kitten, instance: } x; + \rangle\rangle)$$

or in attribute value form:

$$\begin{bmatrix} \text{DET} & \textit{most} & & \\ \text{IND} & \begin{bmatrix} \text{VAR} & \boxed{1} & \\ \text{RESTRICTION} & \begin{bmatrix} \text{RELN} & \textit{kitten} \\ \text{INSTANCE} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Quantifier terms such as these are constructed from the semantic content of the noun and determiner via the Semantics Principle, which we discuss below. In building the semantic content of an entire sentence, the quantifier term is then combined with an infon containing a variable, binding that variable. The result, which might be written in situation theoretic notation as:

$$(\text{most } x \mid \langle\langle \text{kitten, instance: } x; + \rangle\rangle)\ \langle\langle \text{miaowing, miaower: } x; + \rangle\rangle$$

corresponds to the attribute value matrix:

$$\begin{bmatrix} \text{QUANT} & \begin{bmatrix} \text{DET} & \textit{most} & & \\ \text{IND} & \begin{bmatrix} \text{VAR} & \boxed{1} & \\ \text{RESTRICTION} & \begin{bmatrix} \text{RELN} & \textit{kitten} \\ \text{INSTANCE} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{SCOPE} & \begin{bmatrix} \text{RELN} & \textit{miaowing} \\ \text{MIAOWER} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

This structure is constructed from the content of the noun phrase and the content of the remainder of the sentence, again via the Semantics Principle.

The Semantics Principle, simplified for the unary complement case, is stated in [Pollard & Sag 87] as:

$$\textit{headed-phrase}[\,] \Rightarrow \left[\begin{array}{ll} \text{SEM} & \left[\begin{array}{l}\text{CONTENT}\ \ \text{comb-sem}(\boxed{1},\boxed{2}) \\ \text{CONTEXT}\ \ \text{collect-indices}(\boxed{3})\end{array}\right] \\ \text{DTRS}\ \boxed{3} & \left[\begin{array}{ll}\text{HEAD-DTR} & \left[\text{SEM}\ \left[\text{CONTENT}\ \boxed{1}\right]\right] \\ \text{COMP-DTRS} & \left\langle\left[\text{SEM}\ \left[\text{CONTENT}\ \boxed{2}\right]\right]\right\rangle\end{array}\right]\end{array}\right]$$

The Semantics Principle (Version 3)

The function *comb-sem* (called *combine semantics* in [Pollard & Sag 87]) is defined so as to construct and return the appropriate quantified circumstance if its arguments are of the appropriate sorts, or to simply return its first argument (i.e., the SEM|CONTENT of the head daughter) otherwise. That is:

$$\text{comb-sem}(A, B) = \begin{cases} \left[\begin{array}{ll}\text{QUANT} & B \\ \text{SCOPE} & A\end{array}\right] & \text{if } A \text{ has sort } \textit{circumstance} \text{ and } B \text{ has sort } \textit{quantifier} \\ A & \text{otherwise} \end{cases}$$

comb-sem thus has two clauses, different clauses applying to the combination of proper names with verb phrases and the combination of noun phrases containing common nouns with verb phrases.

[Pollard & Sag 87] generalise this principle to allow for multiple complements:

$$\textit{headed-phrase}[\,] \Rightarrow \left[\begin{array}{ll} \text{SEM} & \left[\begin{array}{l}\text{CONTENT}\ \ \text{suc-comb-sem}(\boxed{1},\boxed{2}) \\ \text{CONTEXT}\ \ \text{collect-indices}(\boxed{3})\end{array}\right] \\ \text{DTRS}\ \boxed{3} & \left[\begin{array}{ll}\text{HEAD-DTR} & \left[\text{SEM}\ \left[\text{CONTENT}\ \boxed{1}\right]\right] \\ \text{COMP-DTRS} & \boxed{2}\end{array}\right]\end{array}\right]$$

The Semantics Principle (Version 4)

suc-comb-sem (or *successively-combine-semantics*) can be defined recursively and in terms of *comb-sem* as follows:

$$\text{suc-comb-sem}(A, L) = \begin{cases} A & \text{if } L = \langle\,\rangle \\ \text{suc-comb-sem}(\text{comb-sem}(A, \boxed{1}), T) & \\ \quad \text{if } L = \langle \text{SEM}|\text{CONTENT } \boxed{1} \mid T \rangle & \end{cases}$$

This definition is suitable for headed structures with none, one, or several complement daughters.

Although version 4 is the final version of the Semantics Principle presented in [Pollard & Sag 87], it does not allow for scoping ambiguities, producing only what Pollard & Sag refer to as the "natural scope" reading. This points up a serious problem in the HPSG treatment of quantification. As a first approximation to allowing scoping ambiguities, we could try making *suc-comb-sem* non-deterministic (or relational, rather than functional), and not dependent on the order of the complements. Thus we could replace the above definition with something like:

$$\text{suc-comb-sem}(A, L) = \begin{cases} A & \text{if } L = \langle\,\rangle \\ \text{suc-comb-sem}(\text{comb-sem}(A, \boxed{1}), R) & \\ \quad \text{if delete}([\text{SEM}|\text{CONTENT } \boxed{1}], L, R) & \end{cases}$$

where *delete/3* holds if and only if its second argument is a list, its first argument is some member of that list and its third argument is the list resulting from deleting the first argument from the second argument.

This approach is inherently flawed, however, as, since the last complement is combined with the head after the other complements, rather than at the same time, *suc-comb-sem* cannot scope the semantic content of the last complement with respect to the other complements. Thus the last complement will always have wide scope. In a sentence such as 'some bear gave two thistles to every donkey' there are six possible readings corresponding to the six possible permutations of the quantifiers. The schematic syntactic structure imposed by HPSG's grammar rules ensures that, with the current treatment of semantics, no amount of tinkering with *suc-comb-sem* will allow all six of these permutations to be realised, as the semantic content of the verb phrase 'gave two thistles to every donkey' will always be determined as a single unit over which *some bear* will have scope. Thus with the current mechanism for quantifier scoping, the only two semantic contents that shall be obtainable are:

(some b | ⟪bear, instance: b; +⟫)
(two t | ⟪thistle, instance: t; +⟫)
(every d | ⟪donkey, instance: d; +⟫)
⟪gave, giver: b, given: t, receiver: d; +⟫

(some b | ⟪bear, instance: b; +⟫)
(every d | ⟪donkey, instance: d; +⟫)
(two t | ⟪thistle, instance: t; +⟫)
⟪gave, giver: b, given: t, receiver: d; +⟫

The other four scopings require either that the subject be able to combine with the verb at the same stage as the other complements, or that a radically different method be developed for determining the semantic content of quantified phrases. Such a method should, presumably, also allow the legitimate (and only the legitimate) quantifier scoping ambiguities that arise with quantifiers nested in constructs (cf. [Hobbs & Shieber 87], [Keller 87], [Lewin 90]).

3 Quantification in 2–14–89

[Pollard 89b], in describing a theory he calls 2–14-89, a descendant of HPSG, addresses the problem of licensing correctly scoped quantified infons. His solution involves a QSTORE attribute and two Principles, the Qstore Principle and the Content Principle. The following is a modified version of that proposal. For simplicity we ignore the SEMANTICS|CONTEXT attribute.

We add another attribute, the QSTORE attribute, to the value of the SEMANTICS attribute of a sign. The value of this attribute is a set of quantified circumstances. We also replace the Semantics Principle by the Qstore And Content Principle, which states that

> the SEMANTICS|CONTENT value of a phrase is the result of nondeterministically retrieving any number of quantified circumstances (possibly none) from the SEMANTICS|QSTORE and "quantifying into" the SEMANTICS|CONTENT value of the head-daughter, and the SEMANTICS|QSTORE value of a phrase is

the union of the values of the SEMANTICS|QSTORE attributes of the daughters less those quantified circumstances that are explicitly retrieved.

This Principle can be formulated in terms of a single feature structure with a few functional dependencies:

$$\left[\begin{array}{ll} \text{SEMANTICS} & \left[\begin{array}{ll} \text{CONTENT} & \text{quantify-in}(\boxed{2}, \boxed{3}) \\ \text{QSTORE} & \text{difference}(\text{quants}(\boxed{1}), \boxed{2}) \end{array}\right] \\ \text{DTRS} & \boxed{1}\left[\text{HEAD-DTR|SEMANTICS|CONTENT } \boxed{3}\right] \end{array}\right]$$

The Qstore and Content Principle

where [2] is some arbitrary subset of the quantifiers in [1].

The three functions involved can be informally described as follows:

quants returns the set union of the values of the SEMANTICS|QSTORE attributes of each of the daughters.

difference takes two sets as arguments, where the second is a subset of the first, and returns the set consisting of all those elements that belong to the first set but not the second.

quantify-in takes two arguments such that either the second is of sort *indexed object* and the first is the empty set, or the second is of sort *circumstance* and the first is a set whose elements are all of sort *quantified circumstance*. In the first case, the result is the indexed object, but in the second it is the result of quantifying each quantified circumstance in the set (in an indeterminate order) into the original circumstance.

The result of quantifying a quantified circumstance Q into a circumstance C is the quantified circumstance that results from unifying Q with [SCOPE C]. The indeterminacy in the function *quantify-in* suggests that we are really dealing with a relational dependency, and reaffirms the need to generalise the use of functional dependencies to relational dependencies.

Note that when [2] is the empty set, the Qstore and Content Principle reduces to:

$$\begin{bmatrix} \text{SEMANTICS} & \begin{bmatrix} \text{CONTENT} & \boxed{3} \\ \text{QSTORE} & \text{quants}(\boxed{1}) \end{bmatrix} \\ \text{DTRS} & \boxed{1}\begin{bmatrix}\text{HEAD-DTR}|\text{SEMANTICS}|\text{CONTENT} & \boxed{3}\end{bmatrix} \end{bmatrix}$$

i.e., the semantic content is just the semantic content of the head and the set of unscoped quantifier terms in the whole constituent is just the set union of the sets of unscoped quantifier terms in the subconstituents.

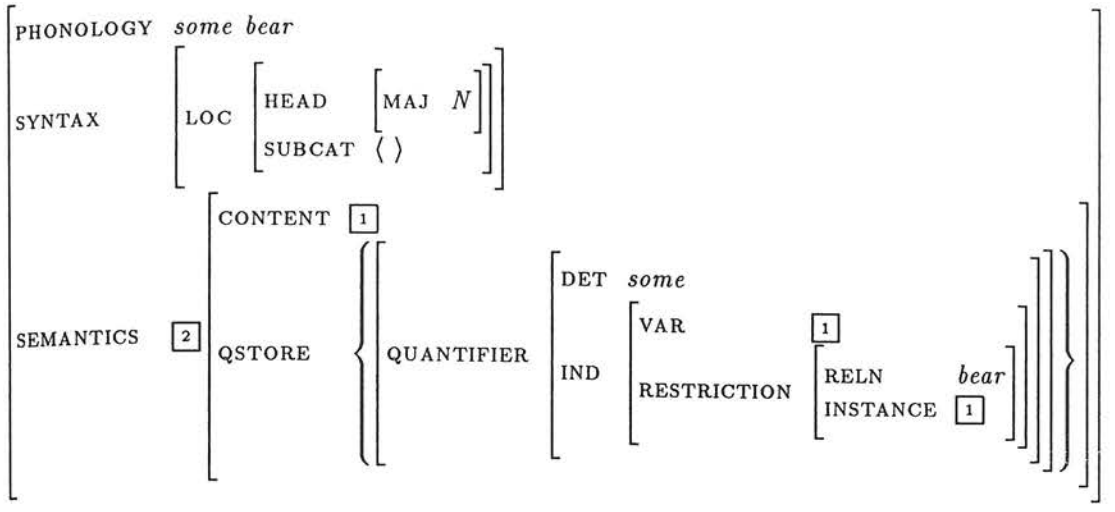
We treat all determiners as quantifiers, the lexical entries for which look like the following:

$$\begin{bmatrix} \text{PHONOLOGY} & \textit{some} \\ \text{SYNTAX} & \begin{bmatrix}\text{LOC}\begin{bmatrix}\text{HEAD}\begin{bmatrix}\text{MAJ} & D\end{bmatrix}\end{bmatrix}\end{bmatrix} \\ \text{SEMANTICS} & \begin{bmatrix} \text{CONTENT} & \boxed{1} \\ \text{QSTORE} & \left\{\begin{bmatrix}\text{QUANTIFIER} & \begin{bmatrix}\text{DET} & \textit{some} \\ \text{IND} & \begin{bmatrix}\text{VAR} & \boxed{1}\end{bmatrix}\end{bmatrix}\end{bmatrix}\right\} \end{bmatrix} \end{bmatrix}$$

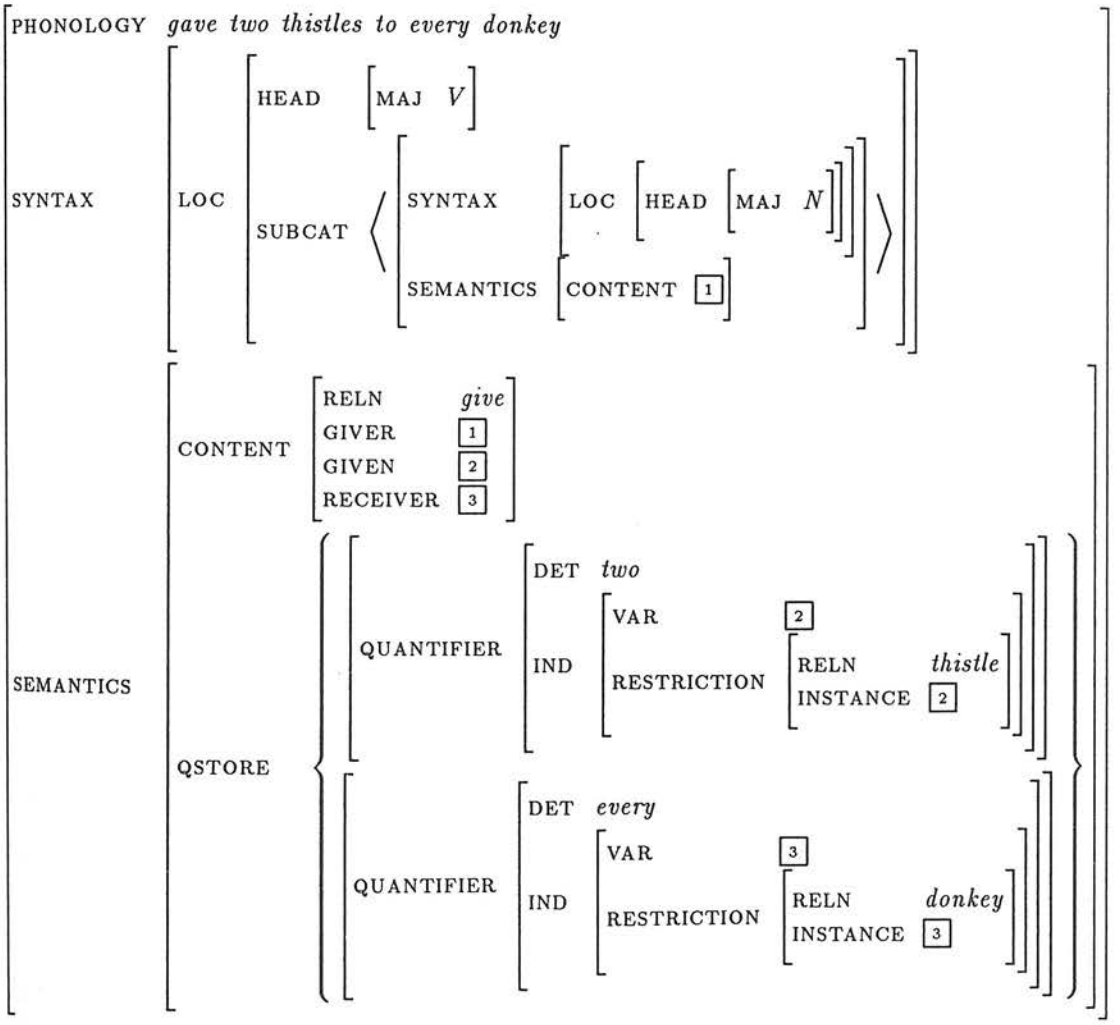
Lexical entries for nouns follow the following pattern:

$$\begin{bmatrix} \text{PHONOLOGY} & \textit{bear} \\ \text{SYNTAX} & \begin{bmatrix}\text{LOC}\begin{bmatrix}\text{HEAD} & \begin{bmatrix}\text{MAJ} & N\end{bmatrix} \\ \text{SUBCAT} & \left\langle\begin{bmatrix}\text{SYNTAX} & \begin{bmatrix}\text{LOC}\begin{bmatrix}\text{HEAD}\begin{bmatrix}\text{MAJ} & D\end{bmatrix}\end{bmatrix}\end{bmatrix} \\ \text{SEMANTICS} & \boxed{2}\end{bmatrix}\right\rangle\end{bmatrix}\end{bmatrix} \\ \text{SEMANTICS} & \boxed{2}\begin{bmatrix} \text{CONTENT} & \boxed{3} \\ \text{QSTORE} & \left\{\begin{bmatrix}\text{QUANTIFIER} & \begin{bmatrix}\text{IND} & \begin{bmatrix}\text{VAR} & \boxed{3} \\ \text{RESTRICTION} & \begin{bmatrix}\text{RELN} & \textit{bear} \\ \text{INSTANCE} & \boxed{3}\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix}\right\} \end{bmatrix} \end{bmatrix}$$

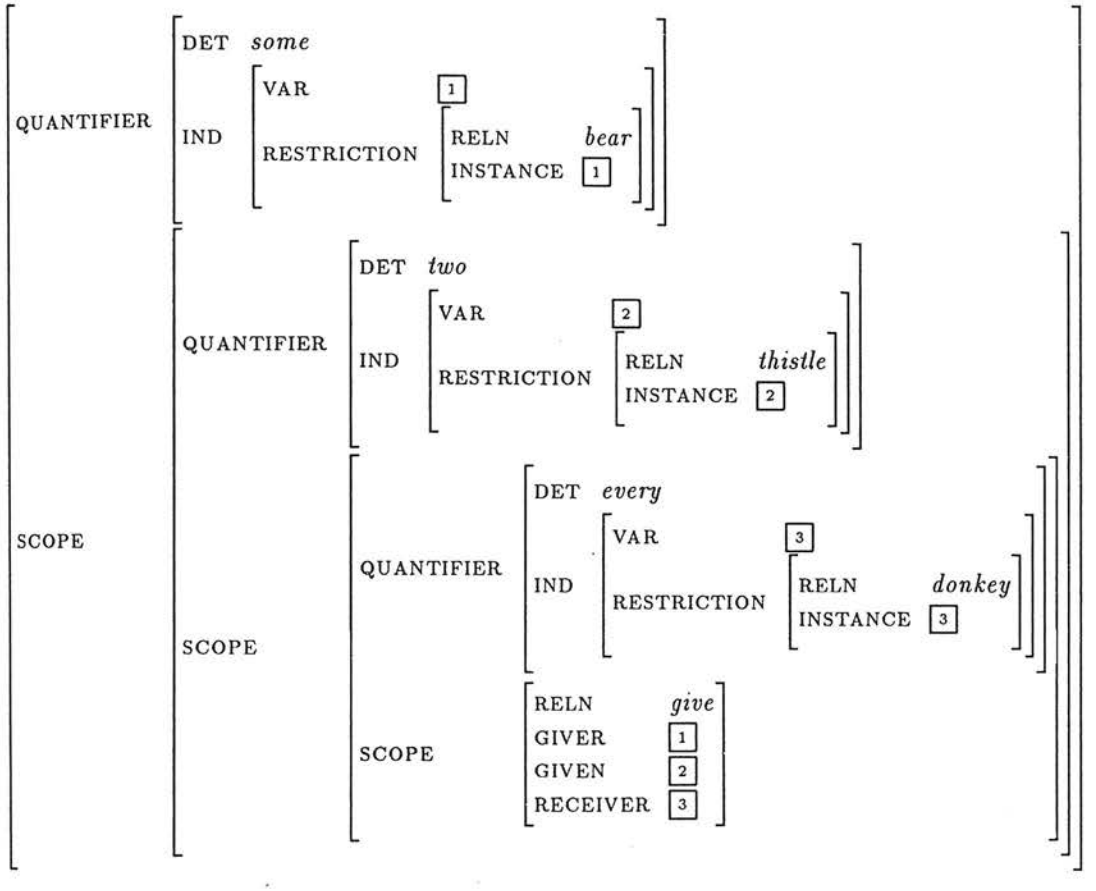
The QSTORE attribute is lexically specified as being the empty set for all other lexical items. When the subcategorisation requirements of the noun are met by the determiner, all the 'gaps' in the SEMANTICS|QSTORE attribute are filled, and $\boxed{1}$ and $\boxed{3}$ are unified, resulting in:



Following this treatment of quantification, we are lead to the following sign for 'gave two thistles to every donkey', where in building the phrase we have chosen not to retrieve any quantifiers from the store.



A condition on a phrasal sign modelling a sentence is that the value of its QSTORE attribute be the empty set. That is, that no quantifiers remain in store. This, of course, does not apply to sentential clauses embedded within other clauses. Thus, when we combine this phrase with a subject we must retrieve all quantifiers (i.e., two from the above sign, which plays the role of the head daughter, and possibly some from the complement daughter (i.e., the subject)). These quantifiers may be quantified into the SEMANTICS|CONTENT attribute of the head in any order. If the subject is *some bear* and we take the order *some-two-every*, then the quantified circumstance that results is:



Note that it is always consistent with the Qstore And Content Principle to retrieve all quantifiers at the last instance. All quantifier scopings can be generated by always storing quantifiers when the rule is not licensing a (non-embedded) sentence. This raises the possibility of separating out the scoping algorithm from the algorithm for storing the quantifiers, and having instead a principle for interpreting a semantic representation in which all quantifiers are stored. This is basically the approach adopted by CPSG.

In closing, Pollard points out that this scoping algorithm suffers from problems when dealing with nested quantification. The Qstore And Content Principle is a direct im-

plementation of Cooper storage ([Robin Cooper 75]), which generates ill-formed representations, representations containing unbound variables, when applied to constructions involving nested quantification (such as 'every pet of some kitten'). [Pollard 89b, p. 182] is pessimistic about this result, stating:

> This is a peculiar state of affairs, because it is the only kind of ill-formedness in the whole theory that is not a case of unification failure. Instead the candidate CONTENT value ... is ruled out because it is a syntactically ill-formed formula in some variety of restricted quantifier logic. This may point to an essential limitation in the power of unification-based grammar formalisms. As Fernando Periera (p.c.) and others have observed, it does not seem to be within the power of such formalisms to model abstraction or variable binding operations in a natural way. To put it another way, there may well be some things that can be done with lambdas that cannot be done with unification.

Whilst the conclusion may well be correct, it does appear that abstraction and variable binding operations cannot be modelled in a natural way, there are other scoping algorithms which could be applied to avoid the unbound variable problem in this instance. In particular, there are the algorithms of [Hobbs & Shieber 87] and [Keller 87]. These algorithms include an explicit free variable constraint to prevent the generation of ill-formed formulae. A better algorithm is that of [Lewin 90], which is sound and complete but does not appeal to any kind of free variable constraint. This is the algorithm employed by CPSG.

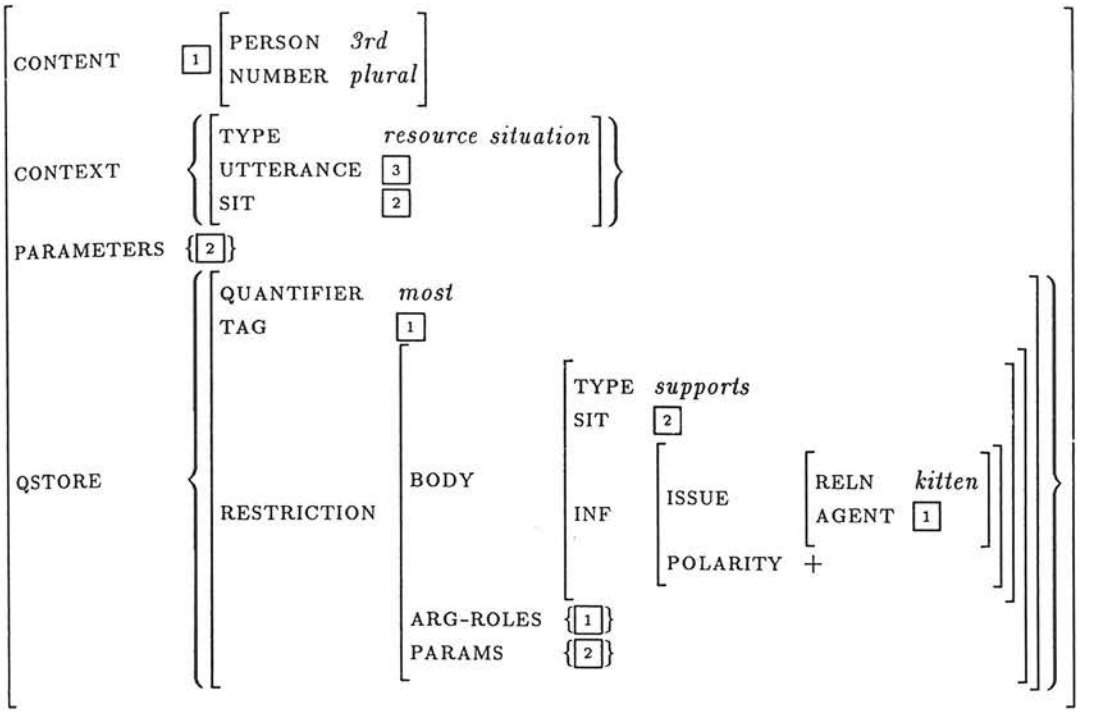
4 Quantification in CPSG

The treatment of quantification in CPSG follows that of 2–14–89 with three major exceptions. Firstly, quantification is treated in terms of types as advocated in section 1. Secondly, the relative scoping of quantifiers is taken to be determined by a global resource situation which the entire utterance exploits. Thirdly, the algorithm of [Lewin 90] is employed to generate all and only the correct quantifier scopings. As discussed below this third change has far reaching consequences.

4.1 Determiners and Noun Phrases in CPSG

Following 2–14–89, we take determiner-noun constructions to have a parameter as their semantic content, with a term in the QSTORE set representing the quantifier. The noun

phrase 'most kittens' has as the value of its SEMANTICS attribute:



where 3 is the value of the PHONOLOGY attribute.

The value of the QSTORE attribute is a set whose elements have three attributes, one indicating the quantifier, one which tags the position where the quantifier belongs in the superordinate structure, and one whose value is the type which acts as the restriction on the quantifier. Note that we do not consider the content, 1, to be a parameter of the semantic content.

As in HPSG, there are several possible treatments of determiners and nouns which are consistent with this treatment of noun phrases. We have already seen (section 3.3 of Chapter 7) that CPSG takes the semantic content of common nouns to be a type. The other semantic attributes follow the pattern in:

$$\left[\begin{array}{ll}
\text{PHON} & \boxed{4}\ \langle \textit{kittens} \rangle \\
\text{SYN} & \left[\text{LOC}\ \left[\text{SPEC}\ \text{DET}_{\boxed{3}}\right]\right] \\
\text{SEM} & \left[\begin{array}{ll}
\text{CONTENT} & \boxed{1} \\
\text{CONTEXT} & \left\{\left[\begin{array}{ll}\text{TYPE} & \textit{resource situation} \\ \text{UTTERANCE} & \boxed{4} \\ \text{SIT} & \boxed{2}\end{array}\right]\right\} \\
\text{PARAMETERS} & \{\boxed{2}\} \\
\text{QSTORE} & \left\{\boxed{3}\left[\begin{array}{ll}\text{QUANTIFIER} & \\ \text{TAG} & \left[\begin{array}{ll}\text{PERSON} & \textit{3rd} \\ \text{NUMBER} & \textit{plural}\end{array}\right] \\ \text{RESTRICTION} & \boxed{1}\end{array}\right]\right\}
\end{array}\right]
\end{array}\right]$$

where [1] is the type of object that has the property of kittenhood in the resource situation [2].

The element of the QSTORE set is coindexed with the semantic content of the subcategorised for determiner, so that when the subcategorisation requirement is met, the appropriate quantifier will be unified into the appropriate term on the QSTORE set. Note also that it is the actual element of the set that is coindexed, so if the set happens to have more than one element when the determiner and $\overline{\text{N}}$ combine, the correct element will be unified with the semantic content of the determiner.

The description of the determiner 'most' extends:

$$\left[\begin{array}{ll}
\text{PHON} & \langle \textit{most} \rangle \\
\text{SEM} & \left[\begin{array}{ll}
\text{CONTENT} & \left[\begin{array}{ll}\text{QUANTIFIER} & \textit{most} \\ \text{TAG} & \left[\begin{array}{ll}\text{PERSON} & \textit{3rd} \\ \text{NUMBER} & \textit{plural}\end{array}\right] \\ \text{RESTRICTION} & \end{array}\right] \\
\text{CONTEXT} & \{\,\} \\
\text{PARAMETERS} & \{\,\} \\
\text{QSTORE} & \{\,\}
\end{array}\right]
\end{array}\right]$$

Note here that the CONTEXT, PARAMETERS and QSTORE are all empty. Note also how the TAG attribute, which corresponds to a parameter, encodes agreement features, both in the case of nouns and in the case of determiners. This, combined with the rule

for combining determiners and nouns, which requires that the semantic content of the determiner be shared with the element in the noun's QSTORE set, ensures determiner-noun agreement.

Two problems remain. We need to correctly account for nested constructions: noun phrases which are modified or which take complements which may contain determiners. On the whole we ignore this complication. We also need to build the semantic content of the noun phrase, the tag associated with the quantifier term, from the determiner and $\overline{\text{N}}$. We cannot simply take the semantic content of the phrase to be the semantic content of the head. This problem is catered for by the semantic principles discussed below.

4.2 The Global Resource Situation

In CPSG, the scoping of quantifiers is taken to be determined by context. In one context one scoping might be preferred, whilst in another a different scoping might be preferred. Pragmatics might bias the preference of some scopings, but even this can be seen to be determined by contextual factors: world knowledge can be analysed as part of a context. We have adopted the situation semantic view that individual noun phrases exploit resource situations. The relative scoping of quantifiers requires a global resource situation: a resource situation that refers to all noun phrases stating the scoping relationships that hold between them. This global resource situation plays much the same role as the *circumstances* in the situation semantic account of anaphora and quantification given by [Gawron & Peters 90a].

Recall from section 1.6 the makeup of the global resource situation in terms of individual resource situations for the subutterances (utterances of subconstituents) of an utterance. Given that these resource situations support infons concerning the relative scoping of quantifier terms, these relationships, together with persistence, may be taken to justify either the partial application of the scoping algorithm to incomplete sentences, or the full application of the scoping algorithm to complete sentences: persistence will ensure that scoping facts do not change in larger situations.

4.3 Semantic Principles

There is a question for any quantifier scoping algorithm concerning whether the scoping should be done "on the fly", or all at once. The algorithms of [Hobbs & Shieber 87], [Keller 87] and [Lewin 90] all take as input an unscoped form, from a special input language, and transform this into a scoped form. Scoping is not done on the fly. In 2-14-89, scoping is done on the fly, and as a consequence of this one scoping may be generated a number of times by the application of the rules at different stages in the phrase structure tree. Unlike the algorithms of Hobbs and Shieber and Keller, Lewin's algorithm is very sensitive to the all at once/on the fly distinction. Unlike the other algorithms it does not generate partially scoped representations. The consequences of this for CPSG are that we really need to generate an unscoped representation for entire sentences and then apply the scoping algorithm. Alternately we may just generate unscoped representations and incorporate a Principle of Semantic Interpretation, to interpret such representations. We consider first the generation of complete unscoped representations.

4.3.1 Percolation of Semantic Features

In most cases, we only need to ensure that the values of all semantic attributes of daughters correctly percolate to their mothers. In all cases the PARAMETERS, CONTEXT and QSTORE attributes of a mother are just the set union of the corresponding attributes on each of the daughters. Furthermore, the CONTENT of most phrases is just the CONTENT of their head. It is only the CONTENT of noun phrases which must be specially constructed.

If the head of a head/specifier phrase is an $\overline{\text{N}}$, then the description of the phrase extends:

$$\begin{bmatrix} \text{SEM|CONTENT} & \boxed{1} \\ \text{DTRS|TOPIC-DTR|SEM|CONTENT|TAG} & \boxed{1} \end{bmatrix}$$

The descriptions of all other head/argument phrases extend:

$$\begin{bmatrix} \text{SEM|CONTENT} & \boxed{1} \\ \text{DTRS|HEAD-DTR|SEM|CONTENT} & \boxed{1} \end{bmatrix}$$

4.3.2 Semantic Interpretation

We began Chapter 7 with a consideration of how situation theoretic objects might be described by feature structures. The need for a Principle of Semantic Interpretation, a principle to allow the interpretation of feature structures with non-empty QSTORE attributes, causes us to consider how feature structures may be interpreted as situation theoretic objects.

We define a map *interp* from feature structures corresponding to the value of a SEMANTICS attribute to a feature structure describing a proposition as follows:

$$interp\left(\begin{bmatrix}\text{CONTENT} & \boxed{1}\\ \text{QSTORE} & \{\,\}\end{bmatrix}\right) = \boxed{1}$$

and

$$interp\left(\begin{bmatrix}\text{CONTENT} & \boxed{1}\\ \text{QSTORE} & \left\{\boxed{q_1},\ldots,\boxed{q_n}\right\}\end{bmatrix}\right) = \begin{bmatrix}\text{TYPE} & \boxed{3}\\ \text{RESTRICTION} & \boxed{5}\\ \text{SCOPE} & \begin{bmatrix}\text{BODY} & \boxed{6}\\ \text{ARG-ROLES} & \{\boxed{4}\}\\ \text{PARAMS} & \{\,\}\end{bmatrix}\end{bmatrix}$$

where the global resource situation specifies that $\boxed{q_1}$ takes scope over all other elements of the QSTORE set,

$$\boxed{q_1} = \begin{bmatrix}\text{QUANTIFIER} & \boxed{3}\\ \text{TAG} & \boxed{4}\\ \text{RESTRICTION} & \boxed{5}\end{bmatrix}$$

and

$$\boxed{6} = interp\left(\begin{bmatrix}\text{CONTENT} & \boxed{1}\\ \text{QSTORE} & \left\{\boxed{q_2},\ldots,\boxed{q_n}\right\}\end{bmatrix}\right)$$

In the case of a one element QSTORE this gives:

$$interp\left(\begin{bmatrix}\text{CONTENT} & \boxed{1}\\ \text{QSTORE} & \left\{\begin{bmatrix}\text{QUANT} & \boxed{3}\\ \text{TAG} & \boxed{4}\\ \text{REST} & \boxed{5}\end{bmatrix}\right\}\end{bmatrix}\right) = \begin{bmatrix}\text{TYPE} & \boxed{3}\\ \text{RESTRICTION} & \boxed{5}\\ \text{SCOPE} & \begin{bmatrix}\text{BODY} & \boxed{1}\\ \text{ARG-ROLES} & \{\boxed{4}\}\\ \text{PARAMS} & \{\,\}\end{bmatrix}\end{bmatrix}$$

Given the value of the SEMANTICS attribute of the description of some constituent, the function *interp* allows us to interpret that value as a feature structure describing a situation theoretic proposition. Note that this function does not cater for nested quantification.

4.4 Discussion

The above treatment leaves a number of points outstanding. The issue of scoping nested quantifiers is not resolved, and it would appear that extensive modifications would be required to provide a satisfactory account. Furthermore we have said nothing about how quantification interacts with other predicates, such as negation or predicates that involve intensional contexts.

With regard to nested quantification, the principal stumbling block for the above account concerns the representation of quantifiers that arise from the restriction. At present there is no clear way to represent such quantifiers. A possible approach to this issue would be to generalise from the start the feature structure description of a proposition, so that *all* propositions include a QSTORE attribute. Thus, the type corresponding to the restriction may be based on a proposition containing unscoped quantifier terms. Adopting this approach would also mean that the QSTORE attribute of the SEMANTICS attribute of the description of a constituent would be incorporated into the CONTENT of that constituent, and we would again be left with the original three SEMANTICS attributes. The interpretation function *interp* would clearly also have to be adjusted.

With regard to negation the simplest treatment leads only to verb phrase negation, and not sentence negation. That is, for a sentence such as 'every kitten didn't miaow', the only reading given would be that in which 'not' has scope only over the verb phrase, describing a situation in which every kitten has the property of not miaowing. To allow sentence negation, a theory of opaqueness for the argument roles of predicates is required. Such a theory is incorporated in the scoping algorithms of [Hobbs & Shieber 87], [Keller 87] and [Lewin 90], but it is unclear just how the theory might be incorporated into the Principle of Semantic Interpretation of CPSG. The treatment of intensional contexts is an even greater problem. Very little work has been done with regard to the problems of intensional contexts in situation semantics (though see [Robin Cooper 87]), and we do not attempt to broach the question here.

5 Distributive Readings of Coordinated Noun Phrases

The treatment of quantification may easily be extended to give distributive readings for noun phrases. We add a further attribute to the value of the SEMANTICS attribute, the CSTORE attribute, which takes as its value a set of *coordinate terms*. These terms are interpreted by the Principle of Semantic Interpretation in a way similar to quantifier terms.

Coordinate terms take the form:

$$\begin{bmatrix} \text{CONJ} & or \\ \text{TAG} & \boxed{1} \\ \text{ARGS} & \{\ldots\} \end{bmatrix}$$

where the value of CONJ is the type corresponding to the conjunction, the value of TAG marks the relevant argument role in the proposition and the value of ARGS is the set of parameters corresponding to the individuals coordinated.

We add further clauses for *interp* along the lines of:

$$interp\left(\begin{bmatrix} \text{CONTENT} & \boxed{1} \\ \text{QSTORE} & \{\,\} \\ \text{CSTORE} & \left\{\begin{bmatrix} \text{CONJ} & \boxed{3} \\ \text{TAG} & \boxed{4} \\ \text{ARGS} & \boxed{5} \end{bmatrix}\right\} \end{bmatrix}\right) = \begin{bmatrix} \text{TYPE} & \boxed{3} \\ \text{ARGS} & \boxed{5} \\ \text{SCOPE} & \begin{bmatrix} \text{BODY} & \boxed{1} \\ \text{ARG-ROLES} & \{\boxed{4}\} \\ \text{PARAMS} & \{\,\} \end{bmatrix} \end{bmatrix}$$

This clause tells how to interpret a description with no quantifier terms and a single coordinate term. Such clauses are interpreted as propositions which take the form:

$$(\{\alpha, \beta, \ldots, \zeta\}, [\dot{x} \mid \tau(\dot{x})] : or)$$

where *or* is interpreted as a binary type that holds between a set of objects and a type if and only if one of those objects is of the type. (This is a different type than that countenanced in section 1.3.2.)

For this to work, we need to ensure that the value of the SEMANTICS attribute for distributively coordinated noun phrases follows the pattern of:

$$\begin{bmatrix} \text{CONTENT} & \boxed{1} \\ \text{CSTORE} & \left\{ \begin{bmatrix} \text{CONJ} & \alpha \\ \text{TAG} & \boxed{1} \\ \text{ARGS} & \left\{ \boxed{c_1}, \ldots, \boxed{c_n} \right\} \end{bmatrix} \right\} \\ \text{QSTORE} & \{\,\} \\ \text{PARAMETERS} & \{\,\} \end{bmatrix}$$

where α is the type corresponding to the relevant conjunction, $\boxed{1}$ is a parameter serving to tag the relevant argument role, and $\boxed{c_1} \ldots \boxed{c_n}$ are the parameters corresponding to the individual conjuncts. The agreement features of $\boxed{1}$ must be consistent with those of the parameters corresponding to the individual conjuncts and the type of conjunction. Thus a relational dependency must hold between the values of each of the attributes of each element of the CSTORE set. Designating this ternary relation *coord-agreement*, we require *coord-agreement*$(\alpha, \{\boxed{c_1}, \ldots, \boxed{c_n}\}, \boxed{1})$ to hold in the above case.

This treatment only allows for coordination of proper names and pronouns. It must be generalised further to allow coordination of quantified noun phrases. The generalisation would presumably involve a further generalisation to the Principle of Semantic Interpretation such that quantified terms within the ARGS attribute of a coordinate term could be interpreted.

6 Summary

In this chapter we have concentrated on the issue of quantification. This began with a situation theoretic argument concerning the treatment of quantification in situation semantics. The argument, that structurally determined "facts" should be treated as propositions rather than as infons, concerns more than just the treatment of quantification.

For completeness, the treatment of quantification in HPSG was then presented, together with the problems it faces in handling quantifier scope. Quantification in 2–14–89 was also considered. This treatment solves many of the problems of the original HPSG treatment, and the treatment of quantification in CPSG owes much to this theory. Even so,

quantification in CPSG is rather different than in 2–14–89. The theory itself yields unscoped representations which must then be scoped according to a Principle of Semantic Interpretation.

Finally, some residual problems with quantification concerning nested quantification and intensional contexts were discussed, before the related issue of the semantics of distributive readings of coordinated noun phrases was resolved.

Chapter 9

Parsing CPSG

The use of the language of feature structures by CPSG not only allows for a degree of precision not generally found in grammatical theories, but also makes CPSG very amenable to computational implementation. This chapter is principally concerned with the problem of parsing a CPSG grammar. We begin by examining previous work related to HPSG. Several implementations of part or all of HPSG have been developed over the past few years, and these are of obvious relevance to any implementation of CPSG. There has also been some work on "head-driven" parsing that is unrelated to HPSG. This work has grown out of a more general movement emphasising the importance of the head of a constituent. We briefly review some of this work in section 1. Section 2 examines how the use of a constituent hierarchy by HPSG can be employed to yield an efficient rule application strategy which involves traversing the constituent hierarchy. This strategy is most easily applied as part of an undirected algorithm, and we discuss two orthogonal binary dimensions along which such strategies may vary, along with a Prolog implementation illustrating the possible approaches. Sections 3, 4 and 5 relate specifically to the implementation and the issues raised. Section 3 concentrates on a number of specific areas of the implementation where unusual techniques have been employed to improve the efficiency of the code. In section 4, lexical organisation is discussed, with special attention to the compilation of the lexical hierarchy and the application of lexical rules. Finally, section 5 discusses the user interface to the implementation and illustrates its use.

1 Existing Algorithms and Implementations for HPSG

There are several algorithms and implementations in existence for HPSG-like grammar formalisms. The original implementation, a "head-driven" active chart parser, was developed at Hewlett-Packard, Palo Alto, where much of the original work on HPSG was conducted. Other implementations include the chart parser of [McIntyre 89], and implementations along the lines of recursive unification as suggested in Pollard's more recent work ([Pollard 89a]).

1.1 Head-Driven Parsing

[Proudian & Pollard 85] describe an implementation of an active chart parser for an early version of HPSG in which the basic strategy involves first locating the head of a phrase and using this, with the relevant grammar rule, to parse the phrase it heads. The details of the parser are somewhat obscure, though it seems that the algorithm is similar to that employed independently by [McIntyre 89], which we discuss below.

In some respects it seems that this implementation is somewhat dated, and not a true reflection of the current state of HPSG. In particular it seems that the implementation favours atomic valued attributes, and it is not clear how "category"-valued attributes (such as SUBCAT or SLASH) are treated. Nevertheless, the algorithm developed, with its emphasis on first finding the head of a phrase, is applicable to HPSG as it currently stands, as well as to CPSG. Some further details of the implementation are provided by [Flickinger *et al.* 85], who discuss the lexicon, and [Proudian & Goddeau 87], who discuss the implementation of a theory of coordination.

In unrelated work, [Satto & Stock 89] present a tabular method for "head-driven bidirectional parsing". The basic algorithm employs (one triangular half of) a matrix T of size $(n+1) \times (n+1)$ to parse a constituent of n words. Each component $T_{i,j}$ of this matrix is initialised to the empty set, and as the algorithm proceeds these sets are augmented with possible analyses of the string between positions i and j. Basically, the algorithm involves working through the matrix and attempting to extend analyses both leftward and rightward. Whilst Satto & Stock do not relate their approach to HPSG, it would appear to be a straightforwarded matter to do so.

There has also been some other work, again unrelated to the Hewlett-Packard project and HPSG itself, by Gibson, on strictly left-right parsing where phrasal nodes are predicted before their heads have been processed ([Gibson 89]). Gibson's strategy involves using constituents of certain categories to trigger hypotheses about their dominating categories. Thus, if in English a determiner is detected, a noun phrase will be hypothesised to dominate that determiner. It is not immediately clear how this strategy might be employed to parse an HPSG-like grammar. Several difficulties are immediate, but most notably, the lexicalist nature of HPSG does not allow such prediction as there is nothing that requires that, for example, only nominal constituents may subcategorise for determiners. The strategy is also at odds with the mechanism within HPSG for determining word order. Nevertheless, this approach might be adopted by first applying some kind of compilation technique to the grammar, determining for each word which constituents it may form a part of. Such an approach would, however, be at odds with HPSG's emphasis on the head.

1.2 McIntyre's Chart Parser

[McIntyre 89] describes an implementation of an HPSG parser as an active chart parser. The implementation represents a fairly straightforward application of an active chart parser. With each edge of the chart is associated a "needs list", which contains those subcategorised for daughters of the head which are not actually spanned by the edge. For inactive edges this needs list is empty. Active edges are by definition those edges whose needs list is non-empty. The needs list is constructed not simply from the grammar rules as in ordinary chart parsing, but from the SUBCAT list of the head together with the appropriate grammar rule. Thus, for example, if a lexical constituent is postulated as a head, and that lexical constituent subcategorises for at least one element, then an active non-lexical constituent will be postulated spanning the same material, such that the head of the non-lexical constituent is the original lexical constituent and the non-lexical constituent satisfies Grammar Rule 2. The needs list will be the list of all daughters which are required but not spanned by the active edge, i.e., all but the last element on the SUBCAT list of the lexical head.

Elements of the needs list are sought on either side of a constituent, so that heads need

not be phrase initial or phrase final. To deal with word order, McIntyre introduces a further attribute, the ORDER attribute, for feature structures of sort *sign*. This attribute takes as its value a feature structure having two attributes START and END whose values indicate the start and end of a constituent (in McIntyre's implementation these attributes are integer valued, with the integers indicating a position in the input string). The value of the ORDER attribute is specified for lexical items when they are originally entered onto the chart, and the English Constituent Order Principle is implemented essentially via feature passing which requires that, for lexically headed phrases the END of the head is the same as the START of the least oblique complement daughter, and for non-lexically headed phrases the START of the head is the same as the END of the most oblique complement daughter. It is unclear why this information cannot be encoded directly through the PHONOLOGY value.

1.3 Recursive Elaboration of Sort Definitions

HPSG's specification of a grammar in terms of sorted feature structures lends itself to an elegant, though it seems inefficient, implementation in terms of recursively expanding sort specifications. A grammar may be formally specified in terms of sorted feature structures by specifying the attributes appropriate for each sort and the sort of their values. Given this, a string with phonology X is a constituent if and only if it can be described by a feature structure of sort *sign* whose PHONOLOGY attribute has value X. Parsing may thus proceed by elaborating this partial description with the information given by the sort definition of *sign*. This sort definition will specify various attributes required of the description, along with the sorts of those attributes' values. The sort definitions may then be applied to each of the values in the elaborated description (some of which may themselves be required to be of sort *sign*), and so on. Mathematically, this solutions corresponds to finding the least fixed point of an operator (defined in [Pollard 89a] as the *denotation* of the grammar) on the Smyth power domain constructed from the compact elements of the lattice of ordinary (unsorted) feature structures ordered by subsumption (see [Pollard 89a] for full details). This approach leads to a sequence of sets of approximations, such that, as the sequence progresses, each approximation is refined until it is a complete description. The set containing all complete descriptions is a fixed point of the operator, and each complete description corresponds to a possible

parse of the initial string. [Pollard 89a] also suggests a similar approach based on the Hoare power domain construction.

This approach bears some superficial similarities to the hierarchy assisted approach which we adopt below, but the approaches are really quite distinct. In particular the recursion in the hierarchy assisted approach does not result from recursively applying sort definitions.

2 Hierarchy Assisted Parsing

The use of a hierarchical classificatory system for structuring a grammar leads to a natural parsing algorithm. In this section we explore this algorithm, together with four strategies which might be adopted in the finer details of implementing such an algorithm.

2.1 The Basic Algorithm

Our use of classification means that a string is a legitimate constituent if and only if it may be classified as being an instance of some type. The use of feature structure descriptions to mediate the *is of type* relation means that a token is an instance of a type if and only if the token's feature structure description is an extension of the type's feature structure description. The use of a hierarchy of types where the immediate subtypes of a type partition that type means that a string is an instance of a type if and only if there is a leaf type such that the string is an instance of every type on the path from the root type to that leaf type. If we add to these facts the assumption that a string with phonology x is a constituent if and only if there exists a description which includes the feature specification [PHON x] and which extends that of some type, then parsing a string with phonology x reduces to finding a path from the root to some leaf such that there exists a feature structure description subsumed by that of each type on the path and by [PHON x]. The existence of such a feature structure description may be proved by its construction, by forming the unification of the initial partial description ([PHON x]) and the feature structure descriptions of each type on the path. If a string is a constituent, then the unification of its original partial description and all feature structure descriptions on the successful path will be a more complete description of the

constituent, simply because given any number of partial descriptions of a token, the unification of those descriptions must also be a description of that token. If there is no path satisfying the requirements, then the string cannot be a legitimate constituent. The act of verifying that a string may be classified thus has the side effect of constructing the description of that string.

Given this formulation of the problem, a "solution" involves traversing the constituent hierarchy to find a suitable path. If only one solution is required (i.e., if only one parse of a string is required) then it may not be necessary to traverse the entire hierarchy. If, however, all solutions (or all parses) are required, then the hierarchy must be traversed in its entirety.

2.2 Traversing the Constituent Hierarchy

When parsing via traversing the constituent hierarchy, there are essentially two independent dimensions along which undirected strategies may vary. Firstly the hierarchy may be traversed in a depth first or breadth first fashion, and secondly with regard to the string we may work in either a top-down fashion or a bottom-up fashion.

2.2.1 Depth First and Breadth First Traversal

According to a depth first strategy, a hierarchy of the form in Figure 9.1 is traversed in numerical node order. We thus begin at the root node (i.e., 1). If that node is not a leaf, choose one of its subnodes (in this case 2, though 9 would be equally suitable), and then recursively apply a depth first strategy to traverse the tree rooted at that subnode. Once this is complete, choose another subnode (if there is one) and traverse the tree rooted at that subnode. The traversal terminates when all subnodes have been explored.

Given that Prolog's theorem prover operates a depth first strategy, this strategy may be implemented in Prolog with the clause:

```
classify(Type, Sign) :-
        satisfies(Type, Sign),
        ((subtype(Type, SubType), classify(SubType, Sign));
```

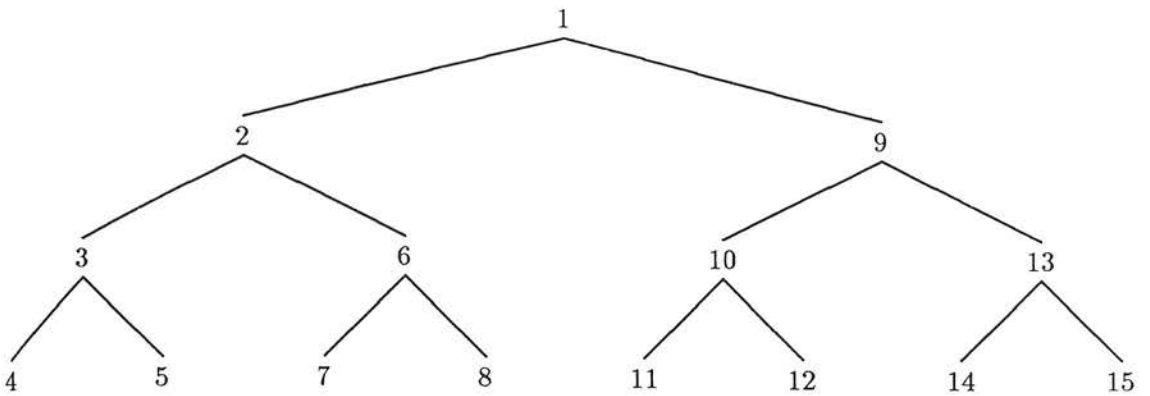


Figure 9.1: Depth First Strategy

```
\+ subtype(Type, _)).
```

That is, a feature structure `Sign` can be classified as being of type `Type` if it satisfies the individual requirements of `Type`, and either there is some subtype `SubType` of `Type` of which `Sign` is an instance, or `Type` has no subtypes. Given this code, whenever a recursive call to `classify/2` fails, Prolog's automatic backtracking will ensure a depth first traversal of the constituent hierarchy.

The breadth first strategy involves exploring the hierarchy in a "horizontal" direction. We begin with a singleton set consisting of a pair made up of the root node and a partial solution satisfying the conditions of the root node. Then for each pair `(solution:node)` in this set, for each subnode `subnode` of `node` we find all pairs of the form `(newsolution:subnode)` such that `newsolution` is an extension of solution satisfying the conditions associated with `subnode`, and group these into a set. We then repeat the process. If any node has no subnodes, then the corresponding solution is a complete solution to the traversal. Thus with a hierarchy such as that in Figure 9.2, the traversal proceeds numerically.

In Prolog a breadth first strategy must explicitly keep track of the set of possible (solution:node) pairs. Such a strategy might be implemented by the following clauses:

```
classify(Sign, Type) :-
        satisfies(Type, Sign),
        copy_term(Sign, CopySign),
```

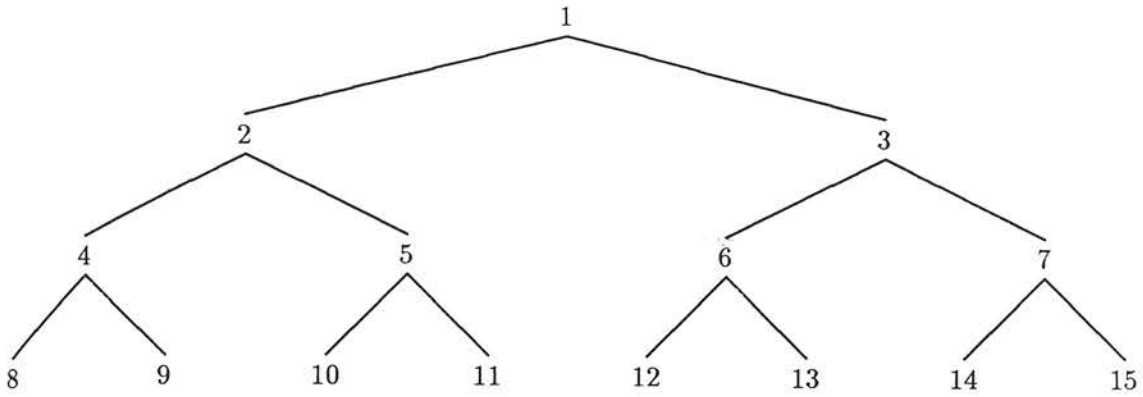


Figure 9.2: Breadth First Strategy

```
        breadth_first([(CopySign:Type)], Solutions), !,
        member(Sign, Solutions).


breadth_first([], []).
breadth_first([(S:T)|Tail], Solutions) :-
        subtype(T, _), !,
        findall((S1:T1), (subtype(T, T1),
                          unify(S, S1),
                          satisfies(T1, S1)), Extensions),
        append(Tail, Extensions, NewNodes),
        breadth_first(NewNodes, Solutions).
breadth_first([(S:T)|Tail], [S|Solutions]) :-
        breadth_first(Tail, Solutions).
```

The first argument of `breadth_first/2` is a list of (description:type) pairs, corresponding to partial possible solutions. The second argument is the list of complete solutions, given the (description:type) pairs of the first argument. `classify/2` calls `breadth_first/2` originally with its first argument instantiated to a list containing just the (description:type) pair associated with the root. `breadth_first/2` then generates all possible complete solutions. `classify/2` is designed to enumerate these solutions on backtracking, via the `member/2` predicate.

The depth first strategy is obviously very sensitive to the order in which subtrees are

traversed. This sensitivity is avoided in the case of the breadth first strategy. In each case, a solution might be found at the first leaf encountered, or it may not be found until the last leaf has been encountered, but with depth first, most of the hierarchy may separate these two leaves, whereas with breadth first, most of the hierarchy will have been examined before any leaves are encountered. That is, in the depth first case some searches will be relatively fast, whilst others will be relatively slow. In the breadth first case all searches will be somewhere between these two extremes. On average, if only one solution is required then depth first is more effective. In the examples above (that is given a symmetrical binary hierarchy), the minimum number of nodes examined in the case of depth first is only 4, whereas at least 8 must be examined in the case of breadth first. Assuming equal probability of each node being a solution, the average number of nodes examined is 11.5 for breadth first and 9.5 for depth first. The depth first strategy is, however, susceptible to a problem with potentially infinite branches of the hierarchy. If the hierarchy contains one infinite branch, then the traversal may not terminate, even if solutions exist. The breadth first strategy is guaranteed to terminate if a solution exists, even if some branches of the hierarchy are infinite. If, however, all solutions are required, then there is little to separate the strategies, as in both cases the entire hierarchy must be traversed. (Note though that the above code is designed to find all solutions, and so will not terminate with an infinite hierarchy.)

A final point is that the breadth first strategy is less space efficient than the depth first strategy as in the depth first case space may be reclaimed on backtracking. In the breadth first case several partial solutions (one for each branch at the current level) must simultaneously be entertained.

2.2.2 Top-Down and Bottom-Up

The depth first/breadth first dimension is common to most algorithms for hierarchy traversal. The use of a constituent hierarchy leads to a dimension that is peculiar to this application. Constituents are composed of subconstituents, and at some stage in the process of verifying that a string is a constituent it is necessary to verify that some substrings are subconstituents.

Formally, within CPSG the requirement that the subconstituents of a constituent are also

constituents is ensured by the use of relational dependencies which require all daughter feature structures of a feature structure describing a token of type constituent to also describe tokens of type constituent. The method of implementation of this unary relational dependency governs the top-down/bottom-up character of the parsing algorithm. A top-down strategy may be employed by simply implementing the dependency as a call to the original parsing routine, so that, for example, in attempting to classify 'Tigger is miaowing', the traversal of the hierarchy will be interrupted whilst the routine attempts to classify the subconstituents (i.e., 'Tigger' and 'is miaowing'). Thus, if parsing is effected by attempting to classify a feature structure description as describing a token of type constituent (as described above), then the top-level parse clause might be implemented as:

```
parse :-
        get_partial_description(Sign),
        classify(constituent, Sign).
```

(where `get_partial_description/1` returns the representation of the feature structure whose PHONOLOGY attribute has as its value the string, or list of words, to be parsed). The clause specifying the feature structure description of the type headed phrase (i.e., the body of the predicate `satisfies(headed_phrase, Sign)`) will include the following calls:

```
⋮
Sign/dtrs/head_dtr <=> HeadDtr,
constituent(HeadDtr),
⋮
```

The first of these ensures that `HeadDtr` is instantiated to the value of the DTRS|HEAD-DTR attribute of `Sign`, and the second implements the unary relational dependency that requires that `HeadDtr` describes a constituent. Given `classify/2`, the relational dependency may be implemented as:

```
constituent(Sign) :-
        classify(constituent, Sign).
```

where `classify/2` is implemented via a depth first or breadth first strategy as discussed above.

On the other hand, a bottom-up strategy may be implemented by first attempting to parse all substrings and keeping a record of well-formed descriptions (much as in traditional chart parsing), and then attempting to classify the string as a whole according to the hierarchy. Given this, the relational dependency reduces to a table look up, as when we need to check that all daughters of a constituent are also constituents, we will have already constructed a table of all well-formed subconstituents. Thus we have:

```
parse :-
        retractall(chart(_)),
        get_string_to_parse(String), !,
        parse_all_substrings(String).

parse_all_substrings(String) :-
        prefix(String, P),
        parse_prefix(P),
        fail.
parse_all_substrings(String).

parse_prefix([]).
parse_prefix([H|T]) :-
        parse_prefix(T),
        parse_string([H|T]), !.

parse_string(String) :-
        Sign/phon <=> list(String),
        classify(constituent, Sign),
        asserta(chart(Sign)),
        fail.
parse_string(String).
```

The relational dependency may then be implemented as:

```
constituent(Sign) :-
        chart(Sign).
```

Whilst the bottom-up strategy is more complex in that it requires more code, it is more efficient due to its chart-like character: old solutions are not forgotten on backtracking as they are in the top-down case. The extra code only serves to control the parsing of substrings, such that no attempt is made to classify a string before an attempt has been made to classify each of its substrings. Given a string $\langle \alpha\ \beta\ \gamma \rangle$, the above code parses the substrings in the order $\langle \alpha \rangle$, $\langle \beta \rangle$, $\langle \alpha\ \beta \rangle$, $\langle \gamma \rangle$, $\langle \beta\ \gamma \rangle$, $\langle \alpha\ \beta\ \gamma \rangle$. The bottom-up strategy also naturally finds all parses, not just the first parse.

3 Program Details and Efficiency Mechanisms

In this section we give details of some of the more interesting and unusual aspects of the program, paying special attention to issues of efficiency. A full program listing is given in Appendix B.

3.1 Feature Structures as Prolog Terms

For efficiency reasons, a system of closed sorts is employed in the encoding of feature structures. This allows feature structures to be represented by Prolog terms, with feature structure unification being effected by a modification of standard Prolog term unification. The representation consists of a pair whose first element is the sort and whose second is a list whose elements represent the values of the attributes defined for that sort. The position of a value in this list indicates which attribute that value corresponds to. Thus, for example, there is a sort `constituent`. Any feature structure of sort `constituent` is defined on the attributes `phon`, `syn`, `sem` and `dtrs`. A Prolog term of the form

```
(constituent, [W, X, Y, Z])
```

represents a feature structure of sort `constituent`, whose `phon` attribute has value `W`, whose `syn` attribute has value `X`, whose `sem` attribute has value `Y` and whose `dtrs` attribute has value `Z`. The ordering conventions, e.g., that the first element in this case corresponds to the `phon` attribute, are established by clauses of the form:

```
avm_frame(constituent, [phon, syn, sem, dtrs]).
```

There is one such clause for each sort.

A more apparent representation would be to employ clauses of the form `constituent(W, X, Y, Z)`. This representation is, however, difficult to handle when the sort of the feature structure (corresponding to the principal functor of the Prolog term) is not known.

In the development of the program, an alternate representation was employed whereby the Prolog terms representing feature structures consisted of a list of pairs of the form `Attribute:Value`. For each sort of feature structure, the order of the attributes was fixed, so that, for example, feature structures of sort constituent where always of the form:

```
[phon:_, syn:_, sem:_, dtrs:_]
```

Sorts where not explicitly indicated. Converting to the alternate representation described above resulted in the times taken to parse some benchmark constituents being reduced to less than half their former values.

3.2 Evaluating Path Equations

The use of disjoint attributes for each sort means that path equations implicitly encode the sorts of feature structures they involve. Thus, for example, a path equation involving a path of the form `X/syn/loc/head` is only well-formed if `X` is of sort `constituent`, because only feature structures of that sort are defined as having a `syn` attribute. Furthermore, the value of the `syn` attribute of `X` must be a feature structure of sort `syntactic`, because only such feature structures are defined on the `loc` attribute, and the value of the `loc` attribute of `X/syn` must be a feature structure of sort `local`, because only such feature structures are defined on the `head` attribute.

Path equations may be evaluated via the `value/2` predicate. A successful call to this predicate returns something like:

```
value((constituent, [_, (syntactic, [Z, _, _]), _, _])/syn/loc, Z).
```

Thus when a call of the form `value(X/syn/loc, Z)` is made, `X` will be instantiated to the Prolog term corresponding to the sorted feature structure:

$$\textit{constituent}\left[\begin{array}{ll} \text{PHON} & \ldots \\ \text{SYN} & \textit{syntactic}\left[\begin{array}{ll} \text{LOC} & \text{Z} \\ \text{BIND} & \ldots \\ \text{CONJ} & \ldots \end{array}\right] \\ \text{SEM} & \ldots \\ \text{DTRS} & \ldots \end{array}\right]$$

where '...' indicates uninstantiated values.

`value/2` is defined recursively by clauses which are automatically constructed by Prolog during a special compilation stage from the `avm_frame/2` clauses, which are essentially type declarations, so that the user need not be involved in this low-level coding — in defining sorts of feature structures all that the user must specify are the `avm_frame/2` clauses.

3.3 Unification

The operator `<=>` is defined for determining the unification of two feature structures. A goal of the form

```
Sign1 <=> Sign2
```

succeeds if and only if the feature structures represented by `Sign1` and `Sign2` unify, in which case `Sign1` and `Sign2` will be further instantiated so that they each represent the unification of the original feature structures.

The arguments of a call to `<=>` are evaluated by `value/2` before being passed to the unification predicates, so that `<=>` may be called with path equations as its arguments. Thus a goal of the form

```
Sign/syn/loc/spec <=> NP
```

will succeed if and only if `Sign` can be instantiated to a term representing a feature structure the value of whose SYN|LOC|SPEC attribute is unifiable with the feature structure represented by `NP`. Furthermore, if the predicate succeeds, the variables will be (further) instantiated to reflect the unification.

3.4 The Constituent Hierarchy

The constituent hierarchy is encoded via `subtype/2` clauses:

```
subtype(constituent, phrasal_constituent).
subtype(constituent, lexical_constituent).
subtype(phrasal_constituent, coordinate_phrase).
subtype(phrasal_constituent, headed_phrase).
⋮
```

With each type in the hierarchy, there is an associated `satisfies/2` clause, specifying the feature structure description associated with that type. Thus, for example, there is a clause for the type `lexical_constituent`:

```
satisfies(lexical_constituent, Sign) :-
        Sign/phon <=> list([_]),
        Sign/dtrs <=> null.
```

The `satisfies/2` clauses are generally more complex than this, as discussed in section 3.6.

3.5 Abbreviations

For ease of grammar writing, `abbreviates/2` clauses are included to allow abbreviations in the specification of feature structure descriptions of types. For example, the subcategorisation requirements of intransitive control verbs, which require a nominative noun phrase and a verb phrase complement, can be stated as:

```
⋮
NP abbreviates noun_phrase([nom]),
VP abbreviates verb_phrase,
Sign/syn/loc/spec <=> NP,
Sign/syn/loc/comps <=> set([VP]),
⋮
```

A call to `abbreviates/2` succeeds just in case the first argument is a term corresponding to a partial feature structure description of the second argument. The definition for the abbreviation of verb phrase is:

```
Sign abbreviates verb_phrase :-
        Sign/syn/conj <=> NIL,
        Sign/syn/loc/head/maj <=> verb,
        Sign/syn/loc/head/left_adjoins <=> set([]),
        Sign/syn/loc/head/right_adjoins <=> set([]),
        Sign/syn/loc/comps <=> null,
        Sign/syn/loc/spec <=> NP,
        NP abbreviates noun_phrase, !.
```

This succeeds if and only if Sign is instantiated to a term corresponding to some extension of the feature structure:

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{verb} \\ \text{LEFT-ADJOINS} & \{\,\} \\ \text{RIGHT-ADJOINS} & \{\,\} \end{bmatrix} \\ \text{SPEC} & \text{NP} \\ \text{COMPS} & \textit{null} \end{bmatrix} \\ \text{CONJ} & \text{NIL} \end{bmatrix} \end{bmatrix}$$

where NP partially describes a noun phrase.

3.6 Partial Execution in the Phrasal Hierarchy

The feature structure descriptions of phrasal types are given in terms of path equations, abbreviations and relational dependencies. For each type, all path equations are partially executed, giving the basic structure of the associated feature structure as a Prolog term. Any abbreviations, are also expanded through partial execution. Relational dependencies cannot be partially executed in the same way and must be evaluated only when the feature structures they constrain are sufficiently complete to allow a unique solution.

As an example, the clause specifying when a feature structure satisfies the idiosyncratic requirements of the type strict-transitive is:

```
satisfies(strict_transitive, Sign) :-
        NP abbreviates noun_phrase([acc, obj]),
```

```
        Sign/syn/loc/comps <=> set([NP]),
        Sign/sem/content/patient <=> NP/sem/content,
        Sign/sem/content/goal <=> null.
```

This is partially executed to yield a Prolog term T such that the compiled clause takes the form:

```
satisfies(strict_transitive, T).
```

Given this compiled clause, determining the success or failure of any call to `satisfies/2` of the form `satisfies(strict_transitive, X)` will involve only pattern matching in the head of the clause. If the clause had also contained a relational dependency, such as:

```
satisfies(headed_phrase, Sign) :-
        ⋮
        Sign/dtrs/head_dtr <=> HeadDtr,
        constituent(HeadDtr).
```

then the relational dependency would not be partially executed (though all other clauses would be), and the compiled clause would take the form:

```
satisfies(headed_phrase, T) :-
        constituent(X).
```

for some complex Prolog term T and subterm X.

This use of partial execution effectively ensures that irrespective of the ordering of clauses in the specification of a phrasal type, relational dependencies will be computed last. Thus, relational dependencies are only computed when the arguments are as fully instantiated as possible. The ordering of relational dependencies with respect to each other is not manipulated by the partial execution process, though the `wait` directive of some second generation Prologs provides scope for this if so desired.

3.7 List Values and List Unification

We represent lists in terms of Prolog terms consisting of the unary functor `list` whose sole argument is a Prolog list. Thus the list $\langle A, B, C \rangle$ is represented as the Prolog term `list([A, B, C])` (where `A` represents A, etc.). List unification is straightforward: two

lists $l = \langle l_1, l_2, \ldots, l_n \rangle$ and $m = \langle m_1, m_2, \ldots, m_n \rangle$ unify iff l_1 and m_1 unify, l_2 and m_2 unify, ...and l_n and m_n unify, and their unification is the list

$$l \sqcup m = \langle l_1 \sqcup m_1, l_2 \sqcup m_2, \ldots, l_n \sqcup m_n \rangle$$

Note that two lists must be of the same length to unify.

With Prolog's representation of lists as consisting of a head, representing the first element of the list, and a tail, representing the remainder of the list, list unification may be implemented by the following clauses:

```
unify_feature_structures(list([]), list([])).
unify_feature_structures(list([H1|T1]), list([H2|T2])) :-
        unify_feature_structures(H1, H2),
        unify_feature_structures(list(T1), list(T2)).
```

As CPSG only employs lists of atoms, list unification can be simplified to Prolog term unification. The actual clause in the implementation concerning list unification is:

```
unify_feature_structures(list(L), list(L)).
```

3.8 Set Values and Set Unification

Sets are represented in much the same way as lists, except that a different unary functor (`set`) is used to identify a set. Thus the set {A, B, C} is represented as the Prolog term `set([A, B, C])`. This representation orders the elements of the set, and so we must be everywhere aware that no predicate should be sensitive to this ordering. That is, we must be careful to ensure that, for example, `set([A, B, C])` and `set([C, A, B])` represent the same set. Computationally this makes things vastly more complex.

Recall from section 5.4 of Chapter 1 that given our interpretation of sets of feature structures (i.e., as describing sets of objects, with a one-one correspondence between the elements (objects) of the set in the world and the elements (descriptions) of the set in the syntax), the unification of two sets involves pairing the elements of each set and forming the set whose elements are the unification of those pairs. Set unification may be implemented via the following clauses:

```
unify_feature_structures(set([]), set([])).
unify_feature_structures(set(S1), set(S2)) :-
        delete(H1, S1, T1), !,
        delete(H2, S2, T2),
        unify_feature_structures(H1, H2),
        unify_feature_structures(set(T1), set(T2)).
```

Here, `delete/3` is used to delete some member of each of the sets to be unified. Then these members are unified before the remainder of the sets are unified. There is no guarantee that `delete/3` will delete members from each set which describe the same object (and so which should be unified when unifying the set). If either recursive call to `unify_feature_structures/2` fails, or if the predicate is retried, then backtracking will cause `delete/3` to choose a different pairing. The cut after the first call to `delete/3` stops backtracking past the second call to `delete/3`, so that pairings are postulated in a canonical representation and each pairing is only considered once.

3.9 $\oplus$ and $\otimes$

Composite feature structures are not implemented in the program, but some comments are in order about how they might be implemented, and about why they are not.

Recall the relationship between feature structures involving $\oplus$ and $\otimes$ from section 1.5 of Chapter 5:

- $\otimes$ feature structures exist in the language and have no direct correlation with objects in the world being modelled,
- $\oplus$ feature structures describe composite objects which do exist in the world being modelled,
- an $\otimes$ feature structure describes an object just in case one of the juncts describes the object, or it describes a composite object,
- an $\otimes$ feature structure describes a composite object just in case each object in the composite is described by one of the juncts, and

- an $\oplus$ feature structure describes an object just in case that object is a composite object consisting of objects which are described by each of the descriptions making up the $\oplus$ feature structure.

The crucial point here is that composite objects, whose direct correlates are $\oplus$ feature structures, exist in the described world whereas $\otimes$ objects do not. Constraints involving $\oplus$ are thus fundamentally different from constraints involving $\otimes$. We may adopt a parallel between Prolog syntax and the syntax of our feature structure language and Prolog terms and the semantic objects (generalised deterministic finite state machines) described by the language. Constraints involving $\oplus$ thus amount to identities, partially specifying a unique Prolog term. Constraints involving $\otimes$ are more complex and a unique semantic object cannot normally be constructed which satisfies such constraints. In Prolog terms this means that constraints involving $\oplus$ may be evaluated when encountered, but the evaluation of constraints involving $\otimes$ has to be delayed until the Prolog term being described by them is sufficiently instantiated.

We may thus freely introduce a binary operator '+' for forming composites: for any two pairwise incomparable Prolog terms `A` and `B` we may form the term `A + B`, corresponding to the composite. We include amongst our unification clauses a clause for such composite terms:

```
(A + B) <=> (C + D) :-
        A <=> C,
        B <=> D.
```

We may not do anything analogous in the case of $\otimes$, as this connective exists only in the syntax.

Looking at composites more closely, note that given the representation of $\oplus$ as a binary operator which Prolog forces us to adopt, we must be careful to ensure that its properties of idempotence, symmetry, and associativity inherited from its definition in terms of an operator on sets are made explicit. Thus we must include the following clauses for the unification operator:

```
X <=> (A + B) :-                                % symmetry
```

```
        X <=> (B + A).

X <=> ((A + B) + C) :-                          % associativity
        X <=> (A + (B + C)).
```

Idempotence may be ensured by the relational dependency involved in coordination which generates composite descriptions.

Turning now to $\otimes$, we need to distinguish carefully between Prolog constraints and Prolog terms. The `freeze/2` predicate of many second generation Prologs might be useful here, so that occurrences of $\otimes$ may be treated as constraints whose application should be delayed until sufficient information is known about which element of the $\otimes$ description should apply. One way to do this might be to embed all uses of $\otimes$ inside `freeze/2`, such that such descriptions are programmed as constraints to be executed when the relevant feature structure is instantiated. A constraint such as:

```
S <=> (S1 * S2)
```

which states that `S` must unify with either `S1` or `S2` or the composite of `S1` and `S2`, might thus be programmed as:

```
S <=> (S1 * S2) :-
        freeze(S, ((S <=> S1);
                   (S <=> S2);
                   (S <=> (S1 + S2))))
```

This is not really satisfactory, because `freeze/2` only waits until its first argument is not a variable, and this will not, in general, ensure that `S` is sufficiently instantiated to only unify with one of the possible disjuncts. The opposite extreme, where the constraint is only executed when `S` is fully instantiated would be more desirable, but still this is not the right condition. Perhaps what is really required is a breadth first proof strategy rather than a depth first proof strategy. In any case, it is the difficulty of implementing these essentially disjunctive constraints which prevents the implementation of $\oplus$ and $\otimes$.

4 Lexical Organisation

4.1 Compiling the Lexicon

To increase efficiency, the lexical portion of the constituent hierarchy is compiled, so that the classification of lexical constituents reduces to lexical look-up, rather than hierarchy traversal. This is achieved in a special compilation stage in which all legitimate descriptions of lexical tokens are found. Each of these descriptions is then retained in the Prolog database in terms of a `lexical_constituent/1` clause. The process is effected by a failure-driven loop which takes the form:

```
compile_lexicon :-
        satisfies(constituent, Sign),
        classify(lexical_constituent, Sign),
        asserta(lexical_entry(Sign)),
        fail.
compile_lexicon.
```

4.2 Lexical Rules and Priority Unification

To maintain a single lexical hierarchy with no cross-classification, only base forms are classified in the hierarchy with lexical rules being used to derive inflected forms. In the special compilation stage, after the lexical portion of the constituent hierarchy has been traversed, several lexical rules are applied, ensuring that

- for each singular noun there are two plural forms, one subcategorising for a determiner and the other not,
- for each base form verb there are finite, past participle and present participle forms,
- for every complement of each verbal form there is a corresponding form containing a gap instead of the complement,
- for each verbal form taking a sentential complement, there is a corresponding form with an NP gap taking a VP complement, and
- for each auxiliary there is an inverted form, as discussed in Chapter 3.

These lexical rules are implemented in terms of priority unification: in each case a partial description is composed consisting of any information in the resultant description which is not part of the original description and the resultant description is formed by priority unification of this partial description with the original feature structure (with the partial description taking priority). The lexical rule which generates plural nouns from the singular forms is:

```
apply_inflectional_rule(L) :-
        L/syn/loc/head/maj <=> noun,
        L/syn/loc/spec <=> (_,_),
        L/phon <=> Phon,
        plural_morphology(Phon, Phons),
        M/phon <=> Phons,
        M <<= L,
        assertz(lexical_entry(M)),
        N/phon <=> Phons,
        N/syn/loc/spec <=> null,
        N <<= L,
        assertz(lexical_entry(N)).
```

Given a particular lexical entry `L`, this rule first checks that it is nominal (`L/syn/loc/head/maj <=> noun`) and that it requires a specifier (`L/syn/loc/spec <=> (_,_)`). The plural phonology of the form is then checked (via table look-up — no morphological analysis is carried out) and a schematic lexical entry `M` is constructed having this plural phonology as the value of its PHONOLOGY attribute. All features of `L` which do not conflict with those of `M` are then copied to `M` via the priority unification relation (`<<=`). The resulting sign `M` is a feature structure description of the lexical entry for the plural form of the noun which requires a determiner. Lexical entries for bare plural are similarly constructed by the remaining lines of this clause.

Calls to `<=>` in lexical rules are partially executed in the same way as such calls in the phrasal hierarchy, slightly increasing the efficiency of the rules.

The order of application of lexical rules is highly significant. The two rules for slash termination must be applied after the inflectional rules to ensure that there are "slashed" forms of all verbal forms, rather than just the base forms. The subject-auxiliary inversion rule must not be applied before the slash termination rules to ensure that slashed forms of inverted auxiliaries are not created. The rules are thus applied in the order listed above.

5 The User Interface

The program consists of a SICStus Prolog saved state, `cpsg`, which may be run by entering the command `cpsg` at the prompt. The program will then loop, parsing strings and allowing simple interrogation of the results via a menu, until terminated. On suitable terminals, the program allows the display of feature structures corresponding to parsed constituents in a SunView[1] window.

5.1 Program Control

When executed, the program begins by displaying a title before prompting for a sentence to parse. The user must then type a string adhering to the requirements of section 5.2 below. The program will then attempt to parse the string as a constituent of the grammar and display statistics about the parsing (time taken to parse and, in the case of the bottom-up strategy, number of parses found), before displaying a menu and entering a loop from which various commands may be selected and executed. The commands include:

- parse another string
- list the well-formed substrings found in parsing the previous string
- save an ASCII format feature structure for some parse in a file
- display a feature structure for some parse on an ASCII terminal
- display a feature structure for some parse in a SunView window
- display a parse tree for some parse in a SunView window

[1]SunView is a trademark of Sun Microsystems, Inc.

- toggle the display of the DAUGHTERS attribute in feature structures
- enter Prolog debug mode
- exit the program, returning to the operating system

The basic structure of a session thus involves the user giving a string, the parser classifying that string and in so doing constructing feature structure descriptions for the string, and the user interrogating the system about the parses found. This process may loop indefinitely.

Only the two depth first strategies (top-down and bottom-up) are implemented. Each strategy forms a separate program which behaves in the above manner.

5.2 User Input

Input, in the form of sample strings to classify, is given at the '> ' prompt. These strings should contain no punctuation and be terminated by a carriage return. Proper names in the lexicon are capitalised, and thus must be capitalised when occurring in input strings. A listing of the lexicon is given in section 2 of Appendix A.

5.3 Terminal Output

The parser includes output routines for displaying feature structures in ASCII format. Thus the feature structure associated with a given parse may be displayed, allowing the user to examine the details of the parses found. An option allows these ASCII representations to be saved to a file for later examination and comparison. Because of the size of the feature structures employed by CPSG, there is an option to suppress the display of the DAUGHTERS attribute, which is responsible for most of the size of feature structures.

5.4 SunView Output

On terminals that support the use of SunView windows, output is provided in a prettier format. Feature structures can be drawn on a SunView canvas with a scrollable window. Thus, for feature structures which are too large to display on the screen, the user can

scroll around the canvas and examine the individual parts of the feature structure. Again, suppression of the DAUGHTERS attribute is possible.

The SunView environment also allows trees to be drawn. From a feature structure description the program extracts the daughters and draws the phrase structure tree corresponding to the parse represented by that feature structure. The nodes are annotated with the phrase they dominate. These trees are meant to primarily represent immediate dominance. The linear ordering of nodes does not follow the linear precedence of subconstituents within a constituent. The actual linear ordering employed results in heads being drawn to the left of their complements/specifiers/adjuncts/etc. The ordering of these other daughters, whilst fixed, is not significant.

5.5 A Sample Run

For completeness, a short sample run is included here, showing the results of parsing the constituent 'Tigger is miaowing'. User input is underlined and the lines are numbered for reference.

```
1     % cpsg
2
3
4                                   Prolog Parser for CPSG
5
6                                     by Richard Cooper
7                                      22nd June 1990
8
9
10
11
12
13    > Tigger is miaowing
14    Finished parsing in 0.450 seconds.
15    Total number of successful parses: 1
16
```

```
17
18              1.  Parse.
19              2.  List wffs of previous parse.
20              3.  Save an avm to a file.
21              4.  Display an avm on the terminal.
22              5.  Display an avm in a SunView window.
23              6.  Display a tree in a SunView window.
24              7.  Toggle display of daughters.
25              8.  Debug.
26              9.  Exit.
27
28
29      Enter Selection: 2
30
31
32      8: [Tigger,is,miaowing]
33      7: [is,miaowing]
34      6: [miaowing]
35      5: [miaowing]
36      4: [Tigger,is]
37      3: [is]
38      2: [is]
39      1: [is]
40      0: [Tigger]
41
42
43      Enter Selection: 9
44
45      { End of SICStus execution, user time 0.550 }
46
47      Bye ...
48      %
```

The program is invoked at line 1. Lines 2 to 12 constitute the title screen. The user is prompted at line 13 for an input string. This is given and 0.450 seconds later a response is given at lines 14 and 15. One successful parse is reported. The menu is then displayed (lines 18 to 26) and the user is prompted for an option (line 29). The user selects option 2, which lists all well formed substrings from the parse just previously carried out. These are numbered for ease of reference. Wff 8 corresponds to the successful parse of the entire string. Wff 7 is a successful parse of the verb phrase 'is miaowing'. Wffs 5 and 6 correspond to parses of the string 'miaowing' as an intransitive verb (wff 5) and as a verb phrase (wff 6). Wff 4 is corresponds to a parse of 'Tigger is miaowing' where 'miaowing' has been extracted. The feature structure corresponding to this wff has a non-empty slash value. There are three parses for 'is'. These correspond to the base form lexical entry, the lexical entry for the gapped form (where its complement verb phrase has been extracted), and the verb phrase formed from combining the gapped form with its required (i.e., no) complements. Wff 0 is the lexical entry for 'Tigger'. At line 43, the user is again prompted for an option. Option 9, `exit`, is chosen and the program terminates.

6 Summary

In this chapter we have discussed the issues involved in parsing CPSG, primarily from the perspective of a constituent hierarchy as developed in Chapter 2. Also discussed were other existing algorithms for parsing HPSG, which might be modified to parse CPSG, and the low level details of the actual implementation, with special reference to efficiency mechanisms and the implementation of the lexicon. In section 5 a brief overview of the program is given. This should allow the user to explore the grammar by examining the structures it produces when parsing sample constituents.

Chapter 10

Conclusion

In the previous chapters many of the details of Classification-based Phrase Structure Grammar, an extended revised version of Head-driven Phrase Structure Grammar, have been presented. Whilst the grammar revises many aspects of HPSG, and whilst the grammar extends the coverage of HPSG, it still raises more questions than it answers. This final chapter reviews CPSG and gives some pointers to some of the questions raised. Section 1 is really a retrospective summary of the main arguments, points and claims of the thesis. As a summary it is necessarily sketchy in places and few details are included.

The scope of the thesis is fairly wide, and many issues are raised which could not possible be adequately dealt with. Section 2 attempts to detail these many points, points that are beyond the scope of the thesis, and as such it is really just a list of unanswered questions, questions which remain for future research.

1 General Summary and Results

1.1 The Formal Framework of CPSG

In Chapter 1 we began by presenting much of the background material on which the formalism of CPSG relies. Whilst a fundamental part of this formalism is the language of feature structures, the language is secondary to the use of classification.

The chapter argues that the use by HPSG of "typed" feature structures places the notion of type at the wrong level. The usual token/type distinction implies that the notions of token and type are independent from the use of feature structures, and following

[Seligman 90a, 90b] we present the notion of an *abstract classification* to clarify the distinction between tokens and types. Tokens and types are thus taken to be prior to the use of feature structure descriptions. Still, the language of feature structures is undeniably of great utility in linguistic theory, and as such we incorporate feature structures into our classifications. Feature structures are thought of as describing both tokens and types. We may then consider domains of described tokens and domains of described types, and from these construct a classification with the *is of type* relation being mediated by the feature structure descriptions of the tokens and types: a token is an instance of a type if and only if its description extends the description of that type.

Seligman also considers law-like dependencies: relations that hold between tokens by virtue of them being instances of related types. Such a notion is also useful within linguistics: on our approach grammar rules and lexical rules can be seen as law-like dependencies. Within linguistics such dependencies are in general a result of the structured nature of tokens: an instance of a noun phrase consists of an instance of a determiner followed by an instance of a common noun. The noun phrase, which is itself a token, consists of two other tokens, a determiner and a noun, and this is vitally important for the law-like dependency. We thus extend Seligman's notion of a law-like dependency to allow for structured tokens.

A further notion discussed is that of an accidental correlation. When looking at a single classification, correlations between tokens and types may be accidental, an idiosyncrasy just of this particular classification, or law-like, a result of some correlation which cannot be violated and which all classifications must necessarily respect. Accidental correlations are purely extensional, whereas law-like dependencies have an intensional character to them. The use of feature structure descriptions of types allows a distinction to be drawn between law-like dependencies and accidental correlations. Essentially the descriptions of types allow the necessary to be distinguished from the accidental.

Having augmented abstract classifications with feature structures, Chapter 1 concludes with a discussion of what we take feature structures to be. We treat feature structures in terms of an extended Rounds-Kasper logic. As in HPSG, CPSG requires more descriptive power than that afforded by unadorned Rounds-Kasper logic. The extensions we make

are fairly similar to those of HPSG, including the use of further logical operators (negation, disjunction and implication) and the use of set and list valued attributes. Unlike HPSG we also employ a system of closed sorts on our domain of feature structures (this is in many ways independent of our notion of type), and allow relational dependencies between the values of attributes. Lastly we consider the generalisation of set and list valued attributes to poset valued attributes. We provide, for each of the extensions, the required modifications to the semantic domain and the necessary satisfiability clauses.

1.2 Constituent Hierarchies

A fundamental notion of CPSG is the extrapolation of the hierarchical structure present in the lexicon of HPSG to all constituents. The lexicon of HPSG is organised according to several hierarchies, with individual lexical items being cross-classified according to those hierarchies. Chapter 2 argues that hierarchical structure can also be motivated on phrasal constituents, and that an entire grammar can be viewed in terms of a constituent hierarchy. This view is actually implicit in the type hierarchies of HPSG, but in HPSG there is no attempt to exploit, or to even acknowledge, the structure. In contrast, the constituent hierarchy is central to CPSG. The hierarchical structure is taken to be independent of the domain of feature structures. It is structure which exists independently on the domain of constituents. This is consistent with the view of tokens and types developed in Chapter 1. Indeed, the hierarchical structure fits very well with the CPSG view of grammar as a classificatory system — the nodes in the hierarchy correspond to classes of tokens, or types. The hierarchical structure is thus structure on the domain of types. This is developed formally in Chapter 2, where *hierarchical classificatory systems*, classificatory systems in which the domain of types is hierarchically structured, are considered.

In terms of HPSG, the leaves of the constituent hierarchy correspond to the grammar rules. In any grammar the grammar rules can be seen as identifying types of phrases: each grammar rule, by licensing phrases, classifies those phrases as being of the type licensed by that grammar rule, and in a disambiguated grammar the grammar rules partition the phrases. Furthermore, the view in HPSG of grammar rules as corresponding to feature structures facilitates the CPSG view of rules as corresponding to types which

are in turn described by feature structures. The feature structures of HPSG correspond to the descriptions of the types of CPSG.

Whilst a constituent hierarchy can be formulated for HPSG from the type hierarchy on the domain of feature structures, there are a number of significant differences between the resultant constituent hierarchy and the constituent hierarchy of CPSG. Because CPSG takes types to be prior to feature structure descriptions, there is no tendency in CPSG to treat the hierarchical structure in terms of similarities in feature structure descriptions. The CPSG constituent hierarchy may be motivated entirely on the grounds of the linguistic generalisations which motivate the types in the first place. This aids the constituent hierarchy of CPSG in its attempts to be language independent: language independent generalisations can be made across types of phrases. This leads to the strong claim that the CPSG constituent hierarchy is a language universal.

Chapter 2 concentrates on the phrasal portion of the hierarchy, but several important claims are also made about the lexical portion. Within HPSG it is necessary to cross-classify lexical items according to several, not entirely independent, hierarchies. CPSG, in contrast, assumes only a single hierarchy according to which only base forms are classified. Inflected forms may be derived from these base forms by the application of lexical rules.

1.3 The Syntactic Treatment of Arguments and Adjuncts

The phrasal hierarchy of CPSG partitions phrasal constituents into headed phrases, phrases with a single head, and coordinate phrases. Headed phrases are in turn partitioned into those consisting of a head together with some of its semantic arguments, those consisting of a head together with a modifier, and those consisting of a head together with some kind of marker (such as a conjunction). Chapter 3 examines these first two subtypes of headed phrases in detail. Much of the treatment follows that of HPSG — the feature structure descriptions given for each type closely resemble the principal grammar rules of HPSG.

There are two principal differences in the treatment of arguments in CPSG and in HPSG. Firstly, following [Borsley 87], specifiers and complements, or external arguments and

internal arguments, are distinguished. Borsley provides several arguments for this distinction. We supplement these arguments with an argument based on the grammar rules of HPSG: Grammar Rule 1 and Grammar Rule 2 of HPSG differentiate between specifiers and complements, and by treating specifiers and complements via separate attributes the question of why a grammar rule shouldn't combine a head with, say, all but two of its arguments doesn't arise. This approach makes the syntax of CPSG very much like $\overline{\text{X}}$-theory, with grammar rules 1 and 2 reducing to two standard $\overline{\text{X}}$ schema. HPSG's Grammar Rule 3, which licenses inverted clauses, is one instance where it seems that no distinction is desired between specifiers and complements. Within CPSG inverted clauses are licensed via a lexical rule paralleling GPSG's metarule of subject-auxiliary inversion. It is claimed that this provides a more satisfactory treatment of inversion, as by restricting the lexical rule to auxiliary verbs, the question of the existence of inverted noun phrases is not raised. A further argument for the treatment of specifiers and complements in terms of different attributes (from Chapter 4) concerns the non-extraction of specifiers: complements can be extracted whereas (non-embedded) specifiers cannot.

The second principal difference between the HPSG account of arguments and that of CPSG concerns the use of an ordering on the arguments. In HPSG arguments are ordered according to obliqueness. This is represented in terms of a SUBCAT *list*. In CPSG arguments are unordered. This is represented in terms of a COMPLEMENTS *set*. Two principal arguments influence the unordered treatment of CPSG. Firstly, the list treatment, which has its origins in standard categorial accounts, requires that complements be *totally* ordered. It is not clear that this is always the case. In particular, some heads, such as verbs which take multiple prepositional phrase complements, seem to take complements which are equally oblique. In such cases it is not clear how the complements should be ordered. Secondly, within standard (i.e., functional) categorial treatments the total ordering required on arguments is in part due to the use of a functional semantic domain: the semantic domain, consisting of curried unary functions, requires that arguments be totally ordered. The semantic domain of HPSG (and CPSG) is based on unification, rather than function application, and as such no ordering on arguments is required in the semantic domain. Given this it seems odd to impose such an ordering on the syntactic domain.

Despite the unordered nature of the COMPLEMENTS set, obliqueness, or at least grammatical function, is clearly linguisticly significant. At the very least there must be some way to distinguish complements within a set. In the move from the list representation to the set representation, CPSG invokes a further attribute, GRAMMATICAL-FUNCTION, which heads mark on their complements. This allows complements to be distinguished by their grammatical function, and means that further principles of grammar may employ information concerning grammatical function. These principles may even require an ordering on the grammatical functions, but there is no requirement that the ordering be total, or that all arguments receive a different grammatical function.

Chapter 3 also discusses head/adjunct phrases. The treatment of adjuncts in CPSG differs from that in HPSG in that, rather than treating heads as being marked for their possible adjuncts, adjuncts are treated as being marked for the possible heads which they may adjoin to. A head may be modified by several adjuncts, so the head itself cannot be marked with a specific adjunct. This treatment has the advantage that it avoids a rather inelegant and obscure requirement of the HPSG treatment of adjunction whereby certain attributes are required to be unifiable without actually being unified.

1.4 Unbounded Dependencies

The analyses of unbounded dependencies in CPSG is relatively straightforward in many respects, but it also raises many interesting questions. In general form the analyses follows that of GPSG: it involves slash termination, slash percolation and slash introduction.

With regard to slash termination, the licensing of phrases containing gaps, CPSG is agnostic. Several possibilities are suggested, including the use of traces (possibly constrained by the presense of some feature), the use of modified phrase structure rules licensing phrases consisting of heads without all of their complements, and the use of lexical rules mapping lexical heads to lexical heads with reduced subcategorisation requirements and corresponding gaps, but each is found to be in some way unsatisfactory.

The treatment in CPSG of slash percolation, the passing of binding features between mother and daughter nodes, is more positive. Following the GPSG account, the Foot

Feature Principle is reconstructed by considering, for each binding attribute, the requirements which a head places on its daughters and the requirements which the daughters place on their head. In feature structure terms this involves splitting the value of each binding attribute into two sets, one whose elements are inherited from the constituents' daughters and one whose elements are imposed by the head on the daughters. The Foot Feature Principle is then reconstructed in terms of a relationship between the values of these sets, and, unlike GPSG, no notion of default is required.

Slash introduction in CPSG is accomplished by the analogue of the GPSG phrase structure rule: a leaf of the constituent hierarchy licenses phrases of type head/filler phrase. This is a subtype of head/argument phrase not discussed in Chapter 3, and one of the claims of Chapter 4 is that head/filler phrase and head/specifier phrase are subtypes of a more general type head/topic phrase. Similarities between these types of phrases are highlighted in Chapter 4, and these similarities motivate the placement of head/filler phrase on CPSG's phrasal hierarchy.

1.5 Coordination

There has been very little work in HPSG on the area of coordination. In Chapter 5 it is acknowledged that there are essentially two possible treatments of coordination. Either the other grammar rules may be modified so that they may accept multiple copies of a single argument or adjunct, provided that structure is suitably licensed by appropriate conjunctions, or a grammar rule may be added to actually license coordinate phrases, phrases such as 'neither Tigger nor Eeyore'. According to the first approach the grammar need not license coordinated constituents as constituents in themselves. CPSG adopts the second approach, assuming that strings such as 'neither Tigger nor Eeyore' actually are constituents in their own right. Such constituents are of type coordinate phrase, and form the second major branch of the phrasal hierarchy. Coordinate phrases much like those in GPSG are thus licensed, included iterated coordinate phrases and binary coordinate phrases.

The principal stumbling block to this kind of approach to coordination in unification-based frameworks in the past has been the problem of cross-categorial coordination: constituents of unlike categories may be coordinated provided the context in which

they are coordinated licenses each constituent individually. This in turn suggests some kind of algebra of syntactic categories, whereby if α and β are two categories there exists a third category which is in some way a composite of these two. Chapter 5 develops this idea by augmenting the logic of feature structures with two further connectives, $\oplus$, which allows composite feature structures to be formed, and $\otimes$, which in some ways is the dual of $\oplus$. Basically the idea is that for any categories (feature structures) $\alpha_1 \ldots \alpha_n$, there exists a composite category (feature structure) $\oplus\{\alpha_1 \ldots \alpha_n\}$, the category (head features) of the coordinate phrase whose conjuncts have categories (head features) $\alpha_1 \ldots \alpha_n$. A composite feature structure is then defined to unify with any other feature structure if and only if the other feature structure unifies with each of the elements of the composite. A precise semantics is given in terms of satisfiability of composite descriptions with respect to generalised finite state automata. The connective $\otimes$ is introduced for reasons of symmetry: $\oplus$ has something of the character of $\wedge$ and $\otimes$ has something of the character of $\vee$. This extension to the logic of feature structures means that coordination can be incorporated into the grammar without altering anything else in the grammar (apart from replacing $\vee$ with $\otimes$).

Chapter 5 completes the syntactic considerations of the thesis.

1.6 Situation Theory

The thesis takes a change of direction in Chapter 6, where semantic issues come to the fore. Chapter 6 is largely independent from CPSG, but is essential to understanding the rational behind the SEMANTIC attributes of the feature structure descriptions of constituents. Situation Theory is a new and developing field, and this chapter attempts to present a coherent version of the theory, along with a discussion of the application of the theory to the semantics of natural language. As a developing field, Situation theory is also open to further development, and this chapter attempts to do this. In particular, a strong line is taken on the distinction between the situation relative and the situation independent, or the absolute. In the theory we present, the situation relative notions of relation and infon are taken to have situation independent analogues of type and proposition. Arguments for the use of such situation independent notions are presented in Chapter 8, where it is argued that certain "facts", those that are

structurally determined (such as quantificational "facts"), should be treated in terms of propositions rather than infons. The structural determination of such facts means that they may be seen as being logical and situation independent, and hence not relative to a situation.

Chapter 6 also attempts to make formally precise many of the situation theoretic notions introduced by the use of mathematical models. Much of this modelling is beyond the scope of the thesis, but many of the more basic notions benefit from the precision engendered by the models.

1.7 Situation Semantics

The treatment of meaning, and in particular linguistic meaning, in situation semantics is presented in Chapter 6. Meaning is taken to be relational, with linguistic meaning relating utterance situations and described situations. Linguistic utterances may also exploit *resource situations*, further situations which provide a sort of context for the utterance.

The actual details of the possible values of the SEMANTICS attribute are mostly presented in Chapter 7. Within CPSG there is an issue concerning whether semantic information should be combined via unification or by predication. The unification-based framework strongly favours the first option, but the second option is certainly possible and consistent with the use of situation theory. Whilst in the end unification is primarily used, the issue is not really resolved — it seems difficult to distinguish between the options — and as such it is important to be aware of the two possibilities.

The value of the SEMANTICS attribute is a feature structure including attributes for the CONTENT, whose value is either a feature structure describing a parametric proposition (for verbal constituents), or a parameter (for saturated nominal constituents), or a type (for unsaturated constituents), or some other such thing, depending on the constituent in question, and the CONTEXT, whose value is a set of parametric propositions which must each hold when anchored by the speaker connections on any particular occasion of use.

The semantics of lexical types in CPSG are fairly straightforward. Proper nouns and

pronouns denote parameters — the value of their CONTENT attribute is a parameter — with various restrictions on that parameter being stated via their CONTEXT attribute. Unlike HPSG, in which common nouns denote quantifier terms, common nouns in CPSG denote types, the type of object with the corresponding property in the relevant resource situation. This makes the treatment of the semantics of noun modification very simple. Noun modifiers map types to types. Verbal types denote parametric propositions, possibly with restrictions on some argument roles being stated via the CONTEXT attribute.

The semantics of various phrasal types is also considered in Chapter 7. For the most part semantic features percolate as in HPSG. The semantic content of a head/argument phrase is generally that of the head of the phrase, with the context being the union of the contexts of the daughters. For head/adjunct phrases, the semantic content of the phrase is generally that of the adjunct. Some coordinate phrases are also treated, with the semantic content being given in terms of structurally determined types.

As an extended example of the treatment of semantics, the auxiliary system of English is considered. A Reichenbachian perspective is adopted and each auxiliary verb is analysed as contributing to the (parametric) proposition expressed by the verb phrase as a whole. The treatment involves the association of locations with propositions, and the auxiliaries function to express properties of this location or to shift the location with respect to an utterance location.

In Chapter 8 various treatments of quantification within situation semantics are considered. HPSG adopts an unusual situation theoretic approach to quantification, and the arguments concerning structural determination motivate what is in many ways a more conventional approach within CPSG: determiners are treated as binary types that hold of two types if and only if the corresponding extensional set theoretic relation holds of the set of objects of the first type and the set of objects of the second type. Quantifier scoping is an obvious problem in HPSG, and this issue is also addressed. 2–14–89, a descendent of HPSG, approaches the scoping problem via a further set-valued attribute which acts as a quantifier store. The value of this attribute is manipulated by semantic principles which implement a variety of Cooper storage. Whilst this is a possibility

within CPSG, unadorned Cooper storage has various problems when dealing with nested constructions, and CPSG instead stores all quantifier terms and provides a Principle of Semantic Interpretation to interpret feature structures whose QSTORE attribute has a value other than the empty set.

Lastly, we consider how distributive readings of coordinated noun phrases may be generated. Such noun phrases are considered along with quantification because in functional semantic frameworks it seems that each requires the noun phrase to act as a function mapping the semantic content of the verb phrase to a proposition. The use of a coordinate term store, paralleling the quantifier store, together with structurally determined types allows distributive readings to be generated without such functional devices.

1.8 Computational Issues

The final content chapter, Chapter 9, deals with some of the computational issues which previous chapters raise, as well as detailing many aspects of an implementation of CPSG developed in conjunction with this thesis.

The basic formal structure of CPSG in terms of a classificatory system leads to a natural implementation of parsing as classification: a string is licensed by the grammar if and only if it can be legitimately classified, where classifying a string involves finding a coherent description of the string which extends the description of some type. The use of a hierarchical classificatory system suggests further how this classification might proceed. Given a string and a constituent hierarchy, the string may be classified (if it really is licensed by the hierarchy) by constructing a partial description of the string (which initially will contain only the strings phonology) and assuming that it is an instance of the type corresponding to the root node of the hierarchy. Given this assumption, the description of the string must also unify with the description of the root type. It is then a matter of recursively choosing a subtype and unifying the partial description of the string with the description of that subtype, until a path from the root to a leaf node is found. If at any point unification fails, backtracking to the previous choice point can be invoked. If no satisfactory path can be found then the string cannot be a legitimate constituent. Chapter 9 considers several different strategies for implementing this algorithm, amounting to different strategies for traversing the hierarchy and

different approaches to the treatment of subconstituents. A fundamental requirement on all feature structure descriptions of constituents is that all their daughters also be constituents. Within CPSG this is expressed in terms of relational dependencies. A top-down strategy may be implemented by taking these relational dependencies to be recursive calls to the classification algorithm. A bottom-up strategy, on the other hand, may be implemented by first classifying all substrings of a string, recording the results and then treating the relational dependency as a table lookup.

The structure of the lexicon, in terms of a classification of base forms with inflected forms being derived by lexical rules, slightly complicates the algorithm, and to simplify matters and improve efficiency the lexicon is precompiled, so that all lexical rules are applied and the classification of lexical items becomes table lookup. Further efficiency mechanisms include the partial execution of most path equations. A whole language for describing feature structures is actually embedded in Prolog (via the use of various operators for paths and unification), and this language is mostly precompiled to maximise efficiency. Only relational dependencies, which must in general be evaluated only when all possible information is known, are not precompiled.

Although Prolog terms are used (via another precompilation stage, which transforms path equations into Prolog terms) to represent feature structures, Prolog's built-in unification procedures cannot be used. This is a consequence of the use of set-valued attributes. Such attributes seriously impair the efficiency of the Prolog implementation of CPSG.

2 Future Research

The broad scope of the thesis leaves many questions unanswered, and there are many areas which would benefit from further research. We concentrate here on the outstanding linguistic, logical and computational aspects. Whilst this section may seem critical in places, in the light of the general program of CPSG the criticisms are constructive.

2.1 Linguistic Aspects

The argument of Chapter 2 for the organisation of a grammar as a constituent hierarchy is appealing in its own right, but further motivation for hierarchical structuring in general, and the actual hierarchy presented in particular, is desirable. Especially desirable would be the consideration of some language apart from English, and the application of classification and hierarchical classification to the constituents of that language. In general, establishing a constituent hierarchy is a global task — it is not something that can be done by looking at a restricted set of data — and as such this is not an inconsiderable task. Nevertheless, the program would be strengthened enormously by such work, which is truly necessary if the constituent hierarchy is to be claimed to be a language universal.

There are many more specific areas of CPSG that require further attention. The treatment of specifiers and complements is superior to that of HPSG, but whilst the treatment of adjuncts attempts to improve on that of HPSG, it is still vaguely unsatisfactory on a number of counts. The most serious of these is the treatment of word order. The division of the adjoinable heads of an adjunct into left adjoinable and right adjoinable captures the basic facts but does so in a somewhat *ad hoc* fashion. Word order in head/specifier and head/complement phrases is determined much more simply by linear precedence rules, and perhaps what is required by the account of adjuncts is a set of linear precedence generalisations governing their distribution in relation to the heads they adjoin to. There is also, of course, the general question of whether adjuncts select for heads or *vice versa*, and if they do select for heads (as we have argued) then what kind of heads adjuncts adjoin to — are they necessarily of category $\overline{X}$?

As made clear in Chapter 4, CPSG is not committed to any particular treatment of slash termination. Whilst the use of a trace, lexical rules and modified phrasal rules are all possible, they are also all slightly problematic. Further work on slash termination is clearly needed, not just to decide amongst these alternatives, but to develop some possibly unrelated mechanism that doesn't suffer from the short-comings of each of these mechanisms. In this regard, the treatment of unbounded dependencies in LFG should not be ignored. It is not clear how LFG's treatment might be incorporated into CPSG, but LFG's notion of global saturation, deriving from its Principles of Coherence

and Completeness, avoids many of the problems of the approaches considered in CPSG. At this stage it is possibly worth reiterating though the success that CPSG has in reconstructing GPSG's Foot Feature Principle, and the insights which this reconstruction gives to the mechanism of the percolation of information relating to binding features. Any treatment of slash termination should presumably not be at the expense of this reconstruction.

Examples of unbounded dependencies in Chapter 4 where drawn from several areas when motivating the various dimensions of variation relevant to our treatment. A number of these examples where not considered in any detail. In particular, cleft sentences, purpose clauses and parasitic gap sentences where used to illustrate contexts in which unbounded dependencies may occur, but no analyses where presented of these phenomena. This just illustrates that what is provided here is a fragment, albeit a larger fragment than that provided in [Pollard & Sag 87]. As such there are many holes which further work should attempt to patch. Such further work will presumably involve augmentations to the constituent hierarchy. What we claim to have in our constituent hierarchy though is all the principal types of constituents. Completeness of the hierarchy could not realistically be hoped for.

A further area of importance relating to unbounded dependencies concerns the conditions under which elements can be extracted. This area is barely broached by the thesis, and there is an interesting question of whether such conditions can be formulated purely in terms of constraints on slash percolation or whether constraints on slash termination are sufficient, or whether some combination of each is required. Within the literature the subjacency constraint of transformational grammars may be taken as a constraint on slash percolation, and subjacency data seems to indicate that percolation, and not just termination, must be constrained. On the other hand, the GPSG approach might be classified as employing constraints on slash termination (via constraints on where the NULL feature may appear). This question cannot be resolved simply, but the distinction between the two types of constraints encouraged by CPSG does seem to suggest a useful starting place.

There are many other syntactic issues beyond the scope of the thesis. To fully develop

CPSG these must of course be considered. A selection of these is provided by current research within HPSG. One focus of this research is anaphora and conditions on anaphor binding. Here it seems that the HPSG notion of obliqueness will be vitally important, especially as it relates to O-command, or obliqueness-command. That the treatment of obliqueness is an important distinguishing characteristic between HPSG and CPSG suggests that some work may be necessary to recover any ground gained by HPSG via this notion. However, the distinction is not serious. CPSG does not claim that obliqueness is not a linguistically significant notion, it just differs in its representation of obliqueness.

Lastly with regard to syntax, the similarities between HPSG and CPSG mean that all research within HPSG is of great relevance to CPSG. The domain of relevant topics is of course much wider than just those of current interest to HPSG, and an important step in the theory would be to attempt to account for just a few of the facts which Government-Binding theory has been wrestling with over the past decade.

The treatment of semantics in CPSG differs in a number of important respects from the treatment of semantics in HPSG. In many ways CPSG's semantics is more refined, but there still remain several areas within the fragment given where the semantic treatment is lacking refinement. In particular, the generalisations which hold over constituents in the syntactic domain are not well correlated with semantic generalisations. Principal culprits in this area are quantification, with the semantic treatment of determiners as heads and the consequent scoping ambiguities that this leads to, and coordination, where syntactic generalisations seem to suggest that the semantics should be treated in terms of polymorphic operators acting on sets of juncts which may in one instance all be types but in another instance all be propositions. It is not clear just how these problems, if they are problems, should be approached.

A further area where the semantic treatment is lacking is that of nested quantification. As discussed in Chapter 8, if we adopt some quantifier scoping algorithm sensitive to nested constructions, such as that of [Lewin 90], then it is unclear how quantifier scoping can be accomplished on the fly: it seems that quantifier scoping is more naturally accomplished by a further interpretive stage. Furthermore, it appears that to implement such an interpretive stage for nested constructions requires a global modification to the

modelling of propositions, so that propositions are universally treated as coming with a (possibly empty) set of stored quantifiers.

The above syntactic and semantic issues all bear on the development of CPSG, and no doubt pursuing any one of them would reveal further questions of equal importance.

2.2 Logical Aspects

On the logical front, there are also several independent areas which suggest future research. Broadly speaking these divide into questions raised by feature structure logics and questions raised by situation theory, but there are also problems which lie between the two, problems which arise from the embedding of situation theory in a feature structure logic.

The underlying feature structure logic of CPSG is much simpler than that of HPSG because the feature structures involved in CPSG are not sorted in the same way as those of HPSG. Nevertheless, several extensions to the basic feature structure language are required. We have questioned the use of some logical operators, in particular implication — the question remains though of whether we need disjunction and negation. On the other hand, we have introduced the further operators $\oplus$ and $\otimes$. Future research should determine just which logical operators are required, and give a sound and complete calculus for their manipulation.

We have also employed list-valued and set-valued features. The interpretation of list-valued features is fairly straightforward, but there are three natural interpretations of set-valued features within the feature structure logic. Our use of sets requires that there be a one-one mapping between elements of the set and elements of the described object. A COMPS set containing two elements is interpreted as meaning that the constituent requires exactly two complements. Set values might also be taken to only partially describe some set. On this view, there may be elements in the described set which are not described by any elements in the set value. This kind of set value is of use to situation theory, where a situation might be partially characterised in terms of a subset of the infons which it supports. The third interpretation of set-valued features is that each element in the description describes some element in the described set, but there

is no requirement that each element in the syntax describe a different element in the semantics. On this account, which corresponds most closely with the use of sets by [Pollard & Sag 87] and [Pollard & Moshier 89], a set-valued feature may have as its value a set of any number of elements and still describe a singleton.

[Pollard & Moshier 89] give a semantics for their interpretation of set in terms of a power domain. The semantics given here for set values, in terms of generalised deterministic finite state automata, is far less elegant and future research into the semantics of feature structure logics in terms of power domains is justified. The possibility that each of the interpretations of set values might be required within a single system should also not be ruled out, and a coherent semantics where all can coexist, together with the operators of $\oplus$ and $\otimes$, is clearly desirable. Another consideration here is whether poset-valued features are required, and if so whether they should be treated as part of a continuum between list-valued and set-valued features. Further explorations in domain theory might well yield a suitable semantics.

A further issue relating particularly to set-valued attributes stems from our use of priority unification in the statement of lexical rules. For each interpretation of set-values above, there is a different way of interpreting set unification, but even within a single approach there are many ways of interpreting the priority unification of two sets. Essentially the possibilities arise from taking different amounts of information as prior. Should priority unification, for example, be defined for sets of different cardinality? If cardinality is something which takes priority, then perhaps it should. (Similar questions can be asked for lists.) We take priority unification to be similar to normal unification with respect to such questions: to be defined the sets must have equal cardinality and to be deterministic an isomorphism between the sets is required. Nevertheless, the operation of priority unification is one which could be better understood, and one which signposts another avenue for further research.

With regard to situation theory, the model constructed in Chapter 6 has two major deficiencies. Firstly, no attempt is made to capture appropriateness conditions on the argument roles of relations and types, and secondly, no account is given of the argument roles of complex relations and types formed by abstraction over parametric infons and

propositions. Further research should be aimed at remedying both of these deficiencies. Appropriateness conditions might be approached in a theory internal way by thinking of them as restrictions on argument roles. Restrictions on argument roles, rather than on parameters, are not currently part of the theory, but it may be that this is what is required. The problems with the argument roles of complex relations and types may similarly be traced to parameters: when we abstract over a parameter we somehow have to keep a record of which argument roles of which parametric objects where mapped to that parameter, and this record should be independent of the parameter (so that α equivalence is a consequence of the modelling). This all suggests something unsatisfactory about the nature of parameters within the theory itself, and perhaps suitable refinements to the theory will solve these problems with the model.

The logical issue of intensionality is a further issue which must be addressed within situation theory. This is one area of situation theory which has received little attention, and to a large extent the successes of possible world semantics in treating the seeking of unicorns have not been replicated. This is an area of future research which is largely independent from CPSG. It may be seen solely as a matter of providing the correct situation theoretic interpretation of infons involving intensional relations, and as such it is most relevant to the development of situation semantics. Nevertheless, the scoping ambiguities possible within intensional contexts must be derivable within CPSG.

Lastly, between the issues of feature structure logics and situation theory lies a very important question concerning the treatment of parameters and abstraction within a feature structure logic. Put simply, the question is one of whether we need some kind of λ operator. In the semantics given, we do. In several places (such as noun modification and certain varieties of coordination) abstraction is necessary. Given the difficulties with modelling abstraction within a feature structure logic, this is almost an embarrassment. Future research must be aimed at either developing a suitable description language that does allow the natural treatment of abstraction, or developing a treatment of semantics with no appeal to abstraction.

2.3 Computational Aspects

The computational questions raised by the thesis span several areas. Firstly there are those raised by portions of the fragment which are not implemented — questions arising from difficulties translating the fragment into Prolog. A second area concerns questions of efficiency — questions raised by various parts of the implementation which are inefficient. Lastly, there are questions raised by the general approach of translating a feature structure description language into Prolog. These questions relate in some ways to the first questions.

The implementation of much of CPSG is assisted by the almost direct translation of the feature structure language into Prolog clauses. For the most part, once a CPSG grammar is specified in terms of a constituent hierarchy and a feature structure description for each type in that hierarchy, the implementation is straightforward. The one area where direct translation is not possible concerns relational dependencies, reflecting the fact that relational dependencies allow us to consider arbitrary relations. Consequently some relational dependencies, especially those dealing with sets, are implemented in simplified forms. This problem might be approached by restricting attention to a subclass of relations, with the extremist approach taking this subclass to be the empty class. A further problem with the Prolog interpretation of relational dependencies is their tendency to be non-declarative. The depth first strategy of the Prolog interpreter is famous for its problems with left recursion, and it is essentially these that make the various relational dependencies very sensitive to their relative ordering. This problem is alleviated to some extent by the use of partial execution, but the implementation is still sensitive to the ordering of relational dependencies. This sensitivity leads, in some instances, to inefficiencies. Current developments in Constraint Logic Programming may provide an answer to some of these problems, a first solution to which might be to treat *all* relations as constraints which can only be evaluated when their arguments are sufficiently instantiated.

Only a restricted fragment of coordination is implemented. As explained in Chapter 9, the connectives $\oplus$ and $\otimes$ are not implemented due to the nature of the constraints which $\otimes$ expresses. Given that there are no formal problems with the connectives, their implementation shouldn't be problematic, and the difficulties which arise are symptomatic of

deeper problems within Constraint Logic Programming. We return to these below.

The implementation aims to be an efficient faithful representation of the grammar. However, there are several inefficient aspects. One of these derives from the use of set-valued features. In CPSG set-valued attributes are common, especially given our treatment of subcategorisation in terms of sets rather than list. However, as discussed in Chapter 9, with our interpretation of set values, set unification is non-deterministic unless a bijection is provided between the elements of each set. Indeed, on this view, list unification is deterministic precisely because such a bijection is implicit in the total order on the elements of the list. For set unification such a bijection is not normally provided. Even so, without a bijection set unification is not as inefficient as it might at first seem: set unification is only polynomially (in fact $O(n^2)$) more expensive than the unification of the individual elements of the set, as for an n-membered set there are only $n(n-1)/2$ possible bijections. This optimistic result is due entirely to our interpretation of set values. It does not hold for the two other interpretations suggested for set values.

The dependency that some relational dependencies have on the instantiation of their arguments gives rise to an area of the implementation which is slightly unfaithful to the grammar. Some relational dependencies (such as that the head of a headed phrase is of type constituent) cannot be stated at the appropriate node because at that stage their arguments are insufficiently instantiated. Consequently they must be stated instead at all immediate subnodes. This is slightly unsatisfactory, but one approach to this might be to replace the partial execution of `satisfies/2` clauses already carried out with a compilation stage, in which all relational dependencies are automatically propagated to leaf nodes, with abbreviation and unification clauses being partially executed as before. This would lead to a hierarchy in which all tests for non-leaf satisfaction involve a single unification, with the satisfaction of leaf types also involving all relational dependencies collected between the leaf and the root.

With regard to the third sort of question raised by the implementation, the more general questions concerning the implementation of feature structure logics within Prolog, it may be worth taking as a point of departure the parallels between the Prolog syntax and

the syntax of the feature structure logic and Prolog terms and the objects of the semantic domain. With these parallels, the syntax is truly seen as a constraint language, with objects being constructed which satisfy those constraints. This also suggests another line for future research: that semantics for the feature structure logic be given in terms of Prolog terms.

Many of the inefficiencies that arise in the implementation do so because of Prolog's search strategy, and the necessity for backtracking. In particular, at least in some cases, it seems that a breadth first rather than depth first strategy might be appropriate. Such a strategy would of course have the corresponding disadvantage of producing many possible objects that would have to be considered at each stage in processing, but this may be more efficient in the end than backtracking, and there is always the possibility of dealing with the solutions in parallel. Basically, the future research relating to computational concerns is in the realm of Constraint Logic Programming, and not CPSG.

Bibliography

[Aczel 88] P. Aczel (1988): *Non-Well-Founded Sets.* CSLI Lecture Note Series, 14. Stanford: CSLI.

[Aczel 90] P. Aczel (1990): Replacement Systems and the Axiomatisation of Situation Theory. In [Robin Cooper *et al.* 90], 3–31.

[Ades & Steedman 82] A. Ades and M. Steedman (1982): On the Order of Words. *Linguistics and Philosophy*, 4, 517–558.

[Akmajian & Heny 75] A. Akmajian and F. Heny (1975): *An Introduction to the Principles of Transformational Syntax.* Cambridge: MIT Press.

[Andrews 83] A. D. Andrews (1983): A Note on the Constituent Structure of Modifiers. *Linguistic Inquiry*, 14, 695–697.

[Bach 83] E. Bach (1983): Generalized Categorial Grammars and the English Auxiliary. In [Heny & Richards 83], 101–120.

[Baltin & Kroch 87] M. Baltin and A. Kroch (eds.) (1987): *Alternative Conceptions of Phrase Structure.* Chicago: University of Chicago Press.

[Barwise 84] J. Barwise (1984): Logic and Information. In [Barwise 89], 37–57.

[Barwise 87a] J. Barwise (1987): Noun Phrases, Generalized Quantifiers and Anaphora. In [Gärdenfors 87], 1–29.

[Barwise 87b] J. Barwise (1987): Situations and Small Worlds. In [Barwise 89], 79–92.

[Barwise 88a] J. Barwise (1988): Situations, Facts, and True Propositons. In [Barwise 89], 221–254.

[Barwise 88b] J. Barwise (1988): Notes on Branch Points in Situation Theory. In [Barwise 89], 255–276.

[Barwise 89] J. Barwise (1989): *The Situation in Logic.* CSLI Lecture Note Series, **17**. Stanford: CSLI.

[Barwise & Cooper 81] J. Barwise & R. Cooper (1981): Generalized Quantifiers and Natural Language. *Linguistics and Philosophy*, 4, 159–219.

[Barwise & Perry 83] J. Barwise & J. Perry (1983): *Situations and Attitudes.* Cambridge: MIT Press.

[Bauerle *et al.* 83] R. Bauerle, C. Schwarze & A. von Stechow (eds.) (1983): *Meaning, Use and Interpretation of Language.* Berlin: de Gruyter.

[van Benthem 86] J. van Benthem (1986): *Essays in Logical Semantics.* Dordrecht: Reidel.

[van Benthem 90] J. van Benthem (ed.) (1990): *Partial and Dynamic Semantics I.* DYANA Deliverable R2.1A.

[Borsley 87] R. Borsley (1987): *Subjects and Complements in HPSG.* CSLI Report #CSLI–87–107.

[Borsley ∞] R. Borsley (Forthcoming): *Subjects, Complements, and Specifiers in HPSG.* To appear in [Pollard & Sag ∞b].

[Braisby & Cooper 89] N. Braisby and R. Cooper (eds.) (1989): *Situation Theoretic Studies in Psychology, Language and Logic.* Edinburgh Working Papers in Cognitive Science, **3**, Edinburgh: Centre for Cognitive Science.

[Bresnan 82a] J. Bresnan (ed.) (1982): *The Mental Representation of Grammatical Relations.* Cambridge: MIT Press.

[Bresnan 82b] J. Bresnan (1982): *The Passive in Lexical Theory.* In [Bresnan 82a], 3–86.

[Bromberger 88] S. Bromberger (1988): *Types and Tokens in Linguistics.* CSLI Report #CSLI–88–125.

[Burris & Sankappanavar 81] S. Burris and H. P. Sankappanavar (1981): *A Course in Universal Algebra.* New York: Springer-Verlag.

[Burton-Roberts 86] N. Burton-Roberts (1986): *Analysing Sentences: An Introduction to English Syntax.* London: Longman.

[Chomsky 81] *Lectures on Government and Binding.* Dordrecht: Foris.

[Chomsky 82] *Some Concepts and Consequences of the Theory of Government and Binding.* Cambridge: MIT Press.

[Colban & Fenstad 87] E. Colban & J. E. Fenstad (1987): *Situations and Prepositional Phrases.* In Proceedings of EACL, 258–261.

[Robin Cooper 75] R. H. Cooper (1975): *Montague's Semantic Theory and Transformational Syntax.* Ph.D. Thesis, University of Amherst.

[Robin Cooper 84] R. H. Cooper (1984): *Aspectual Classes in Situation Semantics.* CSLI Report #CSLI–84–14C.

[Robin Cooper 87] R. H. Cooper (1987): Preliminaries to the Treatment of Generalized Quantifiers in Situation Semantics. In [Gärdenfors 87], 73–91.

[Robin Cooper 88] R. H. Cooper (1988): *Facts in Situation Theory: Representation, Psychology or Reality.* In [Kempson 88], 49–61.

[Robin Cooper 89] R. H. Cooper (1989): *Structural Parameters.* Privately Circulated Note.

[Robin Cooper 90] R. H. Cooper (1990): *Lectures on Situation Theoretic Grammar.* Prepared for the APPIA Advanced School on Natural Language Processing, Guarda, Portugal, October, 1990.

[Robin Cooper ∞] R. H. Cooper (In preparation): *Introduction to Situation Semantics.*

[Robin Cooper *et al.* 90] R. H. Cooper, K. Mukai & J. Perry (1990): *Situation Theory and its Applications, Volume 1.* CSLI Lecture Note Series, **22**. Stanford: CSLI.

[Richard Cooper 90a] R. P. Cooper (1990): *Persistence and Structural Determination.* Paper presented at the 2nd Conference on Situation Theory and Its Applications. To appear in [Gawron *et al.* ∞].

[Richard Cooper 90b] R. P. Cooper (1990): *Coordination in Unification-Based Grammars.* Paper to be presented at the 5th Conference of the European Chapter of the Association for Computational Linguistics.

[Creary & Pollard 85] L. Creary & C. Pollard (1985): *A Computational Semantics for Natural Language*, In Proceedings of the 23rd ACL, 172–179.

[Davidson & Harman 72] D. Davidson and G. Harman (eds.) (1972): *Semantics of Natural Language.* Dordrecht: Reidel.

[Devlin ∞] K. Devlin (Forthcoming): *Logic and Information.*

[Dowty 85] D. Dowty (1985): Type Raising, Functional Composition, and Non-Constituent Coordination. Paper presented at the Tucson Conference on Categorial Grammar, June, 1985. Also in [Oehrle *et al.* 88], 153–197.

[Engdahl 83] E. Engdahl (1983): Parasitic Gaps. *Linguistics and Philosophy*, **6**, 5–34.

[Fenstad *et al.* 85] J. E. Fenstad, P-K. Halvorsen, T. Langholm and J. van Bentham (1985): *Equations, Schemata and Situations: A Framework for Lingusitic Semantics.* CSLI Report #CSLI–85–29.

[Flickinger 87] D. Flickinger (1987): *Lexical Rules in the Hierarchical Lexicon.* Ph.D. Dissertation, Stanford University.

[Flickinger *et al.* 85] D. Flickinger, C. Pollard and T. Wasow (1985): *Structure-Sharing in Lexical Representation.* In Proceedings of the 23rd ACL, 262–267.

[Gärdenfors 87] P. Gärdenfors (ed.) (1987): *Generalized Quantifiers: Linguistic and Logical Approaches.* Studies in Linguistics and Philosophy, **31**. Dordrecht: Reidel.

[Gawron & Peters 90a] M. Gawron & S. Peters (1990): *Anaphora and Quantification in Situation Semantics.* CSLI Lecture Note Series, **19**. Stanford: CSLI.

[Gawron & Peters 90b] M. Gawron & S. Peters (1990): Some Puzzles About Pronouns. In [Robin Cooper *et al.* 90], 395–431.

[Gawron *et al.* ∞] M. Gawron, G. Plotkin & S. Tutiya (forthcoming): *Situation Theory and its Applications, Volume 2.* CSLI Lecture Note Series. Stanford: CSLI.

[Gazdar 81] G. Gazdar (1981): Unbounded Dependencies and Coordinate Structure. *Linguistic Inquiry*, **12**, 155–184.

[Gazdar *et al.* 85] G. Gazdar, E. Klein, G. Pullum and I. Sag (1985): *Generalised Phrase Structure Grammar.* Oxford: Basil Blackwell.

[Geach 72] P. Geach (1972): A Program for Syntax. In [Davidson & Harman 72], 483–498.

[Gibson 89] E. Gibson (1989): *Parsing with Principles: Predicting a Phrasal Node Before Its Head Appears.* In Proceedings of the International Workshop on Parsing Technologies, Pittsburgh, 63–74.

[Groenendijk & Stokhof 90] J. Groenendijk & M. Stokhof (1990): *Dynamic Predicate Logic: Towards a compositional, non-representational semantics of discourse.* In [van Benthem 90], 53–108.

[Halvorsen 87] P.-K. Halvorsen (1987): *Situation Semantics and Semantic Interpretation in Constraint-Based Grammars.* CSLI Report #CSLI–87–101.

[Hanson 89] P. Hanson (ed.) (1989): *Information, Language, and Cognition.*

[Heny & Richards 83] F. Heny and B. Richards (eds.) (1983): *Linguistic Categories. Volume 2: Auxiliaries and Related Puzzles.* Dordrecht: Reidel.

[Hintikka *et al.* 73] J. Hintikka, J. Moravcsik & P. Suppes (1973): *Approaches to Natural Language.* Dordrecht: Reidel.

[Hobbs & Shieber 87] J. Hobbs & S. Shieber (1987): An Algorithm for Generating Quantifier Scopings. *Computational Linguistics*, **13**, 47–63.

[Huck & Ojeda 87] G. Huck & A. Ojeda (1987): *Discontinuous Constituency.* Syntax and Semantics, **20**. Academic Press.

[Johnson 88] M. Johnson (1988): *Attribute-Value Logic and the Theory of Grammar.* CSLI Lecture Note Series, **16**. Stanford: CSLI.

[Kaplan & Zaenen 87] R. Kaplan & A. Zaenen (1987): *Long Distance Dependencies, Constituent Structure, and Functional Uncertainty.* In [Baltin & Kroch 87], 17–42.

[Karttunen 84] L. Karttunen (1984): *Features and Values.* In Proceedings of CoLing '84, 28–33.

[Karttunen 86] L. Karttunen (1986): *DPATR: A Development Environment for Unification-Based Grammars.* CSLI Report #CSLI-86-61.

[Kasper & Rounds 86] R. Kasper & W. Rounds (1986): A Logical Semantics for Feature Structures. In Proceedings of the 24th ACL, 257–265.

[Kasper & Rounds 90] R. Kasper & W. Rounds (1990): The Logic of Unification in Grammar. *Linguistics and Philosophy*, **13**, 35–58.

[Kay 79] M. Kay (1979): *Functional Grammar.* In Proceedings of the 5th Annual Meeting of the Berkeley Linguistic Society, 142–158.

[Keller 87] W. Keller (1987): *Nested Cooper Storage: The Proper Treatment of Quantification in Ordinary Noun Phrases.* Cognitive Science Research Paper, University of Sussex.

[Kempson 88] R. Kempson (ed.) (1988): *Mental Representations: The Interface Between Language and Reality.* Cambridge: Cambridge University Press.

[Lambek 58] J. Lambek (1958): The Mathematics of Sentence Structure. *American Mathematical Monthly*, **65**, 154–170.

[Lewin 90] I. Lewin (1990): *A Quantifier Scoping Algorithm without a Free Variable Constraint.* In Proceedings of CoLing '90, Volume 3, 190–194.

[Link 83] G. Link (1983): The Logical Analysis of Plurals and Mass Terms: A Lattice-Theoretical Approach. In [Bauerle *et al.* 83], 302–323.

[McIntyre 89] A. McIntyre (1989): *Information-Based Parsing.* M.Sc. Thesis. Department of Artificial Intelligence, University of Edinburgh.

[Montague 70] R. Montague (1970): Universal Grammar. Theoria, **36**, 373–398. Reprinted in [Thomason 74], 222–246.

[Montague 73] R. Montague (1973): The Proper Treatment of Quantification in Ordinary English. In [Hintikka *et al.* 73], 221–242. Reprinted in [Thomason 74], 247–270.

[Morrill 88] G. Morrill (1988): *Extraction and Coordination in Phrase Structure Grammar and Categorial Grammar.* Ph.D. Thesis. Centre for Cognitive Science, University of Edinburgh.

[Morrill 89] G. Morrill (1989): *Grammar as Logic.* Research Paper EUCCS/RP–34, Center for Cognitive Science, Edinburgh.

[Moshier & Rounds 87] M. D. Moshier & W. Rounds (1987): A Logic for Partially Specified Data Structures. In Proceedings of the 14th ACM Symposium on Principles of Programming Languages, 156–167.

[Oehrle *et al.* 88] R. Oehrle, E. Bach and D. Wheeler (eds.) (1988): *Categorial Grammars and Natural Language Structures.* Studies in Linguistics and Philosophy, **32**. Dordrecht: Reidel.

[Pereira & Shieber 87] F. Pereira and S. Shieber (1987): *Prolog and Natural-Language Analysis.* CSLI Lecture Note Series, **10**. Stanford: CSLI.

[Perry 87] J. Perry (1987): *From Worlds to Situations.* CSLI Report #CSLI–87–73.

[Pickering & Barry 89] M. Pickering and G. Barry (1989): *Processing Extractions without Gaps.* Research Paper EUCCS/RP–36, Center for Cognitive Science, Edinburgh.

[Pinkal & Gregor 89] M. Pinkal & B. Gregor (eds.) (1989): *Unification in Linguistic Analysis.*

[Pollard 88] C. Pollard (1988): The Nature and Structure of a Computational Linguistic Theory. In Proceedings of the First Republic of China Workshop on Computational Lingusitics.

[Pollard 89a] C. Pollard (1989): Sorts in Unification-Based Grammars and What They Mean. In [Pinkal & Gregor 89].

[Pollard 89b] C. Pollard (1989): The Syntax-Semantics Interface in a Unification-Based Phrase Structure Grammar. In *Views of the Syntax/Semantics Interface.* Proceedings of the Workshop "GPSG and Semantics", Berlin. 167–184.

[Pollard & Moshier 89] C. Pollard and D. Moshier (1989): Unifying Partial Descriptions of Sets. Preprint. To appear in [Hanson 89].

[Pollard & Sag 87] C. Pollard and I. Sag (1987): *Information-Based Syntax and Semantics. Volume 1: Fundamentals.* CSLI Lecture Note Series, **13**. Stanford: CSLI.

[Pollard & Sag 88] C. Pollard and I. Sag (1988): *An Information-Based Theory of Agreement.* CSLI Report #CSLI–88–132.

[Pollard & Sag ∞a] C. Pollard and I. Sag (forthcoming): *Information-Based Syntax and Semantics. Volume 2: Topics in Control and Binding.* CSLI Lecture Note Series. Stanford: CSLI.

[Pollard & Sag ∞b] C. Pollard and I. Sag (forthcoming): *Readings in Information-Based Syntax and Semantics.* CSLI Lecture Note Series. Stanford: CSLI.

[Popowich 88] F. Popowich (1988): *Reflexives and Tree Unification Grammar.* Ph.D. Thesis. Centre for Cognitive Science, University of Edinburgh.

[Proudian & Goddeau 87] D. Proudian and D. Goddeau (1987): *Constituent Coordination in HPSG.* CSLI Report #CSLI–87–97

[Proudian & Pollard 85] D. Proudian and C. Pollard (1985): *Parsing Head-Driven Phrase Structure Grammar.* In Proceedings of the 23rd ACL, 167–171.

[Radford 88] A. Radford (1988): *Transformational Grammar: A First Course.* Cambridge: Cambridge University Press.

[Reichenbach 47] H. Reichenbach (1947): *Elements of Symbolic Logic.* London: Macmillan.

[Ross 67] J. R. Ross (1967): *Constraints on Variables in Syntax.* Ph.D. Dissertation, MIT. Indiana University Linguistics Club.

[Rounds 88] W. Rounds (1988): *Set Values for Unification-Based Grammar Formalisms and Logic Programming.* CSLI Report #CSLI–88–129.

[Rounds & Kasper 86] W. Rounds & R. Kasper (1986): A Complete Logical Calculus for Record Structures Representing Linguistic Information. In Proceedings of the 1st IEEE Symposium on Logic in Computer Science, 38–43.

[Sag 82] I. A. Sag (1982): Coordination, Extraction, and Generalized Phrase Structure Grammar. *Linguistic Inquiry*, **13**, 695–697.

[Sag 86] I. A. Sag (1986): *Grammatical Hierarchy and Linear Precedence.* CSLI Report #CSLI–86–60. Also in [Huck & Ojeda 87], 303–340.

[Sag *et al.* 84] I. Sag, G. Gazdar, T. Wasow and S. Weisler (1984): *Coordination and How to Distinguish Categories.* CSLI Report #CSLI–84–3. Also in Natural Language and Linguistic Theory, **3**, 1985, 117-171.

[Satto & Stock 89] G. Satto and O. Stock (1989): *Head-Driven Bidirectional Parsing: A Tabular Method.* In Proceedings of the International Workshop on Parsing Technologies, Pittsburgh, 43–51.

[Seligman 90a] J. Seligman (1990): *Perspectives in Situation Theory.* In [Robin Cooper *et al.* 90], 147–191.

[Seligman 90b] J. Seligman (1990): *Perspectives: A Relativistic Approach to the Theory of Information.* Ph.D. Thesis, Centre for Cognitive Science, University of Edinburgh.

[Shieber 86] S. Shieber (1986): *An Introduction to Unification-Based Approaches to Grammar.* CSLI Lecture Note Series, 4. Stanford: CSLI.

[Shieber 89] S. Shieber (1989): *Parsing and Type Inference for Natural and Computer Languages.* Ph.D. Thesis. Stanford University.

[Steedman 85] M. Steedman (1985): Dependency and Coördination in the Grammar of Dutch and English. *Language*, **61**, 523–568.

[Steedman 87] M. Steedman (1987): *Constituency and Coordination in a Combinatory Grammar.* In [Baltin & Kroch 87].

[Thomason 74] R. Thomason (1974): *Formal Philosophy: Selected Papers of Richard Montague*. New Haven: Yale University Press.

[Uszkoreit 86] H. Uszkoreit (1986): *Categorial Unification Grammars*. CSLI Report #CSLI–86–46.

[Zeevat *et al.* 87] H. Zeevat, E. Klein and J. Calder (1987): *Unification Categorial Grammar*. Research Paper EUCCS/RP–21, Center for Cognitive Science, Edinburgh.

Appendix A

A Fragment of CPSG

1 The Constituent Hierarchy

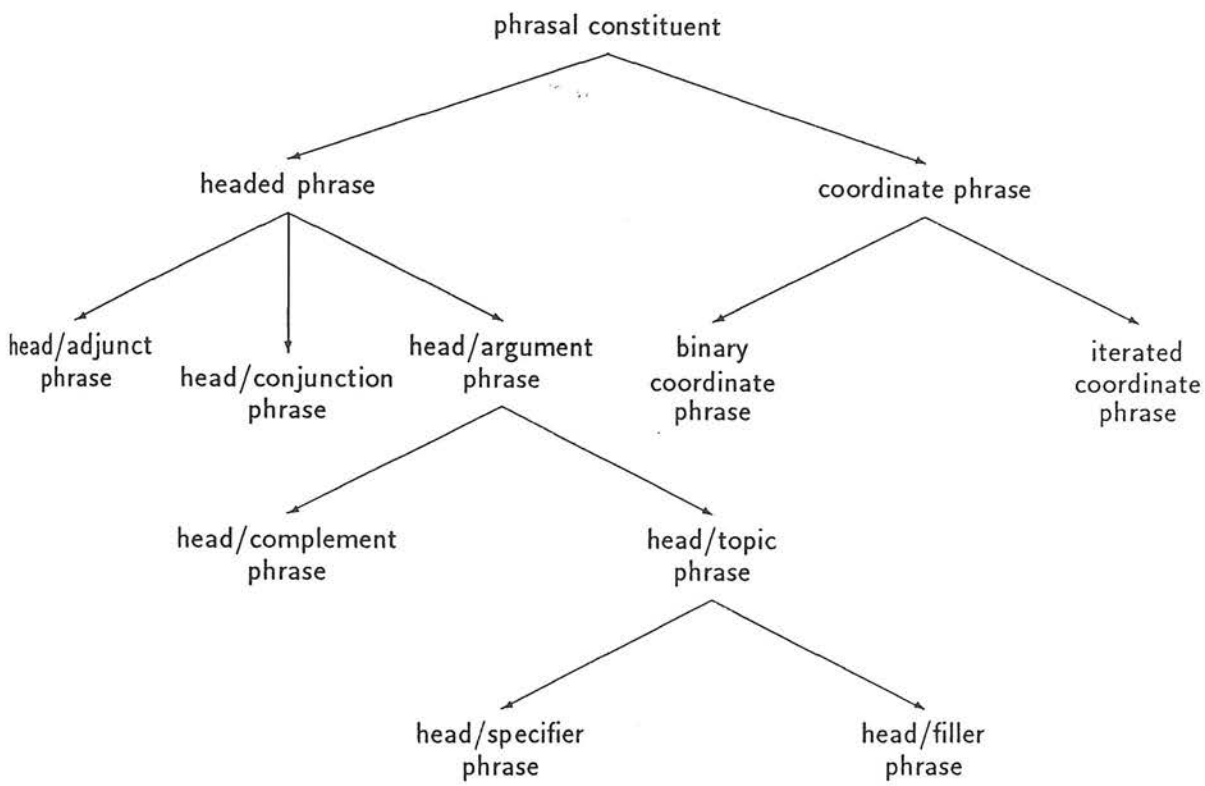


Figure A.1: The Phrasal Hierarchy of CPSG

Not shown in Figure A.1 is the subdivision of head/adjunct phrase into head/pre-adjunct and head/post-adjunct.

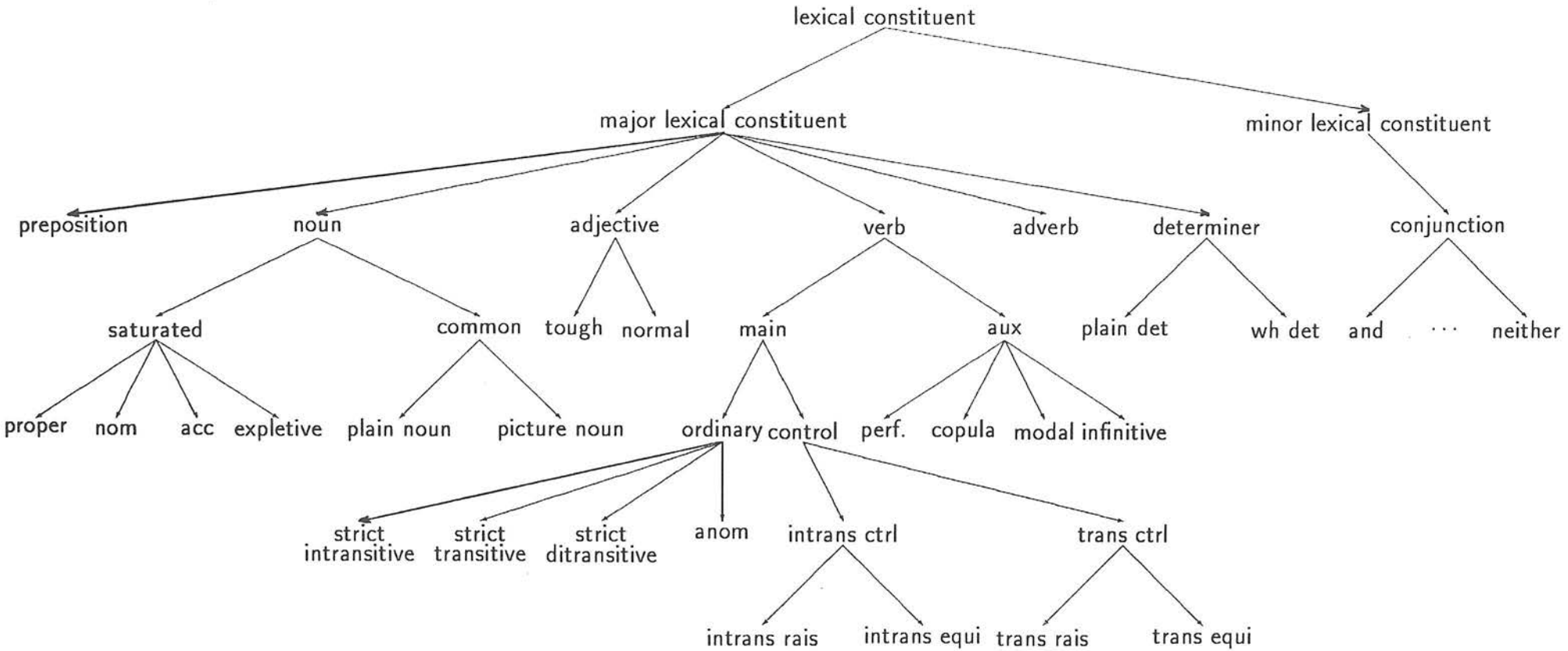


Figure A.2: The Lexical Hierarchy of CPSG

1.1 Descriptions of Phrasal Types

Γ(phrasal constituent) =

$$\begin{bmatrix} \text{PHON} & \langle \ldots \rangle \\ \text{DTRS} & [\ ] \end{bmatrix}$$

Γ(headed phrase) =

$$\left[\begin{array}{ll} \text{SYN} & \left[\text{LOC}\ \left[\text{HEAD}\ \boxed{1}\right]\right] \\ \text{DTRS} & \left[\begin{array}{ll} \text{HEAD-DTR} & \boxed{2}\left[\begin{array}{ll} \text{PHON} & \langle \ldots \rangle \\ \text{SYN} & \left[\begin{array}{ll} \text{LOC} & \left[\text{HEAD}\ \boxed{1}\right] \\ \text{CONJ} & \text{NIL} \end{array}\right] \end{array}\right] \\ \text{COORD-DTRS} & \textit{null} \end{array}\right] \end{array}\right]$$

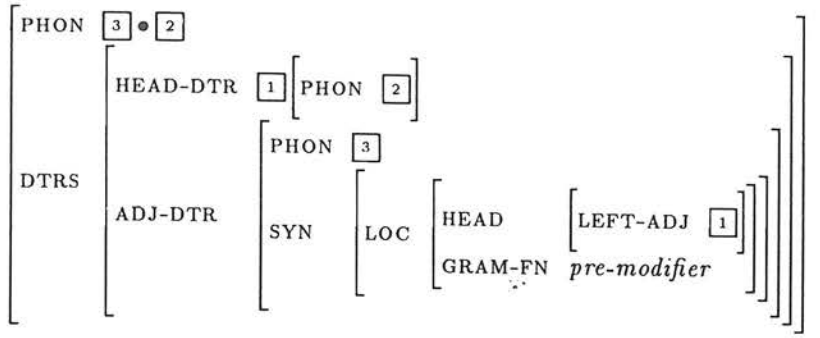
where *constituent*(2)

Γ(head adjunct phrase) =

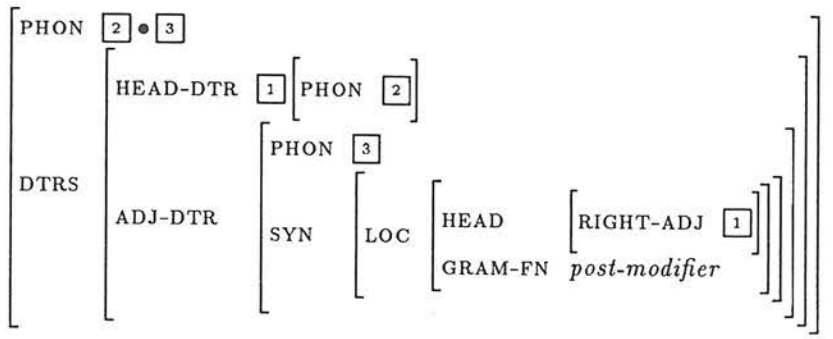
$$\boxed{4} = \left[\begin{array}{ll} \text{SYN} & \left[\begin{array}{ll} \text{LOC} & \left[\begin{array}{ll} \text{SPEC} & \boxed{2} \\ \text{COMPS} & \boxed{3} \end{array}\right] \\ \text{CONJ} & \text{NIL} \end{array}\right] \\ \text{SEM} & \left[\text{CONTENT}\ \boxed{1}\right] \\ \text{DTRS} & \left[\begin{array}{ll} \text{HEAD-DTR} & \left[\text{SYN}\ \left[\text{LOC}\ \left[\begin{array}{ll} \text{SPEC} & \boxed{2} \\ \text{COMPS} & \boxed{3} \end{array}\right]\right]\right] \\ \text{ADJUNCT-DTR}\ \boxed{5} & \left[\begin{array}{ll} \text{SYN} & \left[\begin{array}{ll} \text{LOC} & \left[\begin{array}{ll} \text{SPEC} & \textit{null} \\ \text{COMPS} & \textit{null} \end{array}\right] \\ \text{BIND} & \left[\begin{array}{lll} \text{SLASH} & \text{BOUND} & \{\,\} \\ \text{REL} & \text{BOUND} & \{\,\} \\ \text{QUE} & \text{BOUND} & \{\,\} \end{array}\right] \end{array}\right] \\ \text{SEM} & \left[\text{CONTENT}\ \boxed{1}\right] \end{array}\right] \\ \text{TOPIC-DTR} & \textit{null} \\ \text{COMP-DTRS} & \textit{null} \\ \text{MARKER-DTR} & \textit{null} \end{array}\right] \end{array}\right]$$

where *semantic-inheritance*(4)
and *binding-inheritance*(4)
and *constituent*(5)

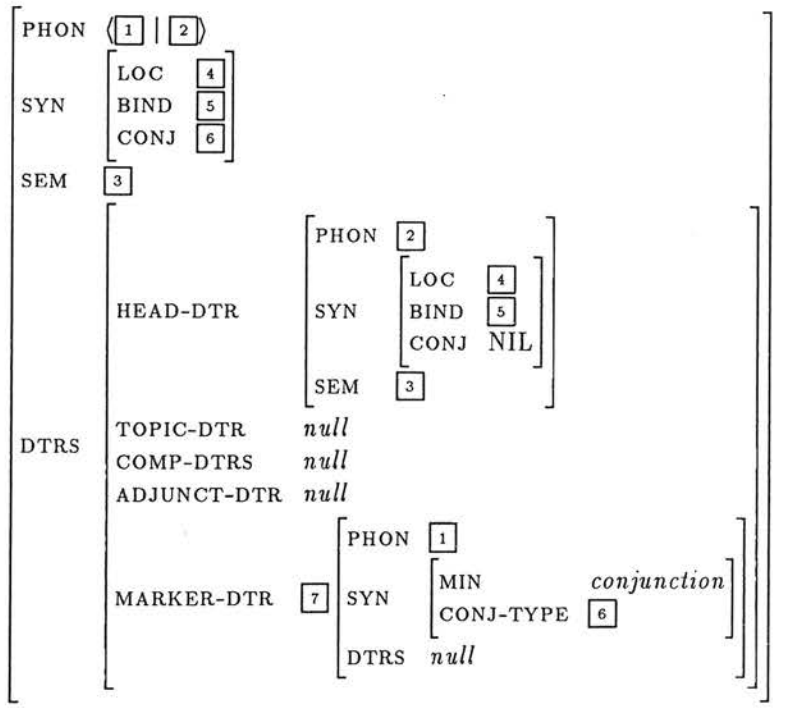
Γ(head left adjunct phrase) =



Γ(head right adjunct phrase) =



Γ(head conjunction phrase) =



where *constituent*(7)

Γ(head argument phrase) =

$$
\boxed{1} = \left[\begin{array}{ll}
\text{SYN} & \left[\begin{array}{ll}
\text{SYN} & \left[\begin{array}{ll}
\text{LOC} & \left[\text{COMPS} \quad \textit{null}\right] \\
\text{CONJ} & \text{NIL}
\end{array}\right]
\end{array}\right] \\
\text{DTRS} & \left[\begin{array}{ll}
\text{HEAD-DTR} & \boxed{2}\left[\text{SYN}\left[\text{BIND}\left[\begin{array}{ll}
\text{REL} & \left[\text{BOUND} \quad \{\,\}\right] \\
\text{QUE} & \left[\text{BOUND} \quad \{\,\}\right]
\end{array}\right]\right]\right] \\
\text{MARKER-DTR} & \textit{null} \\
\text{ADJUNCT-DTR} & \textit{null}
\end{array}\right]
\end{array}\right]
$$

where *semantic-inheritance*(1)
and *binding-inheritance*(1)
and *constituent*(2)

Γ(head complement phrase) =

$$
\left[\begin{array}{ll}
\text{PHON} & \boxed{1} \bullet \boxed{3} \\
\text{SYN} & \left[\text{LOC}\left[\text{SPEC} \quad \boxed{4}\right]\right] \\
\text{SEM} & \left[\text{CONTENT} \quad \boxed{5}\right] \\
\text{DTRS} & \left[\begin{array}{ll}
\text{HEAD-DTR} & \left[\begin{array}{ll}
\text{PHON} & \boxed{1} \\
\text{SYN} & \left[\begin{array}{ll}
\text{LOC} & \left[\begin{array}{ll}
\text{SPEC} & \boxed{4} \\
\text{COMPS} & \boxed{2}
\end{array}\right] \\
\text{BIND} & \left[\text{SLASH}\left[\text{BOUND} \quad \{\,\}\right]\right]
\end{array}\right] \\
\text{SEM} & \left[\text{CONTENT} \quad \boxed{5}\right] \\
\text{DTRS} & \textit{null}
\end{array}\right] \\
\text{TOPIC-DTR} & \textit{null} \\
\text{COMP-DTRS} & \boxed{2}
\end{array}\right]
\end{array}\right]
$$

where *order-phonology*(3, 2)
and *set-of-constituents*(2)

Γ(head topic phrase) =

$$\begin{bmatrix} \text{PHON} & \boxed{1} \bullet \boxed{2} \\ \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{SPEC} & \textit{null} \end{bmatrix} \end{bmatrix} \\ \text{DTRS} & \begin{bmatrix} \text{HEAD-DTR} & \begin{bmatrix} \text{PHON} & \boxed{2} \\ \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{COMPS} & \textit{null} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{TOPIC-DTR} & \boxed{3} \begin{bmatrix} \text{PHON} & \boxed{1} \\ \text{SYN} & \begin{bmatrix} \text{CONJ} & \text{NIL} \end{bmatrix} \end{bmatrix} \\ \text{COMP-DTRS} & \textit{null} \end{bmatrix} \end{bmatrix}$$

where *constituent*(3)

Γ(head specifier phrase) =

$$\boxed{2} = \begin{bmatrix} \text{DTRS} & \begin{bmatrix} \text{HEAD-DTR} & \begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{SPEC} & \boxed{1} \end{bmatrix} \\ \text{BIND} & \begin{bmatrix} \text{SLASH} & \begin{bmatrix} \text{BOUND} & \{\,\} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{TOPIC-DTR} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

where *build-spec-semantics*(2)

Γ(head filler phrase) =

$$\begin{bmatrix} \text{SEM} & \begin{bmatrix} \text{CONTENT} & \boxed{2} \end{bmatrix} \\ \text{DTRS} & \begin{bmatrix} \text{HEAD-DTR} & \begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{SPEC} & \textit{null} \end{bmatrix} \\ \text{BIND} & \begin{bmatrix} \text{SLASH} & \begin{bmatrix} \text{BOUND} & \{\boxed{1}\} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{CONTENT} & \boxed{2} \end{bmatrix} \end{bmatrix} \\ \text{TOPIC-DTR} & \boxed{1} \begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{BIND} & \begin{bmatrix} \text{SLASH} & \begin{bmatrix} \text{ALL} & \{\,\} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Γ(coordinate phrase) =

$$\begin{bmatrix}
\text{PHON} & \boxed{2} \\
\text{SYN} & \begin{bmatrix}
\text{LOC} & \begin{bmatrix}
\text{HEAD} & \textit{conjoin-head}(\boxed{1}) \\
\text{SPEC} & \textit{unify-spec}(\boxed{1}) \\
\text{COMPS} & \textit{unify-comps}(\boxed{1})
\end{bmatrix} \\
\text{BIND} & \textit{unify-bind}(\boxed{1}) \\
\text{CONJ} & \text{NIL}
\end{bmatrix} \\
\text{DTRS} & \begin{bmatrix}
\text{HEAD-DTR} & \textit{null} \\
\text{TOPIC-DTR} & \textit{null} \\
\text{COMP-DTRS} & \textit{null} \\
\text{ADJUNCT-DTR} & \textit{null} \\
\text{MARKER-DTR} & \textit{null} \\
\text{COORD-DTRS} & \boxed{1}
\end{bmatrix}
\end{bmatrix}$$

where *order-phonology*(□2,□1)
and *set-of-constituents*(□1)

Coordination has not been implemented for the reasons discussed in section 3.9 of Chapter 9.

1.2 Descriptions of Some Lexical Types

Γ(lexical constituent) =

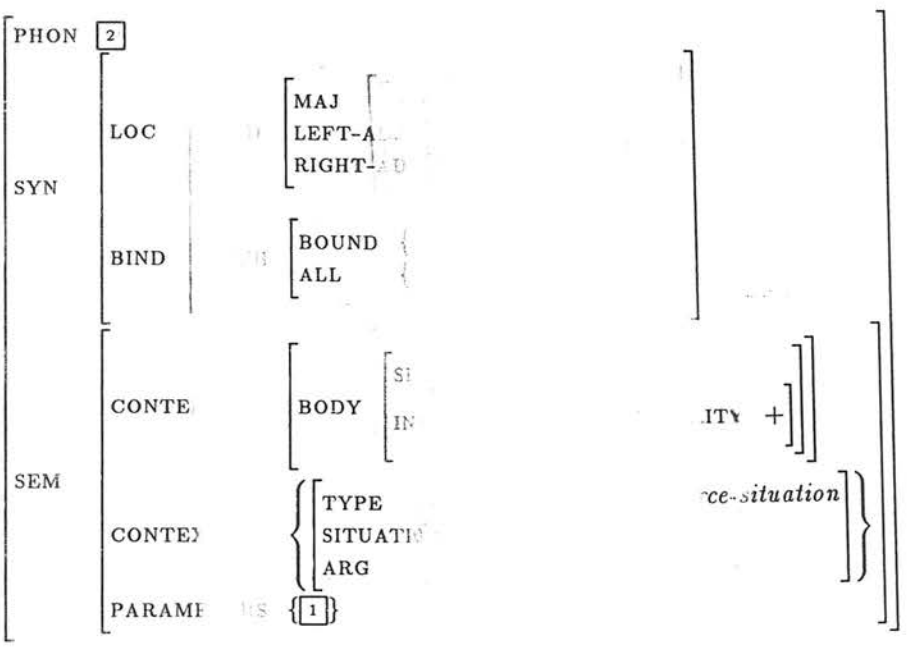
$$\begin{bmatrix}
\text{PHON} & \langle _ \rangle \\
\text{SEM} & \begin{bmatrix} \text{CSTORE} & \{\} \end{bmatrix} \\
\text{DTRS} & \textit{null}
\end{bmatrix}$$

Γ(major lexical constituent) =

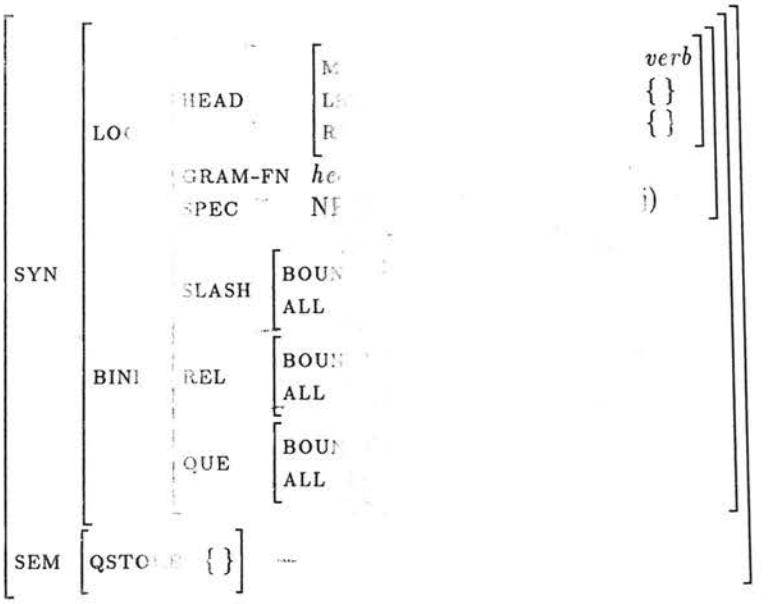
$$\begin{bmatrix}
\text{SYN} & \begin{bmatrix}
\text{LOC} & \begin{bmatrix} \text{HEAD} \\ \text{SPEC} \\ \text{COMPS} \end{bmatrix} \\
\text{BIND} & \begin{bmatrix}
\text{SLASH} & \begin{bmatrix} \text{ALL} & \{\ldots\} \\ \text{BOUND} & \{\ldots\} \end{bmatrix} \\
\text{REL} & \begin{bmatrix} \text{ALL} & \{\ldots\} \\ \text{BOUND} & \{\ldots\} \end{bmatrix} \\
\text{QUE} & \begin{bmatrix} \text{ALL} & \{\ldots\} \\ \text{BOUND} & \{\ldots\} \end{bmatrix}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}$$

Γ(minor lexical constit

Γ(noun) =



Γ(verb) =



Γ(adjective) =

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{adjective} \end{bmatrix} \end{bmatrix} \\ \text{BIND} & \begin{bmatrix} \text{SLASH} & \begin{bmatrix} \text{BOUND} & \{\,\} \\ \text{ALL} & \{\,\} \end{bmatrix} \\ \text{REL} & \begin{bmatrix} \text{BOUND} & \{\,\} \\ \text{ALL} & \{\,\} \end{bmatrix} \\ \text{QUE} & \begin{bmatrix} \text{BOUND} & \{\,\} \\ \text{ALL} & \{\,\} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{CONTEXT} & \{\,\} \\ \text{PARAMETERS} & \{\,\} \\ \text{QSTORE} & \{\,\} \end{bmatrix} \end{bmatrix}$$

Γ(adverb) =

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{adverb} \\ \text{PRD} & - \\ \text{LEFT-ADJ} & \{\boxed{1}\} \\ \text{RIGHT-ADJ} & \{\boxed{1}\} \end{bmatrix} \\ \text{SPEC} & \textit{null} \\ \text{COMPS} & \textit{null} \end{bmatrix} \\ \text{BIND} & \begin{bmatrix} \text{SLASH} & \begin{bmatrix} \text{BOUND} & \{\,\} \\ \text{ALL} & \{\,\} \end{bmatrix} \\ \text{REL} & \begin{bmatrix} \text{BOUND} & \{\,\} \\ \text{ALL} & \{\,\} \end{bmatrix} \\ \text{QUE} & \begin{bmatrix} \text{BOUND} & \{\,\} \\ \text{ALL} & \{\,\} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{CONTEXT} & \{\,\} \\ \text{PARAMETERS} & \{\,\} \\ \text{QSTORE} & \{\,\} \end{bmatrix} \end{bmatrix}$$

where $\boxed{1}$ is $\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{COMPS} & \textit{null} \\ \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{verb} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$

Γ(preposition) =

$$\left[\begin{array}{ll}
\text{SYN} & \left[\begin{array}{ll}
\text{LOC} & \left[\begin{array}{ll}
\text{HEAD} & \left[\begin{array}{ll}\text{MAJ} & \textit{preposition}\\ \text{LEFT-ADJ} & \{\,\}\\ \text{RIGHT-ADJ} & \{\boxed{1},\boxed{2}\}\end{array}\right]\\
\text{GRAM-FN} & \textit{head}\\
\text{SPEC} & \textit{null}\\
\text{COMPS} & \{\text{NP(norm,acc,obj)}\}
\end{array}\right]\\
\text{BIND} & \left[\begin{array}{ll}
\text{SLASH} & \left[\begin{array}{ll}\text{BOUND} & \{\,\}\\ \text{ALL} & \{\,\}\end{array}\right]\\
\text{REL} & \left[\begin{array}{ll}\text{BOUND} & \{\,\}\\ \text{ALL} & \{\,\}\end{array}\right]\\
\text{QUE} & \left[\begin{array}{ll}\text{BOUND} & \{\,\}\\ \text{ALL} & \{\,\}\end{array}\right]
\end{array}\right]
\end{array}\right]\\
\text{SEM} & \left[\begin{array}{ll}\text{CONTEXT} & \{\,\}\\ \text{PARAMETERS} & \{\,\}\\ \text{QSTORE} & \{\,\}\end{array}\right]
\end{array}\right]$$

where $\boxed{1}$ is $\left[\text{SYN}\ \left[\text{LOC}\ \left[\begin{array}{ll}\text{SPEC} & [\ ]\\ \text{COMPS} & \textit{null}\\ \text{HEAD} & \left[\text{MAJ}\ \ \textit{verb}\right]\end{array}\right]\right]\right]$

and $\boxed{2}$ is $\left[\text{SYN}\ \left[\text{LOC}\ \left[\begin{array}{ll}\text{SPEC} & \textit{null}\\ \text{COMPS} & \textit{null}\\ \text{HEAD} & \left[\text{MAJ}\ \ \textit{noun}\right]\end{array}\right]\right]\right]$

Γ(determiner) =

$$\left[\begin{array}{ll}
\text{SYN} & \left[\begin{array}{ll}
\text{LOC} & \left[\begin{array}{ll}
\text{HEAD} & \left[\begin{array}{ll}
\text{MAJ} & \textit{determiner} \\
\text{PRD} & - \\
\text{LEFT-ADJ} & \{\,\} \\
\text{RIGHT-ADJ} & \{\,\}
\end{array}\right] \\
\text{SPEC} & \textit{null} \\
\text{COMPS} & \textit{null}
\end{array}\right] \\
\text{BIND} & \left[\begin{array}{ll}
\text{SLASH} & \left[\begin{array}{ll}\text{BOUND} & \{\,\} \\ \text{ALL} & \{\,\}\end{array}\right] \\
\text{REL} & \left[\begin{array}{ll}\text{BOUND} & \{\,\} \\ \text{ALL} & \{\,\}\end{array}\right]
\end{array}\right]
\end{array}\right] \\
\text{SEM} & \left[\begin{array}{ll}
\text{CONTEXT} & \{\,\} \\
\text{PARAMETERS} & \{\,\} \\
\text{QSTORE} & \{\,\}
\end{array}\right]
\end{array}\right]$$

2 The Lexicon

The statement of the grammar is completed by a lexicon in the form of a list of word-class (or token-type) pairs supplemented with idiosyncratic features for each word. Such a list is partially given here:

Word	Type	Idiosyncratic Features
and	and	$\left[\text{PHON } \langle and \rangle\right]$
both	both	$\left[\text{PHON } \langle both \rangle\right]$
but	but	$\left[\text{PHON } \langle but \rangle\right]$
either	either	$\left[\text{PHON } \langle either \rangle\right]$
neither	neither	$\left[\text{PHON } \langle neither \rangle\right]$
nor	nor	$\left[\text{PHON } \langle nor \rangle\right]$
or	or	$\left[\text{PHON } \langle or \rangle\right]$
cat	plain-noun	$\left[\text{PHON } \langle cat \rangle\right]$
dog	plain-noun	$\left[\text{PHON } \langle dog \rangle\right]$
kitten	plain-noun	$\left[\text{PHON } \langle kitten \rangle\right]$
man	plain-noun	$\left[\text{PHON } \langle man \rangle\right]$

woman	plain-noun	[PHON ⟨*woman*⟩]
picture	picture-noun	[PHON ⟨*picture*⟩]
Fido	proper	[PHON ⟨*Fido*⟩]
Tigger	proper	[PHON ⟨*Tigger*⟩]
Tom	proper	[PHON ⟨*Tom*⟩]
Rome	proper	[PHON ⟨*Rome*⟩]
Rover	proper	[PHON ⟨*Rover*⟩]
he	nom	[PHON ⟨*he*⟩]
she	nom	[PHON ⟨*she*⟩]
who	nom	[PHON ⟨*who*⟩]
him	acc	[PHON ⟨*him*⟩]
her	acc	[PHON ⟨*her*⟩]
whom	acc	[PHON ⟨*whom*⟩]
it	expletive	[PHON ⟨*it*⟩ SYN [LOC [HEAD [FORM *it*]]]]
there	expletive	[PHON ⟨*there*⟩ SYN [LOC [HEAD [FORM *there*]]]]
black	normal	[PHON ⟨*black*⟩]
furry	normal	[PHON ⟨*furry*⟩]
small	normal	[PHON ⟨*small*⟩]
easy	tough	[PHON ⟨*easy*⟩]
tough	tough	[PHON ⟨*tough*⟩]

loudly	adverb	$\left[\text{PHON}\ \langle loudly \rangle\right]$
quickly	adverb	$\left[\text{PHON}\ \langle quickly \rangle\right]$
about	preposition	$\left[\begin{array}{ll}\text{PHON} & \langle about \rangle \\ \text{SYN} & \left[\text{LOC}\ \left[\text{HEAD}\ \left[\text{FORM}\ about\right]\right]\right]\end{array}\right]$
by	preposition	$\left[\begin{array}{ll}\text{PHON} & \langle by \rangle \\ \text{SYN} & \left[\text{LOC}\ \left[\text{HEAD}\ \left[\text{FORM}\ by\right]\right]\right]\end{array}\right]$
from	preposition	$\left[\begin{array}{ll}\text{PHON} & \langle from \rangle \\ \text{SYN} & \left[\text{LOC}\ \left[\text{HEAD}\ \left[\text{FORM}\ from\right]\right]\right]\end{array}\right]$
in	preposition	$\left[\begin{array}{ll}\text{PHON} & \langle in \rangle \\ \text{SYN} & \left[\text{LOC}\ \left[\text{HEAD}\ \left[\text{FORM}\ in\right]\right]\right]\end{array}\right]$
of	preposition	$\left[\begin{array}{ll}\text{PHON} & \langle of \rangle \\ \text{SYN} & \left[\text{LOC}\ \left[\text{HEAD}\ \left[\text{FORM}\ of\right]\right]\right]\end{array}\right]$
on	preposition	$\left[\begin{array}{ll}\text{PHON} & \langle on \rangle \\ \text{SYN} & \left[\text{LOC}\ \left[\text{HEAD}\ \left[\text{FORM}\ on\right]\right]\right]\end{array}\right]$
with	preposition	$\left[\begin{array}{ll}\text{PHON} & \langle with \rangle \\ \text{SYN} & \left[\text{LOC}\ \left[\text{HEAD}\ \left[\text{FORM}\ with\right]\right]\right]\end{array}\right]$
to	preposition	$\left[\begin{array}{ll}\text{PHON} & \langle to \rangle \\ \text{SYN} & \left[\text{LOC}\ \left[\text{HEAD}\ \left[\text{FORM}\ to\right]\right]\right]\end{array}\right]$
bounce	intransitive	$\left[\text{PHON}\ \langle bounce \rangle\right]$
miaow	intransitive	$\left[\text{PHON}\ \langle miaow \rangle\right]$
rain	intransitive	$\left[\text{PHON}\ \langle rain \rangle\right]$

run	intransitive	[PHON ⟨*run*⟩]
talk	intransitive	[PHON ⟨*talk*⟩]
chase	transitive	[PHON ⟨*chase*⟩]
kiss	transitive	[PHON ⟨*kiss*⟩]
love	transitive	[PHON ⟨*love*⟩]
give	ditransitive	[PHON ⟨*give*⟩]
sell	ditransitive	[PHON ⟨*sell*⟩]
argue	anom	[PHON ⟨*argue*⟩]
believe	anom	[PHON ⟨*believe*⟩]
appear	intrans-rais	[PHON ⟨*appear*⟩]
seem	intrans-rais	[PHON ⟨*seem*⟩]
try	intrans-equi	[PHON ⟨*try*⟩]
prefer	intrans-equi	[PHON ⟨*prefer*⟩]
pretend	intrans-equi	[PHON ⟨*pretend*⟩]
believe	trans-rais	[PHON ⟨*believe*⟩]
hope	trans-rais	[PHON ⟨*hope*⟩]
know	trans-rais	[PHON ⟨*know*⟩]
persuade	object-equi	[PHON ⟨*persuade*⟩]
promise	subject-equi	[PHON ⟨*promise*⟩]
have	perfective	[PHON ⟨*have*⟩]
be	copula	[PHON ⟨*be*⟩]
been	copula	[PHON ⟨*been*⟩]

can	modal	[PHON ⟨*can*⟩]
could	modal	[PHON ⟨*could*⟩]
may	modal	[PHON ⟨*may*⟩]
might	modal	[PHON ⟨*might*⟩]
should	modal	[PHON ⟨*should*⟩]
will	modal	[PHON ⟨*will*⟩]
would	modal	[PHON ⟨*would*⟩]
to	infinitive	[PHON ⟨*to*⟩]
a	plain-det	[PHON ⟨*a*⟩ SEM [CONTENT [QUANTIFIER *a*]]]
the	plain-det	[PHON ⟨*the*⟩ SEM [CONTENT [QUANTIFIER *the*]]]
some	plain-det	[PHON ⟨*some*⟩ SEM [CONTENT [QUANTIFIER *some*]]]
no	plain-det	[PHON ⟨*no*⟩ SEM [CONTENT [QUANTIFIER *no*]]]
every	plain-det	[PHON ⟨*every*⟩ SEM [CONTENT [QUANTIFIER *every*]]]
most	plain-det	[PHON ⟨*most*⟩ SEM [CONTENT [QUANTIFIER *most*]]]
which	wh-det	[PHON ⟨*which*⟩ SEM [CONTENT [QUANTIFIER *which*]]]

3 Abbreviations

This section lists some abbreviations used throughout both the thesis and this appendix. Not listed are things like NP(nom,norm), which can be constructed by unifying the abbreviations of NP(nom) and NP(norm).

$NP_{[i]}$:

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{noun} \\ \text{LEFT-ADJOINS} & \textit{null} \\ \text{RIGHT-ADJOINS} & \textit{null} \end{bmatrix} \\ \text{SPEC} & \textit{null} \\ \text{COMPS} & \textit{null} \end{bmatrix} \\ \text{CONJ} & \text{NIL} \end{bmatrix} \\ \text{SEM} & [i] \end{bmatrix}$$

$NP(x)_{[i]}$:

$$NP_{[i]} \sqcup \begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{CASE} & x \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

where $x \in \{\text{nom, acc}\}$.

$NP(x)_{[i]}$:

$$NP_{[i]} \sqcup \begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{FORM} & x \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

where $x \in \{\text{norm, it, there}\}$.

$NP(x)_{[i]}$:

$$NP_{[i]} \sqcup \begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{GRAM-FN} & x \end{bmatrix} \end{bmatrix}$$

where $x \in \{\text{subj, obj, i-obj}\}$.

$VP_{[i]}$:

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{verb} \\ \text{LEFT-ADJOINS} & \textit{null} \\ \text{RIGHT-ADJOINS} & \textit{null} \end{bmatrix} \\ \text{SPEC} & \text{NP} \\ \text{COMPS} & \textit{null} \end{bmatrix} \\ \text{CONJ} & \text{NIL} \end{bmatrix} \\ \text{SEM} & [i] \end{bmatrix}$$

$VP(x)_{[i]}$:

$$VP_{[i]} \sqcup \begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{FORM} & x \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

where $x \in \{\text{bse, fin, psp, prp, pas, inf, ger}\}$.

$VP(x)_{[i]}$:

$$VP_{[i]} \sqcup \begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{GRAM-FN} & x \end{bmatrix} \end{bmatrix}$$

where $x \in$ {head, xcomp}.

$\text{PP}_{\boxed{i}}$:

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{preposition} \end{bmatrix} \\ \text{SPEC} & \text{NP} \\ \text{COMPS} & \textit{null} \end{bmatrix} \\ \text{CONJ} & \textit{romeNIL} \end{bmatrix} \\ \text{SEM} & \boxed{i} \end{bmatrix}$$

$\text{PP}(x)_{\boxed{i}}$:

$$\text{PP}_{\boxed{i}} \sqcup \begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{FORM} & x \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

where $x \in$ {to, of, in, by, from, ...}.

$\text{PP}(x)_{\boxed{i}}$:

$$\text{PP}_{\boxed{i}} \sqcup \begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{GRAM-FN} & x \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

where $x \in$ {obj, i-obj, post-modifier}.

DET:

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{determiner} \\ \text{LEFT-ADJOINS} & \{\,\} \\ \text{RIGHT-ADJOINS} & \{\,\} \end{bmatrix} \\ \text{SPEC} & \textit{null} \\ \text{COMPS} & \textit{null} \end{bmatrix} \\ \text{BIND} & \begin{bmatrix} \text{SLASH} & \begin{bmatrix} \text{ALL} & \{\,\} \\ \text{BOUND} & \{\,\} \end{bmatrix} \\ \text{REL} & \begin{bmatrix} \text{ALL} & \{\,\} \\ \text{BOUND} & \{\,\} \end{bmatrix} \end{bmatrix} \\ \text{CONJ} & \textit{romeNIL} \end{bmatrix} \end{bmatrix}$$

4 Some Lexical Rules

This section lists some CPSG lexical rules. The notation

$$\begin{array}{c} A \\ \Downarrow \\ B \end{array}$$

is interpreted as meaning for any (lexical) constituent x whose description X is subsumed by A, there exists a related constituent described by X/B (read X in the context of B), the priority unification of X with B (where B takes priority).

4.1 Inflectional Lexical Rules

4.1.1 Plural Noun Rules (PNR):

$$\begin{bmatrix} \text{PHON } \boxed{1} \\ \text{SYN } \begin{bmatrix} \text{LOC } \begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{MAJ } \textit{noun} \end{bmatrix} \\ \text{SPEC } [\ ] \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} \text{PHON plural-inflection}(\boxed{1}) \end{bmatrix}$$

$$\begin{bmatrix} \text{PHON } \boxed{1} \\ \text{SYN } \begin{bmatrix} \text{LOC } \begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{MAJ } \textit{noun} \end{bmatrix} \\ \text{SPEC } [\ ] \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} \text{PHON plural-inflection}(\boxed{1}) \\ \text{SYN } \begin{bmatrix} \text{LOC } \begin{bmatrix} \text{SPEC } \textit{null} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

4.1.2 Verb Form Rules (VFR):

$$\begin{bmatrix} \text{PHON } \boxed{1} \\ \text{SYN } \begin{bmatrix} \text{LOC } \begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{MAJ } \textit{verb} \\ \text{FORM } \textit{bse} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} \text{PHON finite-inflection}(\boxed{1}) \\ \text{SYN } \begin{bmatrix} \text{LOC } \begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{FORM } \textit{fin} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \text{PHON } \boxed{1} \\ \text{SYN } \begin{bmatrix} \text{LOC } \begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{MAJ } \textit{verb} \\ \text{FORM } \textit{bse} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} \text{PHON prp-inflection}(\boxed{1}) \\ \text{SYN } \begin{bmatrix} \text{LOC } \begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{FORM } \textit{prp} \\ \text{PRD } \textit{pos} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

4.2 Non-Inflectional Rules

4.2.1 Slash Termination Rule 1 (STR1):

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{verb} \end{bmatrix} \\ \text{COMPS} & \{\boxed{1}, \ldots\} \end{bmatrix} \\ \text{BIND} & \begin{bmatrix} \text{SLASH} & \begin{bmatrix} \text{ALL} & \{\,\} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{COMPS} & \{\ldots\} \end{bmatrix} \\ \text{BIND} & \begin{bmatrix} \text{SLASH} & \begin{bmatrix} \text{ALL} & \{\boxed{1}\} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

4.2.2 Slash Termination Rule 2 (STR2):

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{COMPS} & \{\text{S},\ldots\} \end{bmatrix} \\ \text{BIND} & \begin{bmatrix} \text{SLASH} & \begin{bmatrix} \text{ALL} & \{\,\} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{COMPS} & \{\text{VP},\ldots\} \end{bmatrix} \\ \text{BIND} & \begin{bmatrix} \text{SLASH} & \begin{bmatrix} \text{ALL} & \{\text{NP}\} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

4.2.3 Subject-Auxiliary Inversion Rule (SAIR):

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{verb} \\ \text{FORM} & \textit{fin} \\ \text{INV} & - \\ \text{AUX} & + \end{bmatrix} \\ \text{SPEC} & \boxed{1} \\ \text{COMP} & \{\boxed{2}\} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{MAJ} & \textit{verb} \\ \text{FORM} & \textit{fin} \\ \text{INV} & + \\ \text{AUX} & + \end{bmatrix} \\ \text{SPEC} & \textit{null} \\ \text{COMP} & \{\boxed{1}, \boxed{2}\} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Appendix B

Program Listing

1 foundations.pl

```
/************************************************************************

        File:                   foundations.pl
        Author:                 Richard Cooper
        Date:                   26/10/89

This file contains all the essential little bits, most importantly
operator definitions.  Also included are the clauses for unification
and priority unification.  The file must always be loaded first.

************************************************************************/

:- op(700,xfx,'<=>').
:- op(700,xfx,'<<=').
:- op(800,xfx,':').
:- op(700,xfx,abbreviates).

/************************************************************************/

path(X) :-
        var(X), !, fail.
path(_/_).

/************************************************************************/

member(H, [H|_]).
member(X, [_|T]) :-
        member(X, T).

/************************************************************************/

delete(H, [H|T], T).
delete(X, [H|T], [H|T2]) :-
        delete(X, T, T2).

/************************************************************************/
```

```
append([], L, L).
append([H|T], L, [H|NL]) :-
        append(T, L, NL).

/*********************************************************************/

prefix([H|_], [H]).
prefix([H|T], [H|L]) :-
        prefix(T, L).

/*********************************************************************/

union(set(L), set(M), set(N)) :-
        append(L, M, N).

/*********************************************************************/

difference(set(L), set([]), set(L)).
difference(set(L), set([H|T]), set(D)) :-
        delete(H, L, M),
        difference(set(M), set(T), set(D)), !.

/*********************************************************************/

concat(list([]), list(X), list(X)).
concat(list([H|T]), list(X), list([H|Y])) :-
        concat(list(T), list(X), list(Y)).

/*********************************************************************/

non_empty_concat(X, Y, Z) :-
        concat(X, Y, Z),
        non_empty_list(X),
        non_empty_list(Y).

/*********************************************************************/

non_empty_list(list([_|_])).

/*********************************************************************/

maximum(Y1, Y2, Y1) :-
        Y1 >= Y2, !.
maximum(Y1, Y2, Y2) :-
        Y2 >= Y1, !.

/*********************************************************************/

uninstantiated_avm((T,L)) :-
        var(T).

/*********************************************************************/
```

```
% Unification:

S1 <=> S2 :-
        value(S1, FS1), !,
        value(S2, FS2), !,
        unify_feature_structures(FS1, FS2).

/*********************************************************************/

unify_feature_structures(A, A) :-
        atom(A), !.
unify_feature_structures(V, V) :-
        var(V), !.
unify_feature_structures((Sort, AVMList1), (Sort, AVMList2)) :- !,
        unify_feature_structure_lists(AVMList1, AVMList2).
unify_feature_structures(list(L), list(L)) :- !.
unify_feature_structures(set(S1), set(S2)) :-
        (var(S1); var(S2)), !,
        S1 = S2.
unify_feature_structures(set(S1), set(S2)) :-
        S1 = [H|T1],
        delete(H, S2, T2),
        unify_feature_structures(set(T1), set(T2)).
unify_feature_structures(set([]), set([])) :- !.

/*********************************************************************/

unify_feature_structure_lists(L1, L2) :-
        (var(L1); var(L2)), !,
        L1 = L2.
unify_feature_structure_lists([H1|T1], [H2|T2]) :- !,
        unify_feature_structures(H1, H2),
        unify_feature_structure_lists(T1, T2).
unify_feature_structure_lists([], []) :- !.

/*********************************************************************/

% Priority Unification:

S1 <<= S2 :-
        value(S1, FS1), !,
        value(S2, FS2), !,
        priority_union(FS1, FS2), !.

/*********************************************************************/

priority_union(L, _) :-
        atom(L), !.
priority_union(_, M) :-
        var(M), !.
priority_union(L, M) :-
        var(L), !,
```

```
        L = M.
priority_union(_, M) :-
        atom(M), !.
priority_union((Sort, L1), (Sort, L2)) :- !,
        priority_union_lists(L1, L2).
priority_union(list([]), list([])) :- !.
priority_union(list([H1|T1]), list([H2|T2])) :-
        priority_union(H1,H2),
        priority_union(list(T1), list(T2)), !.
priority_union(set(_), set(_)) :- !.

/*************************************************************************/

priority_union_lists(L1, L2) :-
        var(L1), !,
        L1 = L2.
priority_union_lists([H1|T1], [H2|T2]) :- !,
        priority_union(H1, H2),
        priority_union_lists(T1, T2).
priority_union_lists([], []) :- !.

/*************************************************************************/
```

2 relational_dependencies.pl

```
/*************************************************************************

        File:                   relational_dependencies.pl
        Author:                 Richard Cooper
        Date:                   26/10/89

This file contains the definitions of some relational dependencies:
order-phonology; set-of-constituents; binding-inheritance;
semantic-inheritance; and build-spec-semantics.

*************************************************************************/

% A relational dependency: set-of-constituents

set_of_constituents(set([])).
set_of_constituents(set([H|T])) :-
        constituent(H),
        set_of_constituents(set(T)).

/*************************************************************************/

% A relational dependency: order-phonology

order_phonology(list([]), set([])) :- !.
order_phonology(list(Phon), set([Sign])) :-
```

```
        Sign/phon <=> list(Phon).
order_phonology(Phon, set([SignA, SignB])) :-
        non_empty_concat(P1, P2, Phon),
        SignA/syn/loc/gram_fn <=> GRA,
        SignB/syn/loc/gram_fn <=> GRB,
        relative_order(SignA, SignB, GRA, GRB, Sign1, Sign2),
        Sign1/phon <=> P1,
        Sign2/phon <=> P2.

/*****************************************************************/

relative_order(SignA, SignB, GR, GR, SignA, SignB).
relative_order(SignA, SignB, GR, GR, SignB, SignA).
relative_order(SignA, SignB, GRA, GRB, SignA, SignB) :-
        lp_rule( GRA < GRB).
relative_order(SignA, SignB, GRA, GRB, SignB, SignA) :-
        lp_rule( GRB < GRA).

/*****************************************************************/

lp_rule( obj < i_obj ).
lp_rule( obj < xcomp ).
lp_rule( subj < xcomp ).
lp_rule( _ < post_modifier ).
lp_rule( pre_modifier < _ ).

/*****************************************************************/

% A relational dependency: binding-inheritance

binding_inheritance(Sign) :-
        get_dtrs_list(Sign, DtrsList),
        merge_slash(DtrsList, Slash),
        merge_rel(DtrsList, Rel),
        merge_que(DtrsList, Que),
        Sign/syn/bind/slash/all <=> Slash,
        Sign/syn/bind/rel/all <=> Rel,
        Sign/syn/bind/que/all <=> Que.

/*****************************************************************/

get_dtrs_list(AVM, DtrsList) :-
        (daughters, AllDtrs) <=> AVM/dtrs,
        extract_dtrs_list(AllDtrs, DtrsList).

/*****************************************************************/

extract_dtrs_list([], []).
extract_dtrs_list([null|T], NT) :- !,
        extract_dtrs_list(T, NT).
extract_dtrs_list([set(L)|T], NT) :- !,
        extract_dtrs_list(T, NT1),
        append(L, NT1, NT).
```

```
extract_dtrs_list([D|T], [D|NT]) :- !,
        extract_dtrs_list(T, NT).

/*****************************************************************/

merge_slash([], set([])).
merge_slash([Dtr|OtherDtrs], TotalSlash) :-
        Dtr/syn/bind/slash/all <=> All,
        Dtr/syn/bind/slash/bound <=> Bound,
        difference(All, Bound, Slash),
        merge_slash(OtherDtrs, MoreSlash),
        union(Slash, MoreSlash, TotalSlash).

/*****************************************************************/

merge_rel([], set([])).
merge_rel([Dtr|OtherDtrs], TotalRel) :-
        Dtr/syn/bind/rel/all <=> All,
        Dtr/syn/bind/rel/bound <=> Bound,
        difference(All, Bound, Rel),
        merge_rel(OtherDtrs, MoreRel),
        union(Rel, MoreRel, TotalRel).

/*****************************************************************/

merge_que([], set([])).
merge_que([Dtr|OtherDtrs], TotalQue) :-
        Dtr/syn/bind/que/all <=> All,
        Dtr/syn/bind/que/bound <=> Bound,
        difference(All, Bound, Que),
        merge_que(OtherDtrs, MoreQue),
        union(Que, MoreQue, TotalQue).

/*****************************************************************/

% Relational dependency: semantic-inheritance

semantic_inheritance(Sign) :-
        get_dtrs_list(Sign, DtrsList),
        Sign/sem/context <=> Context,
        collect_context(DtrsList, Context),
        Sign/sem/parameters <=> Parameters,
        collect_parameters(DtrsList, Parameters),
        Sign/sem/qstore <=> QStore,
        collect_qstore(DtrsList, QStore),
        Sign/sem/cstore <=> CStore,
        collect_cstore(DtrsList, CStore).

/*****************************************************************/

collect_context([Dtr], Context) :- !,
        Dtr/sem/context <=> Context.
collect_context([Dtr|OtherDtrs], Context) :-
```

```
        Dtr/sem/context <=> SomeContext,
        collect_context(OtherDtrs, OtherContext),
        union(SomeContext, OtherContext, Context).

/*****************************************************************/

collect_parameters([Dtr], Parameters) :- !,
        Dtr/sem/parameters <=> Parameters.
collect_parameters([Dtr|OtherDtrs], Parameters) :-
        Dtr/sem/parameters <=> SomeParameters,
        collect_parameters(OtherDtrs, OtherParameters),
        union(SomeParameters, OtherParameters, Parameters).

/*****************************************************************/

collect_qstore([Dtr], QStore) :- !,
        Dtr/sem/qstore <=> QStore.
collect_qstore([Dtr|OtherDtrs], QStore) :-
        Dtr/sem/qstore <=> SomeQStore,
        collect_qstore(OtherDtrs, OtherQStore),
        union(SomeQStore, OtherQStore, QStore).

/*****************************************************************/

collect_cstore([Dtr], CStore) :- !,
        Dtr/sem/cstore <=> CStore.
collect_cstore([Dtr|OtherDtrs], CStore) :-
        Dtr/sem/cstore <=> SomeCStore,
        collect_cstore(OtherDtrs, OtherCStore),
        union(SomeCStore, OtherCStore, CStore).

/*****************************************************************/

% Relational dependency: build-spec-semantics

build_spec_semantics(Temp) :-
        Temp/syn/loc/head/maj <=> noun, !,
        Temp/sem/content <=> Temp/dtrs/topic_dtr/sem/content/tag.

build_spec_semantics(Temp) :-
        Temp/sem/content <=> Temp/dtrs/head_dtr/sem/content.

/*****************************************************************/
```

3 lexical_hierarchy.pl

```
/*****************************************************************

        File:                   lexical_hierarchy.pl
        Author:                 Richard Cooper
```

```
        Date:                23/10/89

This file contains the lexical hierachy, expressed in terms of of
subtype/2 clauses, specifying the shape of the hierarchy and satisfies/2
clauses, specifying the properties associated with each type.

************************************************************************/
/************************************************************************

                          subtype(Type, SubType).

************************************************************************/

% Non-leaf types:

subtype(lexical_constituent, major_lexical_constituent).
subtype(lexical_constituent, minor_lexical_constituent).

subtype(minor_lexical_constituent, conjunction).

subtype(conjunction, and).
subtype(conjunction, both).
subtype(conjunction, but).
subtype(conjunction, or).
subtype(conjunction, nor).
subtype(conjunction, either).
subtype(conjunction, neither).

subtype(major_lexical_constituent, preposition).
subtype(major_lexical_constituent, noun).
subtype(major_lexical_constituent, adjective).
subtype(major_lexical_constituent, verb).
subtype(major_lexical_constituent, adverb).
subtype(major_lexical_constituent, determiner).

subtype(noun, common).
subtype(noun, saturated).

subtype(common, plain_common).
subtype(common, picture_common).

subtype(saturated, proper_name).
subtype(saturated, nom_pro).
subtype(saturated, acc_pro).
subtype(saturated, expletive).

subtype(adjective, normal).
subtype(adjective, tough).

subtype(verb, main).
subtype(verb, auxiliary).

subtype(main, ordinary_verb).
```

```
subtype(main, control_verb).

subtype(ordinary_verb, strict_intransitive).
subtype(ordinary_verb, strict_transitive).
subtype(ordinary_verb, strict_ditransitive).
subtype(ordinary_verb, anomolous_verb).

subtype(control_verb, intransitive_control).
subtype(control_verb, transitive_control).

subtype(intransitive_control, intransitive_raising).
subtype(intransitive_control, intransitive_equi).

subtype(transitive_control, transitive_raising).
subtype(transitive_control, transitive_equi).

subtype(transitive_equi, object_equi).
subtype(transitive_equi, subject_equi).

subtype(auxiliary, perfective).
subtype(auxiliary, copula).
subtype(auxiliary, modal).
subtype(auxiliary, infinitive).

subtype(determiner, plain_determiner).
subtype(determiner, wh_determiner).

/*****************************************************************/

% Leaf types:

subtype(and, '$and$').
subtype(both, '$both$').
subtype(but, '$but$').
subtype(either, '$either$').
subtype(neither, '$neither$').
subtype(nor, '$nor$').
subtype(or, '$or$').

subtype(plain_common, '$cat$').
subtype(plain_common, '$dog$').
subtype(plain_common, '$kitten$').
subtype(plain_common, '$man$').
subtype(plain_common, '$woman$').

subtype(picture_common, '$picture$').

subtype(proper_name, '$Fido$').
subtype(proper_name, '$Tigger$').
subtype(proper_name, '$Tom$').
subtype(proper_name, '$Rome$').
subtype(proper_name, '$Rover$').
```

```
subtype(nom_pro, '$he$').
subtype(nom_pro, '$she$').
subtype(nom_pro, '$who$').

subtype(acc_pro, '$him$').
subtype(acc_pro, '$her$').
subtype(acc_pro, '$whom$').

subtype(expletive, '$it$').
subtype(expletive, '$there$').

subtype(normal, '$black$').
subtype(normal, '$furry$').
subtype(normal, '$small$').

subtype(tough, '$easy$').
subtype(tough, '$tough$').

subtype(adverb, '$loudly$').
subtype(adverb, '$quickly$').

subtype(preposition, '$about$').
subtype(preposition, '$by$').
subtype(preposition, '$from$').
subtype(preposition, '$in$').
subtype(preposition, '$of$').
subtype(preposition, '$on$').
subtype(preposition, '$with$').
subtype(preposition, '$to$').

subtype(strict_intransitive, '$bounce$').
subtype(strict_intransitive, '$miaow$').
subtype(strict_intransitive, '$rain$').
subtype(strict_intransitive, '$run$').
subtype(strict_intransitive, '$talk$').

subtype(strict_transitive, '$chase$').
subtype(strict_transitive, '$kiss$').
subtype(strict_transitive, '$love$').

subtype(strict_ditransitive, '$give$').
subtype(strict_ditransitive, '$sell$').

subtype(anomolous_verb, '$argue$').
subtype(anomolous_verb, '$$believe$$').

subtype(intransitive_raising, '$appear$').
subtype(intransitive_raising, '$seem$').

subtype(intransitive_equi, '$try$').
subtype(intransitive_equi, '$prefer$').
subtype(intransitive_equi, '$pretend$').
```

```
subtype(transitive_raising, '$believe$').
subtype(transitive_raising, '$hope$').
subtype(transitive_raising, '$know$').

subtype(object_equi, '$persuade$').

subtype(subject_equi, '$promise$').

subtype(perfective, '$have$').

subtype(copula, '$be$').
subtype(copula, '$been$').

subtype(modal, '$can$').
subtype(modal, '$could$').
subtype(modal, '$may$').
subtype(modal, '$might$').
subtype(modal, '$should$').
subtype(modal, '$will$').
subtype(modal, '$would$').

subtype(infinitive, '$$to$$').

subtype(plain_determiner, '$a$').
subtype(plain_determiner, '$every$').
subtype(plain_determiner, '$no$').
subtype(plain_determiner, '$most$').
subtype(plain_determiner, '$some$').
subtype(plain_determiner, '$the$').

subtype(wh_determiner, '$which$').

/*************************************************************************

                        satisfies(Type, Sign).

*************************************************************************/

% Non-leaf types:

satisfies(lexical_constituent, Sign) :-
        Sign/phon <=> list([_]),
        Sign/sem/cstore <=> set([]),
        Sign/dtrs <=> null.

satisfies(major_lexical_constituent, Sign) :-
        Sign/syn/loc/head <=> _,
        Sign/syn/loc/spec <=> _,
        Sign/syn/loc/comps <=> _,
        Sign/syn/bind/slash/all <=> set(_),
        Sign/syn/bind/rel/all <=> set(_),
        Sign/syn/bind/que/all <=> set(_),
        Sign/syn/bind/slash/bound <=> set(_),
```

```
        Sign/syn/bind/rel/bound <=> set(_),
        Sign/syn/bind/que/bound <=> set(_),
        Sign/syn/conj <=> 'NIL'.

satisfies(minor_lexical_constituent, Sign) :-
        Sign/sem/context <=> set([]),
        Sign/sem/parameters <=> set([]),
        Sign/sem/qstore <=> set([]),
        Sign/syn/min <=> _,
        Sign/syn/conj_type <=> _.

satisfies(conjunction, Sign) :-
        Sign/syn/min <=> conjunction.

satisfies(and, Sign) :-
        Sign/syn/conj_type <=> and.

satisfies(both, Sign) :-
        Sign/syn/conj_type <=> both.

satisfies(but, Sign) :-
        Sign/syn/conj_type <=> but.

satisfies(or, Sign) :-
        Sign/syn/conj_type <=> or.

satisfies(nor, Sign) :-
        Sign/syn/conj_type <=> nor.

satisfies(either, Sign) :-
        Sign/syn/conj_type <=> either.

satisfies(neither, Sign) :-
        Sign/syn/conj_type <=> neither.

satisfies(noun, Sign) :-
        Sign/sem/context <=> set([P]),
        Sign/sem/parameters <=> set([R]),
        P/type <=> resource_situation,
        P/arg <=> Sign/phon,
        P/situation <=> R,
        Sign/sem/content/body/sit  <=> R,
        Sign/sem/content/body/inf/polarity  <=> pos,
        Sign/syn/bind/slash/bound <=> set([]),
        Sign/syn/bind/slash/all <=> set([]),
        Sign/syn/loc/head/left_adjoins <=> set([]),
        Sign/syn/loc/head/right_adjoins <=> set([]),
        Sign/syn/loc/head/maj <=> noun.

satisfies(verb, Sign) :-
        Sign/syn/bind/slash/all <=> set([]),
        Sign/syn/bind/rel/all <=> set([]),
        Sign/syn/bind/que/all <=> set([]),
```

```
        Sign/syn/loc/head/left_adjoins <=> set([]),
        Sign/syn/loc/head/right_adjoins <=> set([]),
        Sign/syn/loc/head/maj <=> verb,
        Sign/syn/loc/gram_fn <=> head,
        Sign/sem/qstore <=> set([]),
        NP abbreviates noun_phrase([nom, subj]),
        NP/syn/bind/slash/bound <=> set([]),
        NP/syn/bind/rel/bound <=> set([]),
        NP/syn/bind/que/bound <=> set([]),
        Sign/syn/loc/spec <=> NP.

satisfies(adjective, Sign) :-
        Sign/sem/content <=> unimplemented,
        Sign/sem/context <=> set([]),
        Sign/sem/parameters <=> set([]),
        Sign/sem/qstore <=> set([]),
        Sign/syn/loc/head/maj <=> adjective,
        Sign/syn/bind/slash/all <=> set([]),
        Sign/syn/bind/slash/bound <=> set([]),
        Sign/syn/bind/rel/all <=> set([]),
        Sign/syn/bind/rel/bound <=> set([]),
        Sign/syn/bind/que/all <=> set([]),
        Sign/syn/bind/que/bound <=> set([]).

satisfies(normal, Sign) :-
        N/syn/loc/head/maj <=> noun,
        N/syn/loc/spec <=> (_,_),
        N/syn/loc/comps <=> null,
        Sign/syn/loc/head/left_adjoins <=> set([N]),
        Sign/syn/loc/head/right_adjoins <=> set([]),
        Sign/syn/loc/comps <=> null,
        Sign/syn/loc/spec <=> null.

satisfies(tough, Sign) :-
        VP abbreviates verb_phrase([inf]),
        NP abbreviates noun_phrase([norm]),
        VP/syn/bind/slash/bound <=> set([NP]),
        Sign/syn/loc/head/left_adjoins <=> set([]),
        Sign/syn/loc/head/right_adjoins <=> set([]),
        Sign/syn/loc/head/prd <=> pos,
        Sign/syn/loc/comps <=> set([VP]),
        Sign/syn/loc/spec <=> VP/syn/loc/spec.

satisfies(adverb, Sign) :-
        Sign/sem/content <=> unimplemented,
        Sign/sem/context <=> set([]),
        Sign/sem/parameters <=> set([]),
        Sign/sem/qstore <=> set([]),
        Sign/syn/bind/slash/all <=> set([]),
        Sign/syn/bind/rel/all <=> set([]),
        Sign/syn/bind/que/all <=> set([]),
        Sign/syn/bind/slash/bound <=> set([]),
        Sign/syn/bind/rel/bound <=> set([]),
```

```
        Sign/syn/bind/que/bound <=> set([]),
        V/syn/loc/head/maj <=> verb,
        V/syn/loc/comps <=> null,
        Sign/syn/loc/head/left_adjoins <=> set([V]),
        Sign/syn/loc/head/right_adjoins <=> set([V]),
        Sign/syn/loc/head/maj <=> adverb,
        Sign/syn/loc/head/prd <=> neg,
        Sign/syn/loc/spec <=> null,
        Sign/syn/loc/comps <=> null.

satisfies(preposition, Sign) :-
        Sign/sem/content <=> unimplemented,
        Sign/sem/context <=> set([]),
        Sign/sem/parameters <=> set([]),
        Sign/sem/qstore <=> set([]),
        NP abbreviates noun_phrase([norm, acc, obj]),
        Sign/syn/bind/slash/all <=> set([]),
        Sign/syn/bind/rel/all <=> set([]),
        Sign/syn/bind/que/all <=> set([]),
        Sign/syn/bind/slash/bound <=> set([]),
        Sign/syn/bind/rel/bound <=> set([]),
        Sign/syn/bind/que/bound <=> set([]),
        N/syn/loc/head/maj <=> noun,
        N/syn/loc/spec <=> null,
        N/syn/loc/comps <=> null,
        V/syn/loc/head/maj <=> verb,
        V/syn/loc/spec <=> (_,_),
        V/syn/loc/comps <=> null,
        Sign/syn/loc/head/maj <=> preposition,
        Sign/syn/loc/head/left_adjoins <=> set([]),
        Sign/syn/loc/head/right_adjoins <=> set([V, N]),
        Sign/syn/loc/gram_fn <=> head,
        Sign/syn/loc/spec <=> null,
        Sign/syn/loc/comps <=> set([NP]).

satisfies(determiner, Sign) :-
        Sign/sem/context <=> set([]),
        Sign/sem/parameters <=> set([]),
        Sign/sem/qstore <=> set([]),
        Sign/syn/loc/head/left_adjoins <=> set([]),
        Sign/syn/loc/head/right_adjoins <=> set([]),
        Sign/syn/loc/head/maj <=> determiner,
        Sign/syn/loc/head/prd <=> neg,
        Sign/syn/loc/spec <=> null,
        Sign/syn/loc/comps <=> null,
        Sign/syn/bind/slash/all <=> set([]),
        Sign/syn/bind/rel/all <=> set([]),
        Sign/syn/bind/slash/bound <=> set([]),
        Sign/syn/bind/rel/bound <=> set([]).

satisfies(plain_determiner, Sign) :-
        Sign/syn/bind/que/bound <=> set([]),
        Sign/syn/bind/que/all <=> set([]).
```

```
satisfies(wh_determiner, Sign) :-
        Sign/syn/bind/que/all <=> set([_]).

satisfies(common, Sign) :-
        DET abbreviates determiner,
        DET/syn/loc/gram_fn <=> spec,
        Sign/syn/bind/rel/all <=> set([]),
        Sign/syn/bind/que/all <=> set([]),
        Sign/syn/bind/rel/bound <=> set([]),
        Sign/syn/bind/que/bound <=> set([]),
        Sign/syn/loc/head/form <=> norm,
        Sign/syn/loc/head/case <=> DET/syn/loc/head/case,
        Sign/syn/loc/gram_fn <=> head,
        Sign/syn/loc/spec <=> DET,
        Sign/sem/qstore <=> set([Q]),
        Q/restriction <=> Sign/sem/content,
        DET/sem/content <=> Q.

satisfies(plain_common, Sign) :-
        Sign/syn/loc/comps <=> set([]).

satisfies(picture_common, Sign) :-
        PP abbreviates prepositional_phrase([obj]),
        Sign/syn/loc/comps <=> set([PP]).

satisfies(saturated, Sign) :-
        Sign/sem/qstore <=> set([]),
        Sign/syn/loc/spec <=> null,
        Sign/syn/loc/comps <=> null.

satisfies(proper_name, Sign) :-
        Sign/sem/content/person <=> third,
        Sign/sem/content/number <=> singular,
        Sign/syn/bind/rel/all <=> set([]),
        Sign/syn/bind/que/all <=> set([]),
        Sign/syn/bind/rel/bound <=> set([]),
        Sign/syn/bind/que/bound <=> set([]),
        Sign/syn/loc/head/form <=> norm.

satisfies(nom_pro, Sign) :-
        Sign/syn/loc/head/case <=> nom,
        Sign/syn/loc/head/form <=> norm.

satisfies(acc_pro, Sign) :-
        Sign/syn/loc/head/case <=> acc,
        Sign/syn/loc/head/form <=> norm.

satisfies(expletive, Sign) :-
        Sign/syn/bind/rel/all <=> set([]),
        Sign/syn/bind/que/all <=> set([]),
        Sign/syn/bind/rel/bound <=> set([]),
        Sign/syn/bind/que/bound <=> set([]),
```

```
        Sign/syn/loc/head/case <=> nom.

satisfies(main, Sign) :-
        Sign/syn/loc/head/inv <=> neg,
        Sign/syn/loc/head/aux <=> neg,
        Sign/syn/loc/head/form <=> bse,
        Sign/syn/loc/head/prd <=> neg,
        Sign/sem/context <=> set([P]),
        Sign/sem/parameters <=> set([]),
        P/type <=> describes,
        P/arg <=> Sign/phon,
        P/situation <=> Sign/sem/content/sit,
        Sign/sem/content/inf/polarity <=> pos.

satisfies(auxiliary, Sign) :-
        Sign/syn/loc/head/aux <=> pos,
        Sign/syn/loc/comps <=> set([XCOMP]),
        XCOMP/syn/loc/spec <=> Sign/syn/loc/spec,
        XCOMP/syn/loc/comps <=> null,
        XCOMP/syn/loc/gram_fn <=> xcomp,
        Sign/sem/context <=> set([P]),
        Sign/sem/parameters <=> set([]),
        Sign/sem/content/prop <=> XCOMP/sem/content.

satisfies(ordinary_verb, Sign) :-
        Sign/sem/content/inf/issue/agent <=>
                        Sign/syn/loc/spec/sem/content.

satisfies(strict_intransitive, Sign) :-
        Sign/syn/loc/comps <=> set([]),
        Sign/sem/content/inf/issue/theme <=> null,
        Sign/sem/content/inf/issue/patient <=> null,
        Sign/sem/content/inf/issue/goal <=> null.

satisfies(strict_transitive, Sign) :-
        NP abbreviates noun_phrase([acc, obj]),
        NP/syn/bind/slash/bound <=> set([]),
        NP/syn/bind/rel/bound <=> set([]),
        NP/syn/bind/que/bound <=> set([]),
        Sign/syn/loc/comps <=> set([NP]),
        Sign/sem/content/inf/issue/patient <=> NP/sem/content,
        Sign/sem/content/inf/issue/theme <=> null,
        Sign/sem/content/inf/issue/goal <=> null.

satisfies(strict_ditransitive, Sign) :-
        NP1 abbreviates noun_phrase([acc, obj]),
        NP1/syn/bind/slash/bound <=> set([]),
        NP1/syn/bind/rel/bound <=> set([]),
        NP1/syn/bind/que/bound <=> set([]),
        NP2 abbreviates noun_phrase([acc, i_obj]),
        NP2/syn/bind/slash/bound <=> set([]),
        NP2/syn/bind/rel/bound <=> set([]),
        NP2/syn/bind/que/bound <=> set([]),
```

```
        Sign/syn/loc/comps <=> set([NP1,NP2]),
        Sign/sem/content/inf/issue/patient <=> NP1/sem/content,
        Sign/sem/content/inf/issue/goal <=> NP2/sem/content,
        Sign/sem/content/inf/issue/theme <=> null.

satisfies(anomolous_verb, _).

satisfies(control_verb, _).

satisfies(intransitive_control, Sign) :-
        Sign/syn/loc/spec <=> X,
        VP abbreviates verb_phrase([inf, xcomp]),
        Sign/syn/loc/comps <=> set([VP]),
        VP/sem/content/inf/issue/agent <=> X/sem/content,
        Sign/sem/content/inf/issue/theme <=> VP/sem/content/inf.

satisfies(transitive_control, Sign) :-
        Sign/syn/loc/spec <=> X,
        NP abbreviates noun_phrase([acc, obj]),
        VP abbreviates verb_phrase([inf, xcomp]),
        Sign/syn/loc/comps <=> set([VP,NP]),
        Sign/sem/content/inf/issue/agent <=> X/sem/content,
        Sign/sem/content/inf/issue/theme <=> VP/sem/content/inf.

satisfies(intransitive_raising, Sign) :-
        Sign/syn/loc/comps <=> set([XCOMP]),
        Sign/syn/loc/spec <=> XCOMP/syn/loc/spec.

satisfies(intransitive_equi, Sign) :-
        NP abbreviates noun_phrase([norm]),
        Sign/syn/loc/spec <=> NP,
        Sign/sem/content/inf/issue/agent <=> NP/sem/content,
        Sign/sem/content/inf/issue/theme/issue/agent <=> NP/sem/content.

satisfies(transitive_raising, Sign) :-
        NP abbreviates noun_phrase,
        VP abbreviates verb_phrase,
        Sign/syn/loc/comps <=> set([NP,VP]),
        Sign/sem/content/inf/issue/theme/issue/agent <=> NP/sem/content.

satisfies(transitive_equi, Sign) :-
        NP1 abbreviates noun_phrase([norm]),
        NP2 abbreviates noun_phrase([norm]),
        Sign/syn/loc/spec <=> NP1,
        Sign/syn/loc/comps <=> set([NP2,_]),
        Sign/sem/content/inf/issue/patient <=> NP2/sem/content.

satisfies(subject_equi, Sign) :-
        Sign/syn/loc/spec <=> NP,
        Sign/sem/content/inf/issue/theme/issue/agent <=> NP/sem/content.

satisfies(object_equi, Sign) :-
        NP abbreviates noun_phrase,
```

```
        VP abbreviates verb_phrase,
        Sign/syn/loc/comps <=> set([NP,VP]),
        Sign/sem/content/inf/issue/theme/issue/agent <=> NP/sem/content.

satisfies(perfective, Sign) :-
        VP abbreviates verb_phrase([psp]),
        Sign/syn/loc/comps <=> set([VP]).

satisfies(copula, Sign) :-
        Sign/syn/loc/comps <=> set([XP]),
        XP/syn/loc/head/prd <=> pos,
        XP/syn/loc/comps <=> null.

satisfies(modal, Sign) :-
        VP abbreviates verb_phrase([bse]),
        Sign/syn/loc/head/form <=> fin,
        Sign/syn/loc/comps <=> set([VP]).

satisfies(infinitive, Sign) :-
        VP abbreviates verb_phrase([bse]),
        Sign/syn/loc/head/form <=> inf,
        Sign/syn/loc/comps <=> set([VP]).

/*****************************************************************/

% Leaf types:

satisfies('$a$', Sign) :-
        Sign/sem/content/quantifier <=> a,
        Sign/phon <=> list(['a']).

satisfies('$the$', Sign) :-
        Sign/sem/content/quantifier <=> the,
        Sign/phon <=> list(['the']).

satisfies('$some$', Sign) :-
        Sign/sem/content/quantifier <=> some,
        Sign/phon <=> list(['some']).

satisfies('$no$', Sign) :-
        Sign/sem/content/quantifier <=> no,
        Sign/phon <=> list(['no']).

satisfies('$most$', Sign) :-
        Sign/sem/content/quantifier <=> most,
        Sign/phon <=> list(['most']).

satisfies('$every$', Sign) :-
        Sign/sem/content/quantifier <=> every,
        Sign/phon <=> list(['every']).

satisfies('$which$', Sign) :-
        Sign/sem/content/quantifier <=> which,
```

```
        Sign/phon <=> list(['which']).

satisfies('$and$', Sign) :-
        Sign/phon <=> list(['and']).

satisfies('$both$', Sign) :-
        Sign/phon <=> list(['both']).

satisfies('$but$', Sign) :-
        Sign/phon <=> list(['but']).

satisfies('$or$', Sign) :-
        Sign/phon <=> list(['or']).

satisfies('$nor$', Sign) :-
        Sign/phon <=> list(['nor']).

satisfies('$either$', Sign) :-
        Sign/phon <=> list(['either']).

satisfies('$neither$', Sign) :-
        Sign/phon <=> list(['neither']).

satisfies('$cat$', Sign) :-
        Sign/sem/content/body/inf/issue/reln <=> cat,
        Sign/phon <=> list(['cat']).

satisfies('$dog$', Sign) :-
        Sign/sem/content/body/inf/issue/reln <=> dog,
        Sign/phon <=> list(['dog']).

satisfies('$kitten$', Sign) :-
        Sign/sem/content/body/inf/issue/reln <=> kitten,
        Sign/phon <=> list(['kitten']).

satisfies('$man$', Sign) :-
        Sign/sem/content/body/inf/issue/reln <=> man,
        Sign/phon <=> list(['man']).

satisfies('$woman$', Sign) :-
        Sign/sem/content/body/inf/issue/reln <=> woman,
        Sign/phon <=> list(['woman']).

satisfies('$picture$', Sign) :-
        Sign/sem/content/body/inf/issue/reln <=> picture_of,
        PP abbreviates prepositional_phrase([of]),
        Sign/phon <=> list(['picture']),
        Sign/syn/loc/comps <=> set([PP]).

satisfies('$Tigger$', Sign) :-
        Sign/phon <=> list(['Tigger']).

satisfies('$Tom$', Sign) :-
```

```
        Sign/phon <=> list(['Tom']).

satisfies('$Fido$', Sign) :-
        Sign/phon <=> list(['Fido']).

satisfies('$Rover$', Sign) :-
        Sign/phon <=> list(['Rover']).

satisfies('$Rome$', Sign) :-
        Sign/phon <=> list(['Rome']).

satisfies('$he$', Sign) :-
        Sign/phon <=> list(['he']),
        Sign/sem/content/gender <=> masculine,
        Sign/sem/content/person <=> third,
        Sign/sem/content/number <=> singular,
        Sign/syn/bind/rel/all <=> set([]),
        Sign/syn/bind/que/all <=> set([]),
        Sign/syn/bind/rel/bound <=> set([]),
        Sign/syn/bind/que/bound <=> set([]).

satisfies('$she$', Sign) :-
        Sign/phon <=> list(['she']),
        Sign/sem/content/gender <=> feminine,
        Sign/sem/content/person <=> third,
        Sign/sem/content/number <=> singular,
        Sign/syn/bind/rel/all <=> set([]),
        Sign/syn/bind/que/all <=> set([]),
        Sign/syn/bind/rel/bound <=> set([]),
        Sign/syn/bind/que/bound <=> set([]).

satisfies('$who$', Sign) :-
        NP abbreviates noun_phrase([nom,norm]),
        Sign/phon <=> list(['who']),
        Sign/sem/content/person <=> third,
        Sign/sem/content/number <=> singular,
        Sign/syn/bind/rel/all <=> set([NP]),
        Sign/syn/bind/que/all <=> set([]),
        Sign/syn/bind/rel/bound <=> set([]).

satisfies('$him$', Sign) :-
        Sign/phon <=> list(['him']),
        Sign/sem/content/gender <=> masculine,
        Sign/sem/content/person <=> third,
        Sign/sem/content/number <=> singular,
        Sign/syn/bind/rel/all <=> set([]),
        Sign/syn/bind/que/all <=> set([]),
        Sign/syn/bind/rel/bound <=> set([]),
        Sign/syn/bind/que/bound <=> set([]).

satisfies('$her$', Sign) :-
        Sign/phon <=> list(['her']),
        Sign/sem/content/gender <=> feminine,
```

```
        Sign/sem/content/person <=> third,
        Sign/sem/content/number <=> singular,
        Sign/syn/bind/rel/all <=> set([]),
        Sign/syn/bind/que/all <=> set([]),
        Sign/syn/bind/rel/bound <=> set([]),
        Sign/syn/bind/que/bound <=> set([]).

satisfies('$whom$', Sign) :-
        NP abbreviates noun_phrase([acc,norm]),
        Sign/phon <=> list(['whom']),
        Sign/sem/content/person <=> third,
        Sign/sem/content/number <=> singular,
        Sign/syn/bind/rel/all <=> set([NP]),
        Sign/syn/bind/que/all <=> set([]),
        Sign/syn/bind/rel/bound <=> set([]).

satisfies('$it$', Sign) :-
        Sign/phon <=> list(['it']),
        Sign/syn/loc/head/form <=> 'it'.

satisfies('$there$', Sign) :-
        Sign/phon <=> list(['there']),
        Sign/syn/loc/head/form <=> 'there'.

satisfies('$small$', Sign) :-
        Sign/phon <=> list(['small']).

satisfies('$black$', Sign) :-
        Sign/phon <=> list(['black']).

satisfies('$furry$', Sign) :-
        Sign/phon <=> list(['furry']).

satisfies('$easy$', Sign) :-
        Sign/phon <=> list(['easy']).

satisfies('$tough$', Sign) :-
        Sign/phon <=> list(['tough']).

satisfies('$quickly$', Sign) :-
        Sign/phon <=> list(['quickly']).

satisfies('$loudly$', Sign) :-
        Sign/phon <=> list(['loudly']).

satisfies('$on$', Sign) :-
        Sign/phon <=> list(['on']),
        Sign/syn/loc/head/form <=> on.

satisfies('$in$', Sign) :-
        Sign/phon <=> list(['in']),
        Sign/syn/loc/head/form <=> in.
```

```
satisfies('$to$', Sign) :-
        Sign/phon <=> list(['to']),
        Sign/syn/loc/head/form <=> to.

satisfies('$of$', Sign) :-
        Sign/phon <=> list(['of']),
        Sign/syn/loc/head/form <=> of.

satisfies('$by$', Sign) :-
        Sign/phon <=> list(['by']),
        Sign/syn/loc/head/form <=> by.

satisfies('$from$', Sign) :-
        Sign/phon <=> list(['from']),
        Sign/syn/loc/head/form <=> from.

satisfies('$with$', Sign) :-
        Sign/phon <=> list(['with']),
        Sign/syn/loc/head/form <=> with.

satisfies('$about$', Sign) :-
        Sign/phon <=> list(['about']),
        Sign/syn/loc/head/form <=> about.

satisfies('$run$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> run,
        Sign/phon <=> list(['run']),
        Sign/syn/loc/spec/syn/loc/head/form <=> norm.

satisfies('$talk$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> talk,
        Sign/phon <=> list(['talk']),
        Sign/syn/loc/spec/syn/loc/head/form <=> norm.

satisfies('$miaow$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> miaow,
        Sign/phon <=> list(['miaow']),
        Sign/syn/loc/spec/syn/loc/head/form <=> norm.

satisfies('$bounce$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> bounce,
        Sign/phon <=> list(['bounce']),
        Sign/syn/loc/spec/syn/loc/head/form <=> norm.

satisfies('$rain$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> rain,
        Sign/phon <=> list(['rain']),
        Sign/syn/loc/spec/syn/loc/head/form <=> it.

satisfies('$love$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> love,
        Sign/phon <=> list(['love']).
```

```
satisfies('$kiss$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> kiss,
        Sign/phon <=> list(['kiss']).

satisfies('$chase$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> chase,
        Sign/phon <=> list(['chase']).

satisfies('$give$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> give,
        Sign/phon <=> list(['give']).

satisfies('$sell$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> sell,
        Sign/phon <=> list(['sell']).

satisfies('$argue$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> argue,
        PPwith abbreviates prepositional_phrase([with, obj]),
        PPabout abbreviates prepositional_phrase([about, obj]),
        Sign/phon <=> list(['argue']),
        Sign/syn/loc/comps <=> set([PPwith, PPabout]).

satisfies('$$believe$$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> believe,
        S/syn/loc/head/maj <=> verb,
        S/syn/loc/head/form <=> fin,
        S/syn/loc/spec <=> null,
        S/syn/loc/comps <=> null,
        Sign/phon <=> list(['believe']),
        Sign/syn/loc/comps <=> set([S]).

satisfies('$appear$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> appear,
        Sign/phon <=> list(['appear']).

satisfies('$seem$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> seem,
        Sign/phon <=> list(['seem']).

satisfies('$try$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> try,
        Sign/phon <=> list(['try']).

satisfies('$prefer$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> prefer,
        Sign/phon <=> list(['prefer']).

satisfies('$pretend$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> pretend,
        Sign/phon <=> list(['pretend']).

satisfies('$hope$', Sign) :-
```

```
        Sign/sem/content/inf/issue/reln <=> hope,
        Sign/phon <=> list(['hope']).

satisfies('$know$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> know,
        Sign/phon <=> list(['know']).

satisfies('$believe$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> believe,
        Sign/phon <=> list(['believe']).

satisfies('$persuade$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> persuade,
        Sign/phon <=> list(['persuade']).

satisfies('$promise$', Sign) :-
        Sign/sem/content/inf/issue/reln <=> promise,
        Sign/phon <=> list(['promise']).

satisfies('$have$', Sign) :-
        Sign/phon <=> list(['have']),
        Sign/syn/loc/head/form <=> bse.

satisfies('$be$', Sign) :-
        Sign/phon <=> list(['be']),
        Sign/syn/loc/head/form <=> bse.

satisfies('$been$', Sign) :-
        Sign/phon <=> list(['been']),
        Sign/syn/loc/head/form <=> psp.

satisfies('$can$', Sign) :-
        Sign/phon <=> list(['can']).

satisfies('$could$', Sign) :-
        Sign/phon <=> list(['could']).

satisfies('$will$', Sign) :-
        Sign/phon <=> list(['will']).

satisfies('$would$', Sign) :-
        Sign/phon <=> list(['would']).

satisfies('$should$', Sign) :-
        Sign/phon <=> list(['should']).

satisfies('$may$', Sign) :-
        Sign/phon <=> list(['may']).

satisfies('$might$', Sign) :-
        Sign/phon <=> list(['might']).

satisfies('$$to$$', Sign) :-
```

```
        Sign/phon <=> list(['to']).

/*************************************************************************/
```

4 phrasal_hierarchy.pl

```
/*************************************************************************

        File:               phrasal_hierarchy.pl
        Author:             Richard Cooper
        Date:               24/10/89

This file contains the phrasal hierachy, expressed in terms of subtype/2
clauses, specifying the shape of the hierarchy and satisfies/2 clauses,
specifying the properties associated with each type.  The satisfies/2
clauses are embedded in data/1 clauses as the act as data for the
partial execution routine.

*************************************************************************/
/*************************************************************************

                        subtype(Type, SubType).

*************************************************************************/

subtype(phrasal_constituent, headed_phrase).

subtype(headed_phrase, head_adjunct_phrase).
subtype(headed_phrase, head_conjunction_phrase).
subtype(headed_phrase, head_argument_phrase).

subtype(head_adjunct_phrase, head_left_adjunct_phrase).
subtype(head_adjunct_phrase, head_right_adjunct_phrase).

subtype(head_argument_phrase, head_complement_phrase).
subtype(head_argument_phrase, head_topic_phrase).

subtype(head_topic_phrase, head_specifier_phrase).
subtype(head_topic_phrase, head_filler_phrase).

subtype(phrasal_constituent, coordinate_phrase).

subtype(coordinate_phrase, binary_coordinate_phrase).
subtype(coordinate_phrase, iterated_coordinate_phrase).

/*************************************************************************

                        satisfies(Type, Sign).

*************************************************************************/
```

```
data(( satisfies(phrasal_constituent, Sign) :-
          unify_feature_structures(Sign, Temp),
          Temp/phon <=> list([_|_]),
          Temp/dtrs/head_dtr <=> _,
          Temp/dtrs/topic_dtr <=> _,
          Temp/dtrs/comp_dtrs <=> _,
          Temp/dtrs/adjunct_dtr <=> _,
          Temp/dtrs/coord_dtrs <=> _ )).

data(( satisfies(headed_phrase, Sign) :-
          unify_feature_structures(Sign, Temp),
          Temp/dtrs/head_dtr/phon <=> list([_|_]),
          Temp/dtrs/head_dtr/syn/loc/gram_fn <=> head,
          Temp/dtrs/head_dtr/syn/conj <=> 'NIL',
          Temp/dtrs/head_dtr/syn/loc/head <=> Temp/syn/loc/head,
          Temp/dtrs/coord_dtrs <=> null )).

data(( satisfies(head_adjunct_phrase, Sign) :-
          unify_feature_structures(Sign, Temp),
          Temp/syn/conj <=> 'NIL',
          Temp/dtrs/topic_dtr <=> null,
          Temp/dtrs/comp_dtrs <=> null,
          Temp/dtrs/marker_dtr <=> null,
          Temp/dtrs/head_dtr <=> HeadDtr,
          Temp/syn/loc/spec <=> HeadDtr/syn/loc/spec,
          Temp/syn/loc/comps <=> HeadDtr/syn/loc/comps,
          Temp/dtrs/adjunct_dtr <=> AdjunctDtr,
          Temp/sem/content <=> AdjunctDtr/sem/content,
          AdjunctDtr/syn/loc/spec <=> null,
          AdjunctDtr/syn/loc/comps <=> null,
          AdjunctDtr/syn/bind/slash/bound <=> set([]),
          AdjunctDtr/syn/bind/rel/bound <=> set([]),
          AdjunctDtr/syn/bind/que/bound <=> set([]) )).

data(( satisfies(head_left_adjunct_phrase, Sign) :-
          unify_feature_structures(Sign, Temp),
          Temp/dtrs/head_dtr <=> HeadDtr,
          Temp/dtrs/adjunct_dtr <=> AdjunctDtr,
          AdjunctDtr/syn/loc/head/left_adjoins <=> set(PossibleHeads),
          AdjunctDtr/syn/loc/gram_fn <=> pre_modifier,
          Temp/phon <=> Phrase,
          HeadDtr/phon <=> Head,
          AdjunctDtr/phon <=> Adjunct,
          non_empty_concat(Adjunct, Head, Phrase),
          constituent(AdjunctDtr),
          member(HeadDtr, PossibleHeads),
          constituent(HeadDtr),
          semantic_inheritance(Temp),
          binding_inheritance(Temp) )).

data(( satisfies(head_right_adjunct_phrase, Sign) :-
          unify_feature_structures(Sign, Temp),
```

```
            Temp/dtrs/head_dtr <=> HeadDtr,
            Temp/dtrs/adjunct_dtr <=> AdjunctDtr,
            AdjunctDtr/syn/loc/head/right_adjoins <=> set(PossibleHeads),
            AdjunctDtr/syn/loc/gram_fn <=> post_modifier,
            Temp/phon <=> Phrase,
            HeadDtr/phon <=> Head,
            AdjunctDtr/phon <=> Adjunct,
            non_empty_concat(Head, Adjunct, Phrase),
            constituent(AdjunctDtr),
            member(HeadDtr, PossibleHeads),
            constituent(HeadDtr),
            semantic_inheritance(Temp),
            binding_inheritance(Temp) )).

data(( satisfies(head_conjunction_phrase, Sign) :-
            unify_feature_structures(Sign, Temp),
            Temp/phon <=> list([Conj | Head]),
            Temp/dtrs/head_dtr <=> HeadDtr,
            Temp/dtrs/topic_dtr <=> null,
            Temp/dtrs/comp_dtrs <=> null,
            Temp/dtrs/adjunct_dtr <=> null,
            Temp/dtrs/marker_dtr <=> ConjDtr,
            Temp/sem <=> HeadDtr/sem,
            HeadDtr/phon <=> list(Head),
            HeadDtr/syn/loc <=> Temp/syn/loc,
            HeadDtr/syn/bind <=> Temp/syn/bind,
            HeadDtr/syn/conj <=> 'NIL',
            ConjDtr/phon <=> list([Conj]),
            ConjDtr/syn/min <=> conjunction,
            ConjDtr/syn/conj_type <=> Temp/syn/conj,
            ConjDtr/dtrs <=> null,
            constituent(ConjDtr),
            constituent(HeadDtr) )).

data(( satisfies(head_argument_phrase, Sign) :-
            unify_feature_structures(Sign, Temp),
            Temp/syn/conj <=> 'NIL',
            Temp/dtrs/marker_dtr <=> null,
            Temp/dtrs/adjunct_dtr <=> null )).

data(( satisfies(head_complement_phrase, Sign) :-
            unify_feature_structures(Sign, Temp),
            Temp/dtrs/head_dtr <=> HeadDtr,
            Temp/dtrs/comp_dtrs <=> CompDtrs,
            Temp/phon <=> Phrase,
            HeadDtr/phon <=> Head,
            HeadDtr/syn/loc/comps <=> CompDtrs,
            HeadDtr/syn/bind/slash/bound <=> set([]),
            HeadDtr/syn/bind/que/bound <=> set([]),
            HeadDtr/syn/bind/rel/bound <=> set([]),
            HeadDtr/dtrs <=> null,
            Temp/syn/loc/spec <=> HeadDtr/syn/loc/spec,
            Temp/syn/loc/comps <=> null,
```

```
            Temp/dtrs/topic_dtr <=> null,
            Temp/sem/content <=> Temp/dtrs/head_dtr/sem/content,
            CompDtrs <=> set(_),
            concat(Head, Comps, Phrase),
            constituent(HeadDtr),
            order_phonology(Comps, CompDtrs),
            set_of_constituents(CompDtrs),
            semantic_inheritance(Temp),
            binding_inheritance(Temp) )).

data(( satisfies(head_topic_phrase, Sign) :-
            unify_feature_structures(Sign, Temp),
            Temp/phon <=> Phrase,
            Temp/syn/loc/spec <=> null,
            Temp/syn/loc/comps <=> null,
            Temp/dtrs/comp_dtrs <=> null,
            Temp/dtrs/head_dtr/phon <=> Head,
            Temp/dtrs/head_dtr/syn/bind/que/bound <=> set([]),
            Temp/dtrs/head_dtr/syn/bind/rel/bound <=> set([]),
            Temp/dtrs/topic_dtr/phon <=> Topic,
            Temp/dtrs/topic_dtr/syn/conj <=> 'NIL',
            Temp/dtrs/head_dtr/syn/loc/comps <=> null,
            non_empty_concat(Topic, Head, Phrase) )).

data(( satisfies(head_specifier_phrase, Sign) :-
            unify_feature_structures(Sign, Temp),
            Temp/dtrs/head_dtr <=> HeadDtr,
            Temp/dtrs/topic_dtr <=> SpecDtr,
            HeadDtr/syn/loc/spec <=> SpecDtr,
            constituent(HeadDtr),
            constituent(SpecDtr),
            build_spec_semantics(Temp),
            HeadDtr/syn/bind/slash/bound <=> set([]),
            semantic_inheritance(Temp),
            binding_inheritance(Temp) )).

data(( satisfies(head_filler_phrase, Sign) :-
            unify_feature_structures(Sign, Temp),
            Temp/dtrs/head_dtr <=> HeadDtr,
            Temp/sem/content <=> Temp/dtrs/head_dtr/sem/content,
            Temp/dtrs/topic_dtr <=> FillerDtr,
            HeadDtr/syn/loc/spec <=> null,
            FillerDtr/syn/bind/slash/all <=> set([]),
            HeadDtr/syn/bind/slash/bound <=> set([FillerDtr]),
            constituent(HeadDtr),
            constituent(FillerDtr),
            semantic_inheritance(Temp),
            binding_inheritance(Temp) )).

data(( satisfies(coordinate_phrase, Sign) :- fail )).
% Not implemented

data(( satisfies(binary_coordinate_phrase, Sign) :- fail )).
```

```
% Not implemented

data(( satisfies(iterated_coordinate_phrase, Sign) :- fail )).
% Not implemented

/*************************************************************************/
```

5 full_hierarchy.pl

```
/*************************************************************************

        File:               full_hierarchy.pl
        Author:             Richard Cooper
        Date:               26/10/89 (Revised for SICStus 23/04/90)

This file loads the entire constituent hierarchy, comprising of the
lexical hiearchy (in uncompiled form), the phrasal hierarchy, and the
node that connects them.  The file is only used in the
lexicon-compilation process.

*************************************************************************/

:- dynamic satisfies/2.

:- multifile subtype/2, satisfies/2.

satisfies(constituent, _).

subtype(constituent, lexical_constituent).

:- [lexical_hierarchy].

subtype(constituent, phrasal_constituent).

:- [phrasal_hierarchy].

/*************************************************************************/

classify(Type, Sign) :-
        satisfies(Type, Sign),
        ((subtype(Type, SubType), classify(SubType, Sign));
         \+ subtype(Type, _)).

/*************************************************************************/
```

6 compiled_hierarchy.pl

```
/*************************************************************************
```

```
        File:                   compiled_hierarchy.pl
        Author:                 Richard Cooper
        Date:                   26/10/89 (Revised 23/04/90 for SICStus)

This file loads the phrasal hierarchy, the compiled lexical hierarchy,
and the node that connects them.

*************************************************************************/

:- multifile subtype/2, satisfies/2.

:- dynamic satisfies/2.

subtype(constituent, lexical_constituent).

satisfies(lexical_constituent, Sign) :-
        Temp/phon <=> Sign/phon,
        lexical_entry(Temp),
        unify_feature_structures(Temp, Sign).

:- [lexicon].

subtype(constituent, phrasal_constituent).

:- [phrasal_hierarchy].

/************************************************************************/

satisfies(constituent, Sign).

/************************************************************************/

classify(Type, Sign) :-
        satisfies(Type, Sign),
        ((subtype(Type, SubType), classify(SubType, Sign));
         \+subtype(Type,_)).

/************************************************************************/
```

7 lexical_rules.pl

```
/*************************************************************************

        File:                   lexical_rules.pl
        Author:                 Richard Cooper
        Date:                   24/04/90

*************************************************************************/
```

```
% Load clauses for determining inflectional forms:

:- compile([inflections]).

/*********************************************************************/

% For each lexical entry, all inflectional rules are first applied, then
% all non-infelctional (slash termination) rules are applied to the
% expanded lexicon.

apply_lexical_rules :-
        lexical_entry(L),
        apply_inflectional_rule(L),
        fail.
apply_lexical_rules :-
        lexical_entry(L),
        apply_non_inflectional_rule(L),
        fail.
apply_lexical_rules.

/*********************************************************************/

% PNR: lexical rule for plurals of nouns (both with and without
% determiners: it generates two new entries from each suitable noun)

data((    apply_inflectional_rule(L) :-
                L/syn/loc/head/maj <=> noun,
                L/syn/loc/spec <=> (_,_),
                L/phon <=> Phon,
                plural_inflection(Phon, Phons),
                M/phon <=> Phons,
                M <<= L,
                assertz(lexical_entry(M)),
                N/phon <=> Phons,
                N/syn/loc/spec <=> null,
                N <<= L,
                assertz(lexical_entry(N)),
                L/phon <=> list(X),
                format("PNR applied to ~w~n", [X]), ttyflush )).

/*********************************************************************/

% VFR: lexical rule for fin and prp forms of bse form verbs.

data((    apply_inflectional_rule(L) :-
                L/syn/loc/head/maj <=> verb,
                L/syn/loc/head/form <=> bse,
                L/phon <=> Phon,
                finite_inflection(Phon, PastPhon),
                M/phon <=> PastPhon,
                M/syn/loc/head/form <=> fin,
                M <<= L,
                assertz(lexical_entry(M)),
```

```
                present_part_inflection(Phon, PRPPhon),
                N/phon <=> PRPPhon,
                N/syn/loc/head/form <=> prp,
                N/syn/loc/head/prd <=> pos,
                N <<= L,
                assertz(lexical_entry(N)),
                L/phon <=> list(X),
                format("VFR applied to ~w~n", [X]), ttyflush )).

/*****************************************************************/

% STR1: lexical rule for slash termination: generates new lexical
% entries with depleted comp sets.

data((    apply_non_inflectional_rule(L) :-
                L/syn/loc/head/maj <=> verb,
                L/syn/bind/slash/all <=> set([]),
                L/syn/loc/comps <=> set(LComps),
                delete(Gap, LComps, MComps),
                M/syn/loc/comps <=> set(MComps),
                M/syn/bind/slash/all <=> set([Gap]),
                M <<= L,
                assertz(lexical_entry(M)),
                L/phon <=> list(X),
                format("STR1 applied to ~w~n", [X]), ttyflush )).

/*****************************************************************/

% STR2: lexical rule for slash termination: generates new lexical
% entries taking VP complements instead of S complements.

data((    apply_non_inflectional_rule(L) :-
                L/syn/loc/head/maj <=> verb,
                L/syn/bind/slash/all <=> set([]),
                L/syn/loc/comps <=> set(LComps),
                VP abbreviates verb_phrase,
                SComp/syn/loc/head/maj <=> verb,
                SComp/syn/loc/spec <=> null,
                SComp/syn/loc/comps <=> null,
                delete(SComp, LComps, TempComps),
                VP <<= SComp,
                VP/syn/loc/spec <=> NP,
                M/syn/loc/comps <=> set([VP|TempComps]),
                M/syn/bind/slash/all <=> set([NP]),
                M <<= L,
                assertz(lexical_entry(M)),
                L/phon <=> list(X),
                format("STR2 applied to ~w~n", [X]), ttyflush )).

/*****************************************************************/

% SAIR: lexical rule for subject-auxiliary inversion: generates new
% "inverted" lexical entries for all auxiliaries.
```

```
data((    apply_non_inflectional_rule(L) :-
              L/syn/loc/head/maj <=> verb,
              L/syn/loc/head/form <=> fin,
              L/syn/loc/head/aux <=> pos,
              L/syn/loc/head/inv <=> neg,
              L/syn/loc/spec <=> NP,
              L/syn/loc/comps <=> set([VP]),
              M/syn/loc/head/inv <=> pos,
              M/syn/loc/spec <=> null,
              M/syn/loc/comps <=> set([NP,VP]),
              M <<= L,
              assertz(lexical_entry(M)),
              L/phon <=> list(X),
              format("SAIR applied to ~w~n", [X]), ttyflush )).

/*****************************************************************/
```

8 inflections.pl

```
/*****************************************************************

        File:               inflections.pl
        Author:             Richard Cooper
        Date:               26/04/90

This file contains very dumb clauses for construting the appropriate
inflections.  It might easily be replaced with some more intelligent
clauses which actually do morphological analysis.  All clauses are
called only from routines in the file lexical_rules.pl.

*****************************************************************/

plural_inflection(list([X]), list([Y])) :-
        convert_to_plural(X, Y).

convert_to_plural(picture, pictures).
convert_to_plural(woman, women).
convert_to_plural(man, men).
convert_to_plural(kitten, kittens).
convert_to_plural(dog, dogs).
convert_to_plural(cat, cats).

/*****************************************************************/

finite_inflection(list([X]), list([Y])) :-
        convert_to_finite(X, Y).

convert_to_finite(run, runs).
convert_to_finite(talk, talks).
```

```
convert_to_finite(miaow, miaows).
convert_to_finite(bounce, bounces).
convert_to_finite(rain, rains).
convert_to_finite(love, loves).
convert_to_finite(kiss, kisses).
convert_to_finite(chase, chases).
convert_to_finite(give, gives).
convert_to_finite(sell, sells).
convert_to_finite(argue, argues).
convert_to_finite(appear, appears).
convert_to_finite(seem, seems).
convert_to_finite(try, tries).
convert_to_finite(prefer, prefers).
convert_to_finite(pretend, pretends).
convert_to_finite(hope, hopes).
convert_to_finite(know, knows).
convert_to_finite(believe, believes).
convert_to_finite(persuade, persuades).
convert_to_finite(promise, promises).
convert_to_finite(have, has).
convert_to_finite(be, is).

/*****************************************************************/

present_part_inflection(list([X]), list([Y])) :-
        convert_to_present_part(X, Y).

convert_to_present_part(run, runing).
convert_to_present_part(talk, talking).
convert_to_present_part(miaow, miaowing).
convert_to_present_part(bounce, bouncing).
convert_to_present_part(rain, raining).
convert_to_present_part(love, loving).
convert_to_present_part(kiss, kissing).
convert_to_present_part(chase, chasing).
convert_to_present_part(give, giving).
convert_to_present_part(sell, selling).
convert_to_present_part(argue, arguing).
convert_to_present_part(appear, appearing).
convert_to_present_part(seem, seeming).
convert_to_present_part(try, trying).
convert_to_present_part(prefer, prefering).
convert_to_present_part(pretend, pretending).
convert_to_present_part(hope, hoping).
convert_to_present_part(know, knowing).
convert_to_present_part(believe, believing).
convert_to_present_part(persuade, persuading).
convert_to_present_part(promise, promising).
convert_to_present_part(have, having).
convert_to_present_part(be, being).

/*****************************************************************/
```

9 abbreviations.pl

```
/*************************************************************************

        File:               abbreviations.pl
        Author:             Richard Cooper
        Date:               26/10/89

This file contains clauses for CPSG abbreviations.

*************************************************************************/

Sign abbreviates noun_phrase :-
        Sign/syn/conj <=> NIL,
        Sign/syn/loc/head/maj <=> noun,
        Sign/syn/loc/head/left_adjoins <=> set([]),
        Sign/syn/loc/head/right_adjoins <=> set([]),
        Sign/syn/loc/spec <=> null,
        Sign/syn/loc/comps <=> null, !.

Sign abbreviates noun_phrase([]) :-
        Sign abbreviates noun_phrase, !.

Sign abbreviates noun_phrase([Case|T]) :-
        case(Case),
        Sign abbreviates noun_phrase(T),
        Sign/syn/loc/head/case <=> Case, !.

Sign abbreviates noun_phrase([Nform|T]) :-
        noun_form(Nform),
        Sign abbreviates noun_phrase(T),
        Sign/syn/loc/head/form <=> Nform, !.

Sign abbreviates noun_phrase([GF|T]) :-
        grammatical_function(GF),
        Sign abbreviates noun_phrase(T),
        Sign/syn/loc/gram_fn <=> GF, !.

Sign abbreviates determiner :-
        Sign/syn/conj <=> NIL,
        Sign/syn/loc/head/maj <=> determiner,
        Sign/syn/loc/head/left_adjoins <=> set([]),
        Sign/syn/loc/head/right_adjoins <=> set([]),
        Sign/syn/loc/spec <=> null,
        Sign/syn/loc/comps <=> null,
        Sign/syn/bind/slash/all <=> set([]),
        Sign/syn/bind/rel/all <=> set([]),
        Sign/syn/bind/slash/bound <=> set([]),
        Sign/syn/bind/rel/bound <=> set([]), !.

Sign abbreviates verb_phrase :-
        Sign/syn/conj <=> NIL,
```

```
        Sign/syn/loc/head/maj <=> verb,
        Sign/syn/loc/head/left_adjoins <=> set([]),
        Sign/syn/loc/head/right_adjoins <=> set([]),
        Sign/syn/loc/comps <=> null,
        Sign/syn/loc/spec <=> NP,
        NP abbreviates noun_phrase, !.

Sign abbreviates verb_phrase([]) :-
        Sign abbreviates verb_phrase.

Sign abbreviates verb_phrase([Vform|T]) :-
        verb_form(Vform),
        Sign abbreviates verb_phrase(T),
        Sign/syn/loc/head/form <=> Vform, !.

Sign abbreviates verb_phrase([GF|T]) :-
        grammatical_function(GF),
        Sign abbreviates verb_phrase(T),
        Sign/syn/loc/gram_fn <=> GF, !.

Sign abbreviates prepositional_phrase :-
        Sign/syn/conj <=> NIL,
        Sign/syn/loc/head/maj <=> preposition,
        Sign/syn/loc/spec <=> null,
        Sign/syn/loc/comps <=> null, !.

Sign abbreviates prepositional_phrase([]) :-
        Sign abbreviates prepositional_phrase, !.

Sign abbreviates prepositional_phrase([Pform|T]) :-
        preposition_form(Pform),
        Sign abbreviates prepositional_phrase(T),
        Sign/syn/loc/head/form <=> Pform, !.

Sign abbreviates prepositional_phrase([GF|T]) :-
        grammatical_function(GF),
        Sign abbreviates prepositional_phrase(T),
        Sign/syn/loc/gram_fn <=> GF, !.

/*****************************************************************/

case(nom).
case(acc).

/*****************************************************************/

noun_form(norm).
noun_form(it).
noun_form(there).

/*****************************************************************/

verb_form(fin).
```

```
verb_form(bse).
verb_form(psp).
verb_form(prp).
verb_form(pas).
verb_form(inf).
verb_form(ger).

/*************************************************************************/

preposition_form(to).
preposition_form(of).
preposition_form(on).
preposition_form(in).
preposition_form(by).
preposition_form(from).
preposition_form(with).
preposition_form(about).

/*************************************************************************/

grammatical_function(head).
grammatical_function(spec).
grammatical_function(subj).
grammatical_function(obj).
grammatical_function(i_obj).
grammatical_function(xcomp).
grammatical_function(pre_modifier).
grammatical_function(post_modifier).

/*************************************************************************/
```

10 attributes.pl

```
/*************************************************************************

        File:               attributes.pl
        Author:             Richard Cooper
        Date:               10/08/90

This file contains avm frames which act as data for compile_value/0.
Each avm_frame/2 clause specifies a frame for a feature structure.  They
are thus really type definitions.  The first argument is the name of the
type and the second is the (ordered) list of attributes defined for that
type.

*************************************************************************/

avm_frame(constituent, [phon, syn, sem, dtrs]).
avm_frame(syntactic, [loc, bind, conj]).
avm_frame(binding, [slash, rel, que]).
```

```
avm_frame(srq, [bound, all]).
avm_frame(daughters, [head_dtr, topic_dtr, comp_dtrs,
                      adjunct_dtr, marker_dtr, coord_dtrs]).
avm_frame(local, [head, gram_fn, spec, comps]).
avm_frame(minor, [min, conj_type]).
avm_frame(head, [maj, case, form, aux, inv,
                 prd, left_adjoins, right_adjoins]).
avm_frame(semantic, [content, context, parameters, qstore, cstore]).
avm_frame(agr, [number, person, gender]).
avm_frame(infon, [issue, polarity]).
avm_frame(supports, [sit, inf]).
avm_frame(aspectual_type, [aspect, prop, location]).
avm_frame(sit_type, [type, arg, situation]).
avm_frame(arg_roles, [reln, agent, patient, goal, theme]).
avm_frame(quantifier, [quantifier, tag, restriction]).
avm_frame(complex_type, [body, arg_roles, params]).

/*****************************************************************/
```

11 bottom_up.pl

```
/*****************************************************************

        File:               bottom_up.pl
        Author:             Richard Cooper
        Date:               12/05/90

This file contains the clauses for a bottom-up control strategy.
Directives load all files on which this is dependent.  Thus the file can
be loaded directly into Prolog.  The final directive saves the state
(for next time, so you don't have to recompile it) and then sets the
thing running.

*****************************************************************/

:- dynamic chart/2.

:- compile([foundations,
            relational_dependencies,
            value,
            partial_execute,
            print,
            bottom_up_front_end]),
   ttyflush.

:- consult([compiled_hierarchy]), ttyflush.

:- partially_execute, ttyflush.

/*****************************************************************/
```

```
% The definition of the relational dependency constituent/1 pretty
% much dictates the overall control strategy.  For bottom-up parsing a
% chart of well-formed substrings is kept, so that checking that a
% feature structure describes a constituent is a matter of table
% look-up:

constituent(Sign) :-
        Temp/phon <=> Sign/phon,
        chart(Temp, _),
        unify_feature_structures(Temp, Sign).

/*************************************************************************/

:- mode parse.

% Parsing with the bottom-up strategy involves parsing all substrings:

parse :-
        get_partial_description(Sign), statistics(runtime, _),
        retractall(chart(_, _)),
        Sign/phon <=> list(String), !,
        parse_all_substrings(String),
        statistics(runtime, [_, Time]),
        count_parses(Sign, Total),
        format("Finished parsing in ~3d seconds.~n", [Time]),
        format("Total number of successful parses: ~d~2n", [Total]),
        ttyflush.

/*************************************************************************/

% Via a failure driven loop, parse all substrings parses all
% substrings of a given string, including the string itself, in a highly
% specific order.  All that is really necessary is that no string is
% parsed before all of its substrings have been parsed.  We do this in a
% left-to-right fashion, taking a (not necessarily proper) prefix of the
% string, assuming all substrings of its proper prefixes to be parsed
% and then parsing all (not necessarily proper) suffixes of that prefix,
% in order of increasing length.  This works because prefix/2, on
% backtracking, produces all prefixes of a string in the order from
% smallest to largest.

parse_all_substrings(String) :-
        prefix(String, P),
        parse_prefix(P),
        fail.
parse_all_substrings(String).

/*************************************************************************/

parse_prefix([H]) :-
        parse_string([H]), !.
parse_prefix([H|T]) :-
```

```
        parse_prefix(T),
        parse_string([H|T]), !.

/*************************************************************************/

% Parsing a string then, given that all its substrings have been parsed,
% is just a matter of classifying that string as a constituent.
% Successful parses are recorded before backtracking is forced to ensure
% that all parses are found.

parse_string(String) :-
        Sign/phon <=> list(String),
        classify(constituent, Sign),
        record_and_fail(Sign).
parse_string(String).

/*************************************************************************/

% Successful parses are recorded, along with an index, so that they can
% be refered to by other routines, most especially the various output
% routines.

record_and_fail(Sign) :-
        chart(Sign, _), !, fail.
record_and_fail(Sign) :-
        chart(_, PrevRef),
        Ref is PrevRef+1,
        asserta(chart(Sign, Ref)), !, fail.
record_and_fail(Sign) :-
        asserta(chart(Sign, 0)), !, fail.

/*************************************************************************/

count_parses(Sign, 0) :-
        \+ chart(Sign, _).
count_parses(Sign, Total) :-
        findall(N, chart(Sign, N), AllParses),
        length(AllParses, Total).

/*************************************************************************/

:- save(cpsg), go.

/*************************************************************************/
```

12 top_down.pl

```
/*************************************************************************

        File:                   top_down.pl
```

```
        Author:             Richard Cooper
        Date:               26/10/89 (revised for SICStus 23/04/90)

This is a top down parser that works by traversing the hierarchy,
attempting to classify a constituent by its feature structure
description, which is a feature structure whose phonology is the
original string, as a constituent.  The final directive saves the state
(for next time, so you don't have to recompile it) and then sets the
thing running.

Note the definition of constituent/1 in terms of classify/2.  It is
this which makes this top down, rather than bottom up.  A chart is
still used for compatability with the bottom-up strategy.  The chart
only ever has at most one entry on it.

**********************************************************************/

:- dynamic chart/2.

:- compile([foundations,
            relational_dependencies,
            value,
            print,
            partial_execute,
            top_down_front_end]),
   ttyflush.

:- consult([compiled_hierarchy]), ttyflush.

/**********************************************************************/

:- partially_execute.

/**********************************************************************/

:- mode constituent(?).

% Arg1 will normally be a partly instantiated possible feature
% structure description.  It is a constituent iff it can be classified
% as such.  The act of classification will normally further instantiate
% Arg1.

constituent(Sign) :-
        classify(constituent, Sign).

/**********************************************************************/

:- mode parse.

parse :-
        get_partial_description(Sign), statistics(runtime, _),
        retractall(chart(_, _)),
        constituent(Sign), statistics(runtime, [_, Time]),
```

```
        asserta(chart(Sign, 0)),
        format("Constituent parsed in ~3d seconds.~2n", [Time]),
        ttyflush.
parse :-
        format("No parses found.~2n", []), ttyflush.

/*****************************************************************/

:- save(cpsg), go.

/*****************************************************************/
```

13 compile.pl

```
/*****************************************************************

        File:               compile.pl
        Author:             Richard Cooper
        Date:               26/10/89

This file should be consulted every time the hierarchies or attributes
are altered.  It recompiles the lexicon and the value/2 clauses (which
form the basis of the term unification algorithm).

*****************************************************************/

:- compile([foundations]), ttyflush.

/*****************************************************************/

% compile_value/0 causes value/2 clauses to be asserted which will
% evaluate paths.  Backtracking (forced by failure in compile_value/1)
% ensures all members of all attribute lists to be considered.

compile_value :-
        format("Compiling value/2 clauses ...", []), ttyflush,
        avm_frame(Sort, Frame),
        compile_value(Sort, Frame).
compile_value :-
        asserta((value(X, X) :- \+ path(X), !)),
        format(" Finished compiling value/2 clauses.~n", []), ttyflush.

compile_value(Sort, Frame) :-
        member(Att, Frame),
        build_frame(Frame, Att, X, NewFrame),
        assertz((value(Z/Att, X) :- value(Z, (Sort, NewFrame)))),
        fail.

/*****************************************************************/
```

```
build_frame([], _, _, []) :- !.
build_frame([Att|T], Att, X, [X|NT]) :- !,
        build_frame(T, Att, X, NT).
build_frame([_|T], Att, X, [_|NT]) :- !,
        build_frame(T, Att, X, NT).

/*********************************************************************/

% compile_lexicon/0 compiles out the lexicon and asserts clauses
% corresponding to each lexical entry, by finding all constituents that
% can be classified as lexical constituents.

compile_lexicon :-
        format("Compiling lexical_entry/1 clauses ...", []), ttyflush,
        classify(lexical_constituent, Sign), write('.'), ttyflush,
        asserta(lexical_entry(Sign)),
        fail.
compile_lexicon :-
        format(" (Applying lexical rules)~n", []), ttyflush,
        apply_lexical_rules,
        format("Finished compiling lexical_entry/1 clauses.~n", []),
        ttyflush.

/*********************************************************************/

save_value :-
        format("Saving value/2 ...", []), ttyflush,
        tell('value.pl'),
        list_all(value(_,_)),
        told,
        format(" Finished saving value/2 clauses.~n", []), ttyflush.

/*********************************************************************/

save_lexicon :-
        format("Saving lexical_entry/1 ...", []), ttyflush,
        tell('lexicon.pl'),
        list_all(lexical_entry(_)),
        told,
        format(" Finished saving lexical_entry/1 clauses.~n", []),
        ttyflush.

/*********************************************************************/

list_all(Head) :-
        clause(Head, Body),
        write_clause(Head, Body),
        fail.
list_all(_).

/*********************************************************************/

write_clause(Head, true) :- !,
```

```
        writeq(Head), write('.'), nl.
write_clause(Head, Body) :-
        writeq(Head), write(' :- '), nl,
        write('        '), writeq(Body), write('.'), nl.

/*****************************************************************/

:- compile([attributes]), ttyflush.

:- dynamic value/2.

:- compile_value, save_value.

:- dynamic lexical_entry/1.

:- multifile data/1.

data( dummy ). % this avoids a bug in the SICStus multifile declaration!

:- consult([full_hierarchy,
            lexical_rules]),
   compile([abbreviations]),
   ttyflush.

:- dynamic apply_inflectional_rule/1, apply_non_inflectional_rule/1.

:- compile([partial_execute]), partially_execute.

:- compile_lexicon, save_lexicon, halt.

/*****************************************************************/
```

14 partial_execute.pl

```
/*****************************************************************

        File:               partial_execute.pl
        Author:             Richard Cooper
        Date:               25/04/90

This file contains a standard sort of partial executer.  It partially
executes clauses whose principle functor is either abbreviates or <=>.

*****************************************************************/

partially_execute :-
        data((Head :- Body)),
        partially_execute(Body, ExpandedBody),
        format("~w partially executed~n", [Head]), ttyflush,
        assertz((Head :- ExpandedBody)),
```

```
        fail.
partially_execute.

partially_execute((Literal, Rest), Expansion) :-
        partially_execute(Literal, ExpandedLiteral),
        partially_execute(Rest, ExpandedRest),
        conjoin(ExpandedLiteral, ExpandedRest, Expansion), !.
partially_execute(A abbreviates B, true) :- !,
        A abbreviates B.
partially_execute(A <=> B, true) :- !,
        A <=> B.
partially_execute(Literal, Literal).

/*****************************************************************/

% conjoin/3 removes any calls which reduce to true.

conjoin((A,B),C,ABC) :-
        conjoin(B,C,BC),
        conjoin(A,BC,ABC).
conjoin(true, A, A) :- !.
conjoin(A, true, A) :- !.
conjoin(A,C,(A,C)).

/*****************************************************************/
```

15 front_end.pl

```
/*****************************************************************

        File:               front_end.pl
        Author:             Richard Cooper
        Date:               26/10/89

This file contains the overall program control clauses which the
top-down and bottom-up parsers share.  It also loads all i/o routines.

*****************************************************************/

:- compile([foreign]).
:- consult([sunview_print]).
:- compile([read]).

/*****************************************************************/

go :-
        space(S),
        format("~5n~28cProlog Parser for CPSG~2n", [S]),
        format("~30cby Richard Cooper~n", [S]),
        format("~31c22nd June 1990~10n", [S]),
```

```
        ttyflush,
        parse,
        format("~n~n", []),
        put_menu,
        loop.

/*********************************************************************/

loop :-
        get_selection(N),
        take_action(N),
        loop.

/*********************************************************************/

put_menu :-
        space(S),
        format("~10c1.  Parse.~n", [S]),
        format("~10c2.  List wffs of previous parse.~n", [S]),
        format("~10c3.  Save an avm to a file.~n", [S]),
        format("~10c4.  Display an avm on the terminal.~n", [S]),
        format("~10c5.  Display an avm in a SunView window.~n", [S]),
        format("~10c6.  Display a tree in a SunView window.~n", [S]),
        format("~10c7.  Toggle display of daughters.~n", [S]),
        format("~10c8.  Debug.~n", [S]),
        format("~10c9.  Exit.~2n", [S]).

/*********************************************************************/

get_selection(N) :-
        prompt(Previous, 'Enter Selection: '),
        read_integer(N),
        prompt(_, Previous), nl, !.

/*********************************************************************/

take_action(N) :-
        N =< 0,
        put_menu,
        get_selection(M),
        take_action(M).
take_action(1) :-
        parse.
take_action(2) :-
        wffs.
take_action(3) :-
        get_file_name(File),
        (valid_file_name(File) ->
            (get_avm_number(N),
             save_avm_in_file(N, File));
            (format("~nInvalid filename: avm not saved~2n", []))).
take_action(4) :-
        get_avm_number(N),
```

```
        draw_avm_on_terminal(N).
take_action(5) :-
        get_avm_number(N),
        display_avm(N), nl.
take_action(6) :-
        get_avm_number(N),
        display_tree(N), nl.
take_action(7) :-
        retract(display_daughters(X)),
        toggle_daughters(X, Y),
        assert(display_daughters(Y)).
take_action(8) :-
        trace, nl.
take_action(9) :-
        format("Bye ...~n", []),
        halt.
take_action(_) :-
        format("Selection out of range!~2n", []),
        loop.

/*********************************************************************/

get_file_name(FileName) :-
        prompt(Previous, 'Input file name: '),
        read_to_end_of_line(String),
        prompt(_, Previous),
        name(FileName, String), !.

/*********************************************************************/

valid_file_name(FileName) :-
        name(FileName, [H|_]),
        letter(H).

/*********************************************************************/

toggle_daughters(yes, no) :-
        format("Daughters will not be displayed.~2n", []), ttyflush.
toggle_daughters(no, yes) :-
        format("Daughters will be displayed.~2n", []), ttyflush.

/*********************************************************************/

:- dynamic display_daughters/1.

display_daughters(yes).

/*********************************************************************/
```

16 bottom_up_front_end.pl

```
/*********************************************************************

        File:                   bottom_up_front_end.pl
        Author:                 Richard Cooper
        Date:                   26/10/89

This file contains the program control clauses which are unique to the
bottom-up parser.  It also loads all proram control clauses, and the i/o
routines.

*********************************************************************/

:- compile([front_end]).

/*********************************************************************/

get_avm_number(N) :-
        prompt(Previous, 'Select avm: '),
        read_integer(N),
        prompt(_, Previous), !.

/*********************************************************************/

wffs :-
        chart(S, Ref),
        S/phon <=> list(T),
        format("~d: ~w~n", [Ref, T]), ttyflush,
        fail.
wffs :-
        nl.

/*********************************************************************/
```

17 top_down_front_end.pl

```
/*********************************************************************

        File:                   top_down_front_end.pl
        Author:                 Richard Cooper
        Date:                   04/10/90

This file contains the program control clauses which are unique to the
top-down parser.  It also loads all proram control clauses, and the i/o
routines.  This parser is less flexible than the bottom-up parse,
finding only a single solution and hence only allowing output of one
solution.
```

```
*************************************************************************/

:- compile([front_end]).

/*************************************************************************/

get_avm_number(0).

/*************************************************************************/
```

18 read.pl

```
/*************************************************************************

        File:               read.pl
        Author:             Richard Cooper
        Date:               23/04/90

This file contains routines for converting terminal input into lists of
atoms.

*************************************************************************/

:- mode get_partial_description(-).

get_partial_description(Sign) :-
        prompt(Previous, '> '),
        read_input(String),
        prompt(_, Previous), !,
        Sign/phon <=> list(String).

/*************************************************************************/

:- mode read_input(-).

%  read_input/1 reads terminal input and converts it to a list of atoms
%  corresponding to the words in the input. Input should be terminated
%  by a newline and contain on punctuation characters.

%         Input is parsed by the grammar
%                         input([W|T]) --> word(W) rest(T)
%
%                         rest(T) --> space input(T)
%                         rest([]) --> newline

%  Next is a look-ahead character which read_word/2 must examine to
%  determine when the end of a word has been found.

read_input([Word|Rest]) :-
        read_word(Word, Next),
```

```
        read_rest(Next, Rest).

/*****************************************************************/

:- mode read_rest(+, -).

% Arg1 is a look-ahead character. Arg2 is the list of words remaining
% in the input.

read_rest(C, []) :-
        newline(C), !.
read_rest(C, T) :-
        space(C), !,
        read_input(T).
read_rest(C, []) :-
        format('Bad character in input stream: "~c".', [C]),
        format('Remainder of input ignored.',[]), nl,
        newline(L), prompt(Prev, ''), ttyskip(L), prompt('', Prev).

/*****************************************************************/

:- mode read_word(-,-).

% read_word/2 reads the next word in the input stream. Next is a
% look-ahead character.

read_word(Word, Next) :-
        get(Letter),
        read_rest_of_word(Letter, Characters, Next),
        name(Word, Characters).

/*****************************************************************/

:- mode read_rest_of_word(+, -, -).

% This procedure uses tail recursion to process the remainder of a word
% after a word  has been detected. It is similar to read_token/3, but is
% required to ensure that punctuation marks are not included at the end
% of words.

read_rest_of_word(Char, [Char|Characters], Next2) :-
        letter_or_apostrophe(Char), !,
        get0(Next),
        read_rest_of_word(Next, Characters, Next2).
read_rest_of_word(Next, [], Next).

/*****************************************************************/

:- mode read_integer(-).

read_integer(N) :-
        get0(Buffer),
        read_integer(Buffer,  0, N).
```

```
read_integer(Buffer, Result, Result) :-
        newline(Buffer), !.
read_integer(Buffer,  SoFar, Result) :-
        digit(Buffer, Value), !,
        NSF is 10*SoFar + Value,
        get0(NewBuffer),
        read_integer(NewBuffer, NSF, Result).
read_integer(C,  _, -1) :-
        format('Bad character in input stream: "~c".',[C]),
        format('Input ignored.', []), nl,
        newline(L), prompt(Prev, ''), ttyskip(L), prompt('', Prev).

/*****************************************************************/

read_to_end_of_line(S) :-
        get0(Buffer),
        read_to_end_of_line(Buffer, S).

read_to_end_of_line(Buffer, []) :-
        newline(Buffer).
read_to_end_of_line(Buffer, [Buffer|L]) :-
        get0(NewBuffer),
        read_to_end_of_line(NewBuffer, L).

/*****************************************************************/

:- mode letter_or_apostrophe(+).

% Succeeds if Arg1 is the ascii value of a letter or apostrophe.

letter_or_apostrophe(39).        % Apostrophe
letter_or_apostrophe(A) :-
        letter(A).

letter(A) :-
        A >= 65, A =< 90.        % Lowercase letter
letter(A) :-
        A >= 97, A =< 122.        % Uppercase letter

/*****************************************************************/

digit(D, Value) :-
        D > 47, D < 58,
        Value is D - 48.

/*****************************************************************/

:- mode space(?).

% Succeeds if Arg1 is the ascii value for a space, i.e. 32.

space(32).
```

```
/*****************************************************************/

:- mode newline(?).

% Succeeds if Arg1 is the ascii value for a newline, i.e., 10.

newline(10).

/*****************************************************************/
```

19 print.pl

```
/*****************************************************************

        File:               print.pl
        Author:             Richard Cooper
        Date:               26/10/89

This file contains routines for pretty printing avms in ascii.

*****************************************************************/

:- compile([attributes]).

/*****************************************************************/

save_avm_in_file(N, File) :-
        chart(Sign, N) ->
                (tell(File),
                print_avm(Sign),
                told,
                format("~nAvm successfully saved.~2n", []));
                format("~nNo such avm! ", []),
                format("List wffs to see valid avm numbers.~2n", []).

/*****************************************************************/

draw_avm_on_terminal(N) :-
        chart(Sign, N) ->
                print_avm(Sign);
                format("~nNo such avm! ", []),
                format("List wffs to see valid avm numbers.~2n", []).

/*****************************************************************/

print_avm(AVM) :-
        asserta(cross_reference(1)),
        print_avm(AVM, 0, [], Out), nl,
        print_avm_code_list(Out),
```

```
        retract(cross_reference(_)), !.

/*************************************************************************/

print_avm_code_list([]).
print_avm_code_list([H:N | T]) :-
        write('#'), left_justify(N, 3), write('  '),
        print_avm(H, 6, [], Out), nl,
        print_avm_code_list(Out), nl,
        print_avm_code_list(T).

/*************************************************************************

print_avm(AVM, Indent, In, Out) prints the whole AVM starting from
current cursor position (assumed to be at Indent) and ending at
position 0 on the next line.  In and Out are lists of avm:number
pairs.  Elements of Out should each be print_avmed at the end.  In
should intially be [].

*************************************************************************/

print_avm(AVM, _, In, In) :-
        var(AVM), !,
        write(AVM), nl.
print_avm(AVM, _, In, In) :-
        atom(AVM), !,
        write(AVM), nl.
print_avm((_,M), _, In, In) :-
        var(M), !,
        write(M), nl.
print_avm(list(L), _, In, Out) :-
        !, write('['), print_list(L, In, Out), write(']'), nl.
print_avm(set(L), _, In, Out) :-
        !, write('{'), print_list(L, In, Out), write('}'), nl.
print_avm((S,[V|T]), Indent, In, Out) :-
        avm_frame(S, [A|Rest]),
        left_justify(A, 10), NewIndent is Indent + 10,
        print_avm(V, NewIndent, In, Temp),
        print_rest_of_avm(T, Rest, Indent, Temp, Out).

/*************************************************************************/

print_rest_of_avm([], [], _, XRefs, XRefs).
print_rest_of_avm([V|T], [dtrs|Rest], Indent, In, Out) :-
        display_daughters(no), !,
        tab(Indent), left_justify(dtrs, 10),
        NewIndent is Indent + 10,
        dont_print_dtrs(V, NewIndent, In, Temp),
        print_rest_of_avm(T, Rest, Indent, Temp, Out).
print_rest_of_avm([V|T], [A|Rest], Indent, In, Out) :-
        tab(Indent), left_justify(A, 10),
        NewIndent is Indent + 10,
        print_avm(V, NewIndent, In, Temp),
```

```
        print_rest_of_avm(T, Rest, Indent, Temp, Out).

/*****************************************************************/

dont_print_dtrs(V, NewIndent, Out, Out) :-
        var(V), !,
        write(V), nl.
dont_print_dtrs(A, NewIndent, Out, Out) :-
        atom(A), !,
        write(A), nl.
dont_print_dtrs(_, NewIndent, Out, Out) :-
        write('...'), nl.

/*****************************************************************/

print_list([], In, In).
print_list([A], In, Out) :-
        !, print_list_item(A, In, Out).
print_list([H|T], In, Out) :-
        print_list_item(H, In, Temp), write(', '),
        print_list(T, Temp, Out).

/*****************************************************************/

print_list_item(X, In, In) :-
        list_of_atoms(X), !,
        write(X).
print_list_item(X, In, In) :-
        identical_member(X, In, N),
        write('#'), write(N).
print_list_item(X, In, [X:N|In]) :-
        retract(cross_reference(N)),
        N1 is N + 1,
        asserta(cross_reference(N1)),
        write('#'), write(N).

/*****************************************************************/

identical_member(X, [Y:N|_], N) :-
        X == Y.
identical_member(X, [_|T], N) :-
        identical_member(X, T, N).

/*****************************************************************/

list_of_atoms(X) :-
        var(X), !.
list_of_atoms(X) :-
        atom(X), !.
list_of_atoms([]).
list_of_atoms([H|T]) :-
        list_of_atoms(H),
        list_of_atoms(T).
```

```
/*****************************************************************/

left_justify(Label, Width) :-
        name(Label, L),
        length(L, M),
        T is Width - 1 - M,
        write(Label), write(':'), tab(T).

/*****************************************************************/
```

20 sunview_print.pl

```
/*****************************************************************

        File:                   sunview_print.pl
        Author:                 Richard Cooper
        Date:                   26/10/89

*****************************************************************/

%       parameters for avm drawing routines

letter_width(10).
text_height(11).
space_between_a_and_v(15).
interline_space(2).
between_avm_hspace(3).
angle_bracket_width(20).
extra_hspace(2).
extra_vspace(2).

%       parameters for tree drawing routine

between_node_space(40).
inter_level_space(100).

/*****************************************************************/

display_avm(N) :-
        chart(AVM, N) ->
                (draw_avm(AVM, 100, Xdim, 100, Ydim, no_draw),
                 initialise_avm_frame(Xdim, Ydim, 1),
                 draw_avm(AVM, 50, _, 50, _, draw),
                 display_frame(1));
                format("~nNo such avm! ", []),
                format("List wffs to see valid avm numbers.~2n", []).

/*****************************************************************/
```

```
% A call to draw_avm/7 looks like
%          draw_avm(AVM, Xleft, Xright, Ytop, Ybottom, Flag)
%  AVM is the attribute value matrix to be drawn.  Flag is a flag to
% indicate whether it should be drawn or not.  If set to no_draw the avm
% is not drawn but its dimensions are calculated.  Xleft and Ytop should
% be initially instantiated they give the coordinates of the top left
% hand corner of the avm.  If the call is successful, Xright and Ybottom
% will be instantiated on its return to the coordinates of the lower
% right hand corner.  Note that SunView uses a coordinate system with X
% increasing from left to right and Y increasing from top to bottom.

draw_avm(V, Xleft, Xright, Ytop, Ybottom, Flag) :-
        var(V), !,
        draw_avm('?', Xleft, Xright, Ytop, Ybottom, Flag).

draw_avm(A, Xleft, Xright, Ytop, Ybottom, Flag) :-
        atom(A), !,
        extra_vspace(V),
        text_height(T),
        extra_hspace(H),
        TextLeft is Xleft + H,
        TextBaseLine is Ytop + V + T,
        put_text(A, TextLeft, TextBaseLine, Flag, 1),
        length_of_text(A, Xextent),
        Ybottom is TextBaseLine + V,
        Xright is TextLeft + Xextent + H.

draw_avm(list(V), Xleft, Xright, Ytop, Ybottom, Flag) :-
        var(V), !,
        draw_avm('<???>', Xleft, Xright, Ytop, Ybottom, Flag).

draw_avm(list([]), Xleft, Xright, Ytop, Ybottom, Flag) :- !,
        draw_avm('< >', Xleft, Xright, Ytop, Ybottom, Flag).

draw_avm(list(AVMs), Xleft, Xright, Ytop, Ybottom, Flag) :- !,
        angle_bracket_width(B),
        extra_hspace(H),
        Left is Xleft + B + H,
        draw_list_of_avms(AVMs, Left, Right, Ytop, Ybottom, Flag),
        Xright is Right + B + H,
        draw_angle_brackets(Xleft, Xright, Ytop, Ybottom, Flag, 1).

draw_avm(set(V), Xleft, Xright, Ytop, Ybottom, Flag) :-
        var(V), !,
        draw_avm('{???}', Xleft, Xright, Ytop, Ybottom, Flag).

draw_avm(set([]), Xleft, Xright, Ytop, Ybottom, Flag) :- !,
        draw_avm('{ }', Xleft, Xright, Ytop, Ybottom, Flag).

draw_avm(set(AVMs), Xleft, Xright, Ytop, Ybottom, Flag) :- !,
        angle_bracket_width(B),
        extra_hspace(H),
        Left is Xleft + B + H,
```

```
        draw_list_of_avms(AVMs, Left, Right, Ytop, Ybottom, Flag),
        Xright is Right + B + H,
        draw_curly_brackets(Xleft, Xright, Ytop, Ybottom, Flag, 1).

draw_avm(AVM, Xleft, Xright, Ytop, Ybottom, Flag) :-
        uninstantiated_avm(AVM),
        draw_avm('[ ]', Xleft, Xright, Ytop, Ybottom, Flag).

draw_avm((Type,Values), Xleft, Xright, Ytop, Ybottom, Flag) :-
        avm_frame(Type, Attributes),
        longest_label(Attributes, LabelLength),
        extra_hspace(H),
        extra_vspace(V),
        space_between_a_and_v(AV),
        Left is Xleft + H + H,
        Mid is Left + LabelLength + AV,
        Top is Ytop + V,
        draw_av_pairs(Attributes, Values, Left, Mid,
                      Right, Top, Bottom, Flag),
        Ybottom is Bottom + V,
        Xright is Right + H + H,
        LeftBracket is Xleft + H,
        RightBracket is Xright - H,
        draw_square_brackets(LeftBracket, RightBracket,
                             Top, Bottom, Flag, 1).

/*****************************************************************/

draw_av_pairs([dtrs], [Val], Left, XMid, Right, Ytop, Ybottom, Flag) :-
        display_daughters(no), !,
        extra_vspace(V),
        text_height(TextHeight),
        ValueTop is Ytop + V,
        dont_draw_daughters(Val, XMid, Right,
                            ValueTop, ValueBottom, Flag),
        Ybottom is ValueBottom + V,
        TextBaseLine is (Ytop + Ybottom + TextHeight) // 2,
        put_text(dtrs, Left, TextBaseLine, Flag, 1).

draw_av_pairs([Att], [Val], Left, XMid, Right, Ytop, Ybottom, Flag) :-
        extra_vspace(V),
        text_height(TextHeight),
        ValueTop is Ytop + V, !,
        draw_avm(Val, XMid, Right, ValueTop, ValueBottom, Flag),
        Ybottom is ValueBottom + V,
        TextBaseLine is (Ytop + Ybottom + TextHeight) // 2,
        put_text(Att, Left, TextBaseLine, Flag, 1).

draw_av_pairs([Att|OtherAs], [Val|OtherVs], Left, XMid,
              Right, Ytop, Ybottom, Flag) :-
        draw_av_pairs([Att], [Val], Left, XMid, Xtemp1,
                      Ytop, Ytemp, Flag),
        extra_vspace(V),
```

```
        Ynext is Ytemp - V,
        draw_av_pairs(OtherAs, OtherVs, Left, XMid, Xtemp2,
                      Ynext, Ybottom, Flag),
        maximum(Xtemp1, Xtemp2, Right).

/*****************************************************************/

draw_list_of_avms([H], Left, Right, Ytop, Ybottom, Flag) :- !,
        draw_avm(H, Left, Right, Ytop, Ybottom, Flag).

draw_list_of_avms([H|T], Left, Right, Ytop, Ybottom, Flag) :-
        draw_avm(H, Left, Mid, Ytop, Y1, Flag),
        between_avm_hspace(BAS),
        NewLeft is Mid + BAS,
        draw_list_of_avms(T, NewLeft, Right, Ytop, Y2, Flag),
        maximum(Y1, Y2, Ybottom).

/*****************************************************************/

dont_draw_daughters(V, XMid, Right, ValueTop, ValueBottom, Flag) :-
        var(V), !,
        draw_avm(V, XMid, Right, ValueTop, ValueBottom, Flag).
dont_draw_daughters(A, XMid, Right, ValueTop, ValueBottom, Flag) :-
        atom(A), !,
        draw_avm(A, XMid, Right, ValueTop, ValueBottom, Flag).
dont_draw_daughters(_, XMid, Right, ValueTop, ValueBottom, Flag) :-
        draw_avm('...', XMid, Right, ValueTop, ValueBottom, Flag).

/*****************************************************************/

display_tree(N) :-
        chart(AVM, N) ->
                (draw_tree(AVM, 100, Xdim, 100, _, Ydim, no_draw),
                initialise_tree_frame(Xdim, Ydim, 1),
                draw_tree(AVM, 50, _, 50, _, _, draw),
                display_frame(1));
                format("~nNo such avm! ", []),
                format("List wffs to see valid avm numbers.~2n", []).

/*****************************************************************/

% A call to draw_tree/7 looks like
%          draw_tree(AVM, Left, Right, Top, Root, Bottom, Flag)
% AVM is the attribute value matrix encoding the tree to be drawn.
% Flag is a flag to indicate whether the tree should be drawn or not.
% If set to no_draw the tree is not drawn but its dimensions are
% calculated.  Left and Top should be initially instantiated they give
% the coordinates of the top left hand corner of the box surrounding the
% tree.  If the call is successful, Right and Bottom will be
% instantiated on its return to the coordinates of the lower right hand
% corner, and Root will be instanted to the y coordinate of the root
% node of the tree.  Note that SunView uses a coordinate system with X
% increasing from left to right and Y increasing from top to bottom.
```

```
draw_tree(AVM, Left, Right, Top, Root, Bottom, Flag) :-
        AVM/dtrs <=> null, !,
        between_node_space(BNS),
        Tmp is Left + BNS,
        AVM/phon <=> list(Phon),
        extra_vspace(V),
        text_height(TH),
        TextPlace is Top + TH + V,
        Bottom is TextPlace + V,
        convert_phon_to_atom(Phon, Text),
        put_text(Text, Tmp, TextPlace, Flag, 1),
        length_of_text(Text, LabelLength),
        Right is Left + BNS + LabelLength + BNS,
        Root is (Left + Right) // 2.

draw_tree(AVM, Left, Right, Top, Root, Bottom, Flag) :-
        inter_level_space(ILS),
        extra_vspace(V),
        text_height(TH),
        DtrTop is Top + TH + V + ILS,
        NodeBtm is Top + TH + V,
        get_dtrs_list(AVM, Dtrs),
        draw_dtr_trees(Dtrs, Left, Left, Right,
                       DtrTop, NodeBtm, Root, Bottom, Flag),
        AVM/phon <=> list(Phon),
        YTextPlace is Top + TH + V,
        convert_phon_to_atom(Phon, Text),
        length_of_text(Text, LabelLength),
        XTextPlace is Root - (LabelLength // 2),
        put_text(Text, XTextPlace, YTextPlace, Flag, 1).

/*****************************************************************/

draw_dtr_trees([H], FarLeft, Left, Right,
               DtrTop, NodeBtm, Root, Bottom, Flag) :-
        draw_tree(H, Left, Right, DtrTop, DtrRoot, Bottom, Flag),
        Root is (FarLeft + Right) // 2,
        put_line(Root, NodeBtm, DtrRoot, DtrTop, Flag, 1).

draw_dtr_trees([H|T], FarLeft, Left, FarRight,
               DtrTop, NodeBtm, Root, Bottom, Flag) :-
        draw_tree(H, Left, Right, DtrTop, DtrRoot, Bottom1, Flag),
        draw_dtr_trees(T, FarLeft, Right, FarRight,
                       DtrTop, NodeBtm, Root, Bottom2, Flag),
        maximum(Bottom1, Bottom2, Bottom),
        put_line(Root, NodeBtm, DtrRoot, DtrTop, Flag, 1).

/*****************************************************************/

% Clauses for to buffer interface to SunView.  Only if the 2nd last
% argument is the atom draw is anything actually drawn.
```

```
put_text(Text, X, Y, no_draw, 1).
put_text(Text, X, Y, draw, 1) :-
        put_text(Text, X, Y, 1).

put_line(X1, Y1, X2, Y2, no_draw, 1).
put_line(X1, Y1, X2, Y2, draw, 1) :-
        put_line(X1, Y1, X2, Y2, 1).

draw_angle_brackets(Xleft, Xright, Ytop, Ybottom, no_draw, 1).
draw_angle_brackets(Xleft, Xright, Ytop, Ybottom, draw, 1) :-
        angle_bracket_width(W),
        extra_vspace(V),
        Ymid is (Ytop + Ybottom) // 2,
        InsideLeft is Xleft + (W - V),
        InsideRight is Xright - (W - V),
        put_line(InsideRight, Ytop, Xright, Ymid, 1),
        put_line(InsideRight, Ybottom, Xright, Ymid, 1),
        put_line(InsideLeft, Ytop, Xleft, Ymid, 1),
        put_line(InsideLeft, Ybottom, Xleft, Ymid, 1).

draw_curly_brackets(Xleft, Xright, Ytop, Ybottom, no_draw, 1).
draw_curly_brackets(Xleft, Xright, Ytop, Ybottom, draw, 1) :-
        angle_bracket_width(W),
        extra_vspace(V),
        Ymid is (Ytop + Ybottom) // 2,
        Ytm is Ytop + V,
        Ymt is Ymid - V,
        Ymb is Ymid + V,
        Ybm is Ybottom - V,
        InsideLeft is Xleft + (W - V),
        InsideRight is Xright - (W - V),
        put_line(InsideRight, Ytop, Xright, Ytm, 1),
        put_line(Xright, Ytm, InsideRight, Ymt, 1),
        put_line(InsideRight, Ymt, Xright, Ymid, 1),
        put_line(Xright, Ymid, InsideRight, Ymb, 1),
        put_line(InsideRight, Ymb, Xright, Ybm, 1),
        put_line(Xright, Ybm, InsideRight, Ybottom, 1),
        put_line(InsideLeft, Ytop, Xleft, Ytm, 1),
        put_line(Xleft, Ytm, InsideLeft, Ymt, 1),
        put_line(InsideLeft, Ymt, Xleft, Ymid, 1),
        put_line(Xleft, Ymid, InsideLeft, Ymb, 1),
        put_line(InsideLeft, Ymb, Xleft, Ybm, 1),
        put_line(Xleft, Ybm, InsideLeft, Ybottom, 1).

draw_square_brackets(Xleft, Xright, Ytop, Ybottom, no_draw, 1).
draw_square_brackets(Xleft, Xright, Ytop, Ybottom, draw, 1) :-
        InsideLeft is Xleft + 5,
        InsideRight is Xright - 5,
        put_line(InsideRight, Ytop, Xright, Ytop, 1),
        put_line(InsideRight, Ybottom, Xright, Ybottom, 1),
        put_line(InsideLeft, Ytop, Xleft, Ytop, 1),
        put_line(InsideLeft, Ybottom, Xleft, Ybottom, 1),
        put_line(Xleft, Ytop, Xleft, Ybottom, 1),
```

```
        put_line(Xright, Ytop, Xright, Ybottom, 1).

/*****************************************************************/

% miscellaneous clauses for the drawing routines:

longest_label([A], L) :-
        length_of_text(A, L).
longest_label([A|Rest], L) :-
        longest_label(Rest, LTemp),
        length_of_text(A, SL),
        maximum(SL, LTemp, L).

length_of_text(Text, Space) :-
        letter_width(S),
        name(Text, List),
        length(List, L),
        Space is L * S.

convert_phon_to_atom([], '').
convert_phon_to_atom([Phon], Phon) :- !.
convert_phon_to_atom([Head|Tail], Phon) :-
        convert_phon_to_atom(Tail, TP),
        name(Head, L),
        name(TP, M),
        space(S),
        append(L, [S|M], N),
        name(Phon, N).

/*****************************************************************/
```

21 foreign.pl

```
/*****************************************************************

        File:               foreign.pl
        Author:             Richard Cooper
        Date:               26/04/90

This file contains the declarations of the c functions requried to
draw in the SunView canvas.  The source for the functions resides in
canvas.c.

*****************************************************************/

foreign_file('canvas.o', [initialise_avm_frame,
                          initialise_tree_frame,
                          display_frame,
                          put_text,
                          put_line]).
```

```
/*****************************************************************/

% initialise_avm_frame/3: Takes x-dimension and y-dimension of the
% canvas to be initiliased for the drawing of an avm.  Returns 1 as the
% third argument if successful.

foreign(initialise_avm_frame, c,
        initialise_avm_frame(+integer, +integer, [-integer])).

% initialise_tree_frame/3: Takes x-dimension and y-dimension of the
% canvas to be initiliased for the drawing of a tree.  Returns 1 as the
% third argument if successful.

foreign(initialise_tree_frame, c,
        initialise_tree_frame(+integer, +integer, [-integer])).

% display_frame/1: Displays the current frame. Returns 1 if successful.

foreign(display_frame, c,
        display_frame([-integer])).

% put_text/4: Puts a Prolog atom at a specified x,y position, Returns 1
% if successful.

foreign(put_text, c,
        put_text(+string, +integer, +integer, [-integer])).

% put_line/5: Draws a line from x,y (given by first two arguments) to
% x',y' (given by seccond two arguments. Returns 1 if successful.

foreign(put_line, c,
        put_line(+integer, +integer, +integer, +integer, [-integer])).

/*****************************************************************/

% The foreign file must be linked with suntool, sunwindow and pixrect
% library files:

:- load_foreign_files(['canvas.o'],
                      ['-lsuntool', '-lsunwindow', '-lpixrect']).

/*****************************************************************/
```

22 canvas.c

```
/*****************************************************************

        File:               canvas.c
        Author:             Richard Cooper
```

```
        Date:                   26/04/90

This file contains the c routines for drawing in the SunView window.

It must be complied as:

          cc -o canvas.o canvas.c -lsuntool -lsunwindow -lpixrect

All calls are to standard SunView library routines.

*****************************************************************/

#include <suntool/sunview.h>
#include <suntool/canvas.h>
#include <suntool/textsw.h>
#include <stdio.h>

#define minimum(x1, x2) ((x1) > (x2) ? (x2) : (x1))
#define maximum(x1, x2) ((x1) > (x2) ? (x1) : (x2))

static Frame frame;
static Canvas canvas;
static Textsw textsw;

long initialise_avm_frame(x_max, y_max)
long x_max, y_max;

{
        frame = window_create(NULL, FRAME,
                FRAME_LABEL, "Feature Structure",
                WIN_WIDTH, minimum(maximum(500,x_max),900),
                WIN_HEIGHT, minimum(y_max,900),
                0);

        canvas = window_create(frame, CANVAS,
                WIN_WIDTH, minimum(maximum(500,x_max),900),
                WIN_HEIGHT, minimum(y_max,900),
                CANVAS_AUTO_SHRINK, FALSE,
                CANVAS_WIDTH, x_max,
                CANVAS_HEIGHT, y_max,
                WIN_VERTICAL_SCROLLBAR, scrollbar_create(0),
                WIN_HORIZONTAL_SCROLLBAR, scrollbar_create(0),
                0);

        return(1);
}

long initialise_tree_frame(x_max, y_max)
long x_max, y_max;

{
        frame = window_create(NULL, FRAME,
                FRAME_LABEL, "Phrase Structure Tree:",
```

```
                WIN_WIDTH, minimum(maximum(500,x_max),900),
                WIN_HEIGHT, minimum(y_max,900),
                0);

        canvas = window_create(frame, CANVAS,
                WIN_WIDTH, minimum(maximum(500,x_max),900),
                WIN_HEIGHT, minimum(y_max,900),
                CANVAS_AUTO_SHRINK, FALSE,
                CANVAS_WIDTH, x_max,
                CANVAS_HEIGHT, y_max,
                WIN_VERTICAL_SCROLLBAR, scrollbar_create(0),
                WIN_HORIZONTAL_SCROLLBAR, scrollbar_create(0),
                0);

        return(1);
}

long display_frame()

{
        window_main_loop(frame);
        return(1);
}

long put_text(text_to_put, x_position, y_position)
char *text_to_put;
long x_position, y_position;

{
        Pixwin *pw;

        pw = canvas_pixwin(canvas);
        pw_text(pw, x_position, y_position, PIX_SRC, 0, text_to_put);
        return(1);
}

long put_line(x_1, y_1, x_2, y_2)
long x_1, y_1, x_2, y_2;

{
        Pixwin *pw;

        pw = canvas_pixwin(canvas);
        pw_vector(pw, x_1, y_1, x_2, y_2, PIX_SRC, 1);
        return(1);
}
```