DESCRIPTION BASED PARSING IN A CONNECTIONIST NETWORK

James Brinton Henderson

A Dissertation
in
Computer and Information Science

Presented to the Faculties of the University of Pennsylvania
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

1994

Mitchell Marcus
Supervisor of Dissertation

Mark Steedman
Graduate Group Chairperson

© Copyright
James Brinton Henderson
1994

This dissertation is dedicated to my parents, Rolland and Janet Henderson, and to my grandmother, Rebecca Hetzel, and the memories of Theodore Hetzel, Walter Henderson, and Emma Henderson.

Acknowledgements

I'd like to gratefully acknowledge the help and support of my advisor, Mitch Marcus. I'd also like to particularly thank my committee members, Don Hindle, Tony Kroch, Lokendra Shastri, and Mark Steedman. Their input was crucial to making this dissertation both interesting from a variety of perspectives, and completable. Special thanks goes to Lokendra Shastri for his encouragement and advice.

Many people have had an impact on this work. Thanks to the members of the CLiFF group, who heard and commented on countless versions of this work over the years. Thanks to the residents of IRCS, for many spontaneous, thoughtful discussions. I also appreciate the many conversations I've had with visitors to IRCS and at conferences, particularly at the CUNY Conference on Human Sentence Processing.

None of this would be possible without the unwavering support and encouragement of my parents, for which I am very grateful. I've also depended on my friends at Penn, who have tried their best to keep me sane, and the purveyors of truck food, who have tried their best to keep me fed. Special thanks to my friends Ivan, Hunter, Raphael, Mike, and Abigail.

Abstract

Description Based Parsing in a Connectionist Network

James Brinton Henderson

Mitchell Marcus

Recent developments in connectionist architectures for symbolic computation have made it possible to investigate parsing in a connectionist network while still taking advantage of the large body of work on parsing in symbolic frameworks. This dissertation investigates syntactic parsing in the temporal synchrony variable binding model of symbolic computation in a connectionist network. This computational architecture solves the basic problem with previous connectionist architectures, while keeping their advantages. However, the architecture does have some limitations, which impose computational constraints on parsing in this architecture. This dissertation argues that, despite these constraints, the architecture is computationally adequate for syntactic parsing, and that these constraints make significant linguistic predictions. To make these arguments, the nature of the architecture's limitations are first characterized as a set of constraints on symbolic computation. This allows the investigation of the feasibility and implications of parsing in the architecture to be investigated at the same level of abstraction as virtually all other investigations of syntactic parsing. Then a specific parsing model is developed and implemented in the architecture. The extensive use of partial descriptions of phrase structure trees is crucial to the ability of this model to recover the syntactic structure of sentences within the constraints. Finally, this parsing model is tested on those phenomena which are of particular concern given the constraints, and on an approximately unbiased sample of sentences to check for unforeseen difficulties. The results show that this connectionist architecture is powerful enough for syntactic parsing. They also show that some linguistic phenomena are predicted by the limitations of this architecture. In particular, explanations are given for many cases of unacceptable center embedding, and for several significant constraints on long distance dependencies. These results give evidence for the cognitive significance of this computational architecture and parsing model. This work also shows how the advantages of both connectionist and symbolic techniques can be unified in natural language processing applications. By analyzing how low level biological and computational considerations influence higher level processing, this work has furthered our understanding of the nature of language and how it can be efficiently and effectively processed.

Contents

Acknowledgements **iv**

Abstract **v**

1 Introduction **1**

1.1 Motivations . . . 3

1.1.1 Investigating Architecture-Level Constraints . . . 3

1.1.2 Combining Connectionism with Linguistic Theories . . . 4

1.2 Overview of the Computational Constraints . . . 5

1.2.1 Constraints on Previous Connectionist Parsers . . . 6

1.2.2 The Connectionist Architecture . . . 7

1.2.3 Constraints on Syntactic Parsing . . . 8

1.2.4 The Relationship to Previously Proposed Constraints . . . 11

1.3 Overview of the Parsing Model . . . 12

1.3.1 Representing Phrase Structure Trees . . . 12

1.3.2 Recovering Phrase Structure Trees . . . 18

1.3.3 The Connectionist Implementation . . . 24

1.4 Overview of the Evaluation . . . 26

1.4.1 Phenomena of Particular Concern . . . 27

1.4.2 Testing for Adequacy and Significance . . . 28

1.4.3 Advantages of the Architecture . . . 30

1.5 Summary . . . 31

2 The Computational Constraints **33**

2.1 Constraints on Previous Connectionist Parsers . . . 34

2.2 Connectionist Symbolic Computation . . . 38

2.2.1 Temporal Synchrony Variable Binding . . . 38

2.2.2 Other Characteristics of the S&A Architecture 41
2.2.3 Other Models of Connectionist Symbolic Computation 42
2.2.4 Limitations of the S&A Architecture 43
2.3 Constraints on the Parsing Model 46

3 The Grammatical Framework **49**
3.1 Related Approaches to Grammatical Representation 50
3.1.1 Requirements for the Grammatical Framework 50
3.1.2 Specifying Independent Information Independently 52
3.1.3 Computational Locality 54
3.2 Structure Unification Grammar 55
3.2.1 Describing Phrase Structure 56
3.2.2 Accumulating Phrase Structure 60
3.2.3 A Formal Specification of SUG 65
3.3 Expressing Grammatical Information in SUG 66
3.3.1 Using SUG's Large Domain of Locality 68
3.3.2 Trading Ambiguity for Partial Specification 71
3.3.3 Partial Descriptions of Phrase Structure in Other Formalisms 73
3.4 Forgetting Grammatical Information in SUG 75

4 The Parsing Model **79**
4.1 Related Approaches to Syntactic Parsing 80
4.1.1 Requirements for the Parsing Model 80
4.1.2 Inferring New Information 83
4.1.3 Modeling Memory Bounds 84
4.2 NNEP's Grammars 85
4.2.1 Restrictions on the Grammars 86
4.2.2 Restrictions on the Derivations 87
4.3 NNEP's Operations 88
4.3.1 Combination Operations 89
4.3.2 State Internal Operations 92
4.3.3 The Forgetting Operation 93
4.4 NNEP's State Information 94
4.4.1 The Representation of State Information 95
4.4.2 Maintaining State Information 98

4.5 NNEP's Disambiguation Mechanism . . . 101
4.5.1 Coordinating Disambiguation Decisions . . . 101
4.5.2 Choosing an Action . . . 102
4.5.3 Estimating Choice Probabilities . . . 106
4.6 NNEP's Output . . . 110

5 The Connectionist Implementation 111
5.1 Network Structure . . . 112
5.2 The Connectionist Architecture . . . 114
5.2.1 Units and Links . . . 114
5.2.2 Implementing Predicates . . . 115
5.2.3 Implementing Pattern-Action Rules . . . 116
5.3 Implementing NNEP's Operations and Grammar . . . 117
5.3.1 Combination Operation Rule Patterns . . . 118
5.3.2 Combination Operation Rule Actions . . . 121
5.3.3 State Internal Operation Rules . . . 123
5.3.4 The Forgetting Rule . . . 124
5.4 Maintaining the Parser State . . . 125
5.5 The Time Course of Parsing . . . 126
5.6 Characteristics of the Implementation . . . 127
5.7 The Simulation of the Implementation . . . 128

6 Testing the Parsing Model 130
6.1 Phrase Structure Analyses . . . 133
6.2 Recovering Long Distance Dependencies . . . 136
6.3 Representing Local Ambiguities . . . 148
6.4 Handling the Parser's Resource Bounds . . . 161
6.5 The Diversity of Language . . . 168

7 Conclusion 170
7.1 The Adequacy and Significance of the Architecture . . . 171
7.2 Summary . . . 173
7.3 Discussion . . . 177
7.4 Future Work . . . 178

A Disambiguation Interventions 181

Bibliography 183

Chapter 1

Introduction

Several properties of connectionist networks have made them popular for both cognitive modeling and technological applications. They have effective learning algorithms, can combine multiple soft constraints, perform massively parallel computation, are sometimes biologically motivated, and often exhibit other desirable characteristics such as graceful degradation. However, traditional connectionist architectures are unable to capture the generalizations that arise as a result of the compositional nature of many phenomena (i.e. systematicity) (Fodor and Pylyshyn, 1988). The paradigmatic example of such a phenomena is natural language syntax. Because of this limitation, previous efforts to use connectionist networks for parsing natural language have not made much progress. Fortunately, recent work on extending traditional connectionist architectures to support symbolic computation (Shastri and Ajjanagadde, 1993) has produced a connectionist computational architecture that can directly express generalizations over constituents, thereby capturing systematicity within the connectionist architecture. This dissertation investigates using this connectionist computational architecture for recovering the syntactic structure of natural language sentences. The extension (temporal synchrony variable binding) is shown to be sufficient to make connectionist networks powerful enough for syntactic parsing. This demonstration makes it possible to apply the advantages of connectionist networks to natural language processing. In addition, the architecture is constrained in ways that make significant predictions about the nature of language. Thus the investigation of natural language processing in this computational architecture is not a mere question of implementation.

Like other purely connectionist architectures, Shastri and Ajjanagadde's architecture (1993) uses many simple computing units that communicate with each other using only an output activation value. The pattern of activation over these units represents the predications that are being stored (or their probabilities), and the interconnection pattern (i.e. the links) implements the rules of the system. Since a given link connects a fixed pair of units, the rules of the system are specific to the information represented by different units, but generalize across information represented by different times. As in recurrent connectionist networks, this property can be used to implement rules that are independent of absolute position in the input sequence by presenting the input sequentially in time. For example, the same link can be used to test whether the next word is a noun whether that word is the first or tenth word in the sentence. Unfortunately, other recurrent networks only use time to represent input and computation sequence, so rules do not generalize over any other type of information. This is a problem because rules must be able to generalize over phrase structure constituents in order to capture the compositional aspects of natural language syntax. For example, a rule which computes the effect of an adjective on its noun phrase shouldn't

care if the noun phrase is the subject or the object of the sentence. To capture these generalizations, a network needs to represent constituent identity using time, not space. Shastri and Ajjanagadde (1993) provide exactly such a mechanism, called temporal synchrony variable binding. Course grained temporal distinctions are still used to represent the sequence of inputs and computations, but in this architecture, fine grained temporal distinctions are used to represent variables. Since in the parser variables refer to phrase structure constituents, temporal synchrony variable binding allows the compositional aspects of natural language syntax to be directly expressed.

Shastri and Ajjanagadde (1993) show how this connectionist architecture (henceforth the S&A architecture) can be used to perform the very fast common sense reasoning they call reflexive reasoning. They argue that the S&A architecture is biologically motivated, supports the massively parallel use of knowledge, supports evidential reasoning, has psychologically plausible limitations, and supports symbolic computation. This dissertation investigates how this same architecture can be applied to recovering the syntactic constituent structure of natural language sentences. There are two basic questions to be asked about syntactic parsing in the S&A architecture: are the architecture's abilities adequate to account for what people are able to do, and and do its limitations help explain what people are not able to do. In other words, is the S&A architecture computationally adequate for syntactic parsing, and does it impose linguistically significant constraints? This dissertation provides an affirmative answer to both these questions.

At the current stage of this investigation, the question of computational adequacy is the most important one. For no other connectionist parser has a convincing argument been made that it is powerful enough to handle the complexities of natural language. In addition, a demonstration of computational adequacy is a prerequisite to a demonstration of linguistically significant constraints. If the constraints of the architecture prevent it from being able to parse good sentences, then its inability to parse any bad sentences is not very significant. Thus the primary concern of this document is to show that the S&A architecture is computationally adequate for syntactic parsing. Because of the computational characteristics and biological motivations of this connectionist architecture, such a demonstration is interesting in and of itself.

A demonstration of computational adequacy would ensure that a syntactic parser could be implemented in the S&A architecture, but it would not by itself be informative for higher level theories of language. To demonstrate that connectionist networks can be more than mere implementations, the limitations of the architecture have to be shown to make significant predictions about the nature of language. While the linguistic implications of the constraints imposed by the S&A architecture have not been investigated to the same extent as the issue of the architecture's computational adequacy, some linguistic phenomena can be explained in terms of constraints on the parsing model which are motivated by the limitations of the architecture. This dissertation lays the ground work for more such investigations in the future.

The arguments for the computational adequacy and linguistic significance of the S&A architecture are based on an analysis of the limitations of the architecture. Because temporal synchrony variable binding allows computation in this architecture to be characterized in terms of traditional symbolic computation,[1] the limitations of the architecture can be characterized as a set of computational constraints on symbolic computation. This method prevents the irrelevant details of connectionist networks from interfering with the discussion, and it allows previous work on computationally

[1] By symbolic computation I mean the use of abstract representations of entities, their properties, and the generalizations about these properties, as in, for example, predicate calculus. I do not mean to exclude the use of continuous valued parameters (such as probabilities), or the use of feature decompositions to describe entities (as in distributed connectionist representations).

constrained parsing to be used. Given this set of computational constraints, a parsing model is developed which is designed to comply with these constraints. This parsing model is then tested on its ability to handle the phenomena which are of particular concern given these constraints. Showing how a parser can handle each of these phenomena demonstrates that the limitations of the architecture do not prevent syntactic parsing, and thus that the architecture is computationally adequate for syntactic parsing. To ensure that the analysis of what phenomena to include in these tests is not flawed, a test on an essentially unbiased data set is also analyzed. The results from the phenomena-specific tests also indicate what kinds of sentences the computational constraints prevent the parser from being able to handle. These results show that the limitations of the architecture make significant linguistic predictions.

The remainder of this chapter starts with a discussion of the motivations for this work. This is followed by an overview of the other chapters in this dissertation. Section 2 starts with a brief discussion of the limitations of previous connectionist architectures. Then the S&A architecture is described, and the constraints which it imposes on the parser are characterized. These constraints are combined with constraints on the parser's input and output to form the basic requirements for the parsing model, which are then compared to previously proposed computational constraints on natural language. In section 3 the parsing model is described. First the parser's grammatical framework is given, then the parser's design, and then the connectionist implementation is briefly described. Section 4 outlines the argument that this parser demonstrates the adequacy and linguistic significance of the S&A architecture for syntactic parsing. This argument involves testing the parser on data pertaining to expressing phrase structure analyses, recovering long distance dependencies, representing local ambiguities, staying within the parser's resource bounds, and the diversity of phenomena in language. The last section summarizes this chapter and gives an outline of the rest of this dissertation.

1.1 Motivations

There are two types of motivations for the work done in this dissertation. One is that this work uses independently motivated architecture-level constraints to make predictions about the nature of language. The difficulty in justifying one set of computational constraints over another has limited the significance of investigations of computationally constrained parsing. The other is that this work answers technological questions about combining connectionist architectures with symbolic theories of sentence processing. Previous investigations of connectionist parsing have typically ignored, or even denied the significance of, the substantial body of existing work on natural language.

1.1.1 Investigating Architecture-Level Constraints

While most investigations of natural language have concentrated on the nature of grammars, there are undoubtedly some characteristics of language which are best described in terms of the computations which people perform to process language. Unfortunately, the nature of these computations is relatively unknown. This is largely because we have known so little about the nature of the computational device (namely the brain) on which these computations are performed. Without making assumptions about the nature of this device, any investigation of the computational constraints on language is limited to using asymptotic complexity arguments. Asymptotic complexity measures are too course grained to be adequate for investigating these constraints. Church (1980) showed

that natural language[2] is finite state. This is the strongest claim one can make using asymptotic complexity arguments, and it still does not make any predictions about specific natural language sentences. Thus any further progress requires making assumptions about the nature of the computational mechanisms our brains use to process language. While such assumptions have been investigated, it is difficult to decide which of the large variety of possible constraints to investigate. Also, when a linguistically significant computational constraint is found, it is difficult to argue that the constraint is due to the limitations of the human language processor, and not due to other factors.

This dissertation makes a very specific assumption about the computational device used in processing language, namely that it functions in the manner described in the Shastri and Ajjanagadde (1993) connectionist computational architecture. In addition to supporting symbolic computation, this architecture is computationally constrained. It is these constraints which form the basis of this investigation of the computational constraints on language. While the constraints themselves could be assumed independent of this particular architecture, the architecture provides a basis for selecting one set of constraints over another. Because there are a number of motivations for this architecture which are independent of the linguistic consequences of its constraints, the linguistic consequences can be said to be explained by these computational constraints. Without such independent motivations, computational constraints become just another mechanism for describing linguistic data.

Investigating independently motivated architecture-level constraints is not only important for the study of the nature of language, it is also important for natural language processing applications. People are extremely fast and accurate in recovering the syntactic structure of sentences, even if the sentence's conceptual content is difficult to understand. Currently, the properties of natural language which make this efficiency and accuracy possible are not well understood. While it is theoretically possible to discover these properties using only considerations of current artificial computation technology and asymptotic complexity measures, such efforts have not been fully successful to date. Because natural languages have undoubtedly evolved (genetically or historically) to allow them to be efficiently and accurately processed by our brains, a promising alternative approach is to use biological considerations to investigate these properties. This is the approach taken here. The use of biological considerations in this investigation does not limit the applicability of its results to biologically motivated computational architectures. Once we have discovered the properties of natural language which make efficient and accurate processing possible, these insights can be applied to natural language processing applications implemented in any computational architecture.[3]

1.1.2 Combining Connectionism with Linguistic Theories

The kinds of problems which connectionist models have been successful with are largely orthogonal to the kinds of problems which symbolic theories of language have been successful with. Connectionist architectures have been successful at modeling processes which involve learning, and combining, multiple sources of soft constraints. Linguistic theories have been successful at modeling phenomena which involve manipulating complex, discrete representations. Processing natural language involves both these types of problems. Recently the natural language processing (NLP)

[2] By "natural language" I mean pretheoretic observable communication via language. I do not mean the competence abstraction of language which is prevalent in most linguistic work.

[3] The argument made in this paragraph is analogous to one made by Shastri to motivate the investigation of reflexive reasoning in the S&A architecture (personal communication).

community has been very interested in statistical models, precisely because the hardest remaining issues in language processing are those which have not been addressed with symbolic methods. Most connectionist researchers have not come to the reciprocal realization. The lack of work combining connectionist methods with traditional NLP methods has prevented research on NLP from making use of the strengths of connectionism. The work presented in this dissertation lays the groundwork for making connectionist methods available for NLP researchers.

The work in this dissertation concentrates on the those aspects of syntactic parsing which other connectionist approaches have had trouble with. Because of this, this investigation does not take full advantage of the abilities of connectionist networks. It does, however, make it possible to take advantage of these abilities in future work. In particular, the ability of connectionist networks to learn how to combine multiple sources of soft constraints would be very useful in both grammar induction and disambiguation. Grammar induction can be done by starting with a broad class of grammar entries, and removing or keeping them based on their usefulness in parsing a training set. Connectionist learning algorithms do exactly this, except they decrease or increase weights rather than removing or keeping grammar entries. Decreasing the weight of a grammar entry to zero removes it from the grammar. Such training also subsumes the parameter estimation needed to do disambiguation. Thus both grammar induction and disambiguation can be done by applying connectionist learning techniques to train that portion of the network which implements the grammar. More discussion of this future work is given in section 7.4.

1.2 Overview of the Computational Constraints

As discussed above, this dissertation argues for the computational adequacy and linguistic significance of the Shastri and Ajjanagadde connectionist computational architecture. While these arguments involve the development of a specific parsing model which is implemented in the primitive computational devices of the architecture, that level of description is too detailed to provide a useful basis for discussing the motivations for, and implications of, many aspects of the parser's design. Fortunately, computation in this architecture can be characterized in terms of symbolic computation, which has been found to be an appropriate level of abstraction for this type of investigation. This prevents the irrelevant details of the architecture from interfering with the investigation of parsing issues.[4] However, in order to do the investigation at the level of symbolic computation, the characteristics of the architecture which *are* relevant for parsing need to be characterized at that level. These characteristics form a set of constraints on symbolic computation. These computational constraints, plus the constraints from the nature of the parsing task, form the set of computational constraints on syntactic parsing which will be investigated in this dissertation.

This section is an overview of the discussion in chapter 2. It outlines the computational constraints on the syntactic constituent structure parsing model. The discussion starts with a brief characterization of the computational constraints imposed by the architectures used in other investigations of connectionist parsing, and what empirical investigations would be needed to argue that these constraints do not prevent such parsers from ever being adequate. Then a description of how the S&A architecture supports symbolic computation is given. This mechanism has some limitations which constitute the primary constraints on the parsing model developed in this dissertation. These constraints are characterized and then combined with constraints on the parser's input and output.

[4]This approach represents a departure from standard connectionist methodology. Even if the reader is not convinced by the above argument, hopefully they will find the results of this investigation sufficient to justify this divergence.

The resulting set of constraints require that the parser use bounded memory, not use disjunction, limit the storage and processing of relations, produce incrementally interpretable output, and parse in quasi-real time. The feasibility and implications of parsing within this set of constraints are the central concern of this dissertation. This section concludes with a discussion of how these constraints relate to previously proposed computational constraints on natural language.

1.2.1 Constraints on Previous Connectionist Parsers

While it is unreasonable to expect an argument for the adequacy of a computational architecture to include a complete natural language parser implemented in that architecture, it is reasonable to expect such an argument to address those parsing issues which are of particular concern given the constraints imposed by the architecture. Unfortunately, previous investigations of connectionist parsing have not met this criteria. All these investigations use connectionist architectures with two major limitations: they are finite state, and they can not factor the representation of constituent identity from the representation of constituent features.

The first limitation requires that a parser be able to parse in bounded memory. While there are several ways to do this, it is not clear how it could be done for the representations used in these parsers. In particular, since the representations do not distinguish constituent identity information from constituent feature information, they can not take advantage of the fact that the set of features needed during a parse is fixed, while the set of constituents needs to be dynamically determined. Since these parsers have not been tested with sentences which are too long for the whole analysis to be represented in the network, we don't have any evidence that such sentences could be parsed effectively.

The constraint that the representation of constituent identity can not be factored from the representation of constituent features is a more serious challenge than the previous one.[5] There are two implications of this constraint. First, conflating these two dimensions of information results in a very large number of primitive features. While it is theoretically possible for a network to support this many primitive features, none of the previous connectionist parsers have been tested on a sufficiently diverse or complex set of sentences to determine whether it is tractable. The second implication of not being able to factor features from constituents is that constraints or rules which are learned for a pattern of features for one constituent do not generalize to the same pattern of features for another constituent. Some researchers claim this as an advantage, because these networks always take the "context" of each constituent into consideration. However, it is the difficulty these networks have with abstracting away from irrelevant context information which prevents them from taking advantage of the compositional aspects of natural language. This inability to generalize in the optimal way also means that they must be trained using a very large and particularly well balanced set of sentences. Perhaps there are solutions to this problem, but until connectionist parsers implemented in architectures with this limitation have been tested on a large enough and complex enough set of sentences, there is little reason to believe that they can be adequate for parsing natural language. A discussion of specific parsing models is given in section 2.1.

[5] This constraint is due to the lack of a mechanism for doing dynamic variable binding. This process will be discussed in the next section.

1.2.2 The Connectionist Architecture

The Shastri and Ajjanagadde connectionist computational architecture has several characteristics which make it well suited for investigating natural language parsing. As argued in (Shastri and Ajjanagadde, 1993), the architecture is biologically motivated, supports the massively parallel use of knowledge, supports evidential reasoning, has psychologically plausible limitations, and supports symbolic computation. For our purposes the last of these properties is the most important. It is this ability which allows the representation of constituency to be factored from the representation of features.

To support symbolic computation, it must be possible to represent, and compute with, multiple properties of multiple things. Such a representation must have a mechanism for distinguishing which properties are for which things. This is called the variable binding problem. For example, to represent the situation on the top left of figure 1.1, we need to represent that the square is striped and the triangle is spotted. This can be done with the following logical formula.

$$\exists x, \exists y, \mathit{striped}(x) \wedge \mathit{square}(x) \wedge \mathit{spotted}(y) \wedge \mathit{triangle}(y)$$

In this formula, the variables are used to represent the bindings between predications. The name "x" does not mean anything in and of itself, but the sharing of it represents that the thing which is striped is the same as the thing which is square, and possibly different from the thing which is spotted and a triangle. This information can be represented in the S&A architecture as shown in the rest of figure 1.1. As in many connectionist architectures, different predicates are represented with different units.[6] The pattern of activation over these units represents the predications which are true (or the probability of their truth). The problem with this simple representation is that there is no representation of which predicates are true of which thing. To represent the depicted situation, all four units would have to be active, but then we would not know whether it is the square or the triangle which is striped. The S&A architecture solves this problem by using units which, rather than producing sustained output, produce a pulse train of activation.[7] If two units are pulsing synchronously, then they are representing predications about the same thing, and if they are not pulsing synchronously, then they are representing predications about possibly different things. Thus the temporal synchrony of unit activation is used to represent the bindings between predications, just as variables are used to do this in logical formulae. This mechanism is called temporal synchrony variable binding, and it is the core feature of the S&A architecture. For the purposes of this investigation I will be assuming that these units all fire at the same frequency, so temporal synchrony reduces to having the same phase in the periodic pattern of activation. These phases, then, are equivalent to variables, as shown in figure 1.1. In the parsing model discussed below, variables refer to phrase structure constituents, so these phases represent constituent identities.

As in other connectionist architectures, computation in the S&A architecture is done using links between units. Links multiply the output value of their input unit by their weight to get their activation. Some links provide this activation as input to another unit, where it is summed with the activation from other input links. The S&A architecture also allows links which use their activation to inhibit the activation of another link. A primary link's activation is multiplied by one minus the activation of each inhibiting link.[8] Sets of interconnected links are used to implement

[6] To prevent confusion, I will refer to nodes in a connectionist network as "units", and nodes in a phrase structure tree as "nodes".

[7] There are other kinds of units which do produce sustained output. These units represent predications about the situation as a whole, rather than information about individual entities.

[8] The use of inhibitory links is necessary to implement signal gates that don't introduce significant propagation delays, and to allow dynamically calculated probabilities to be multiplied.

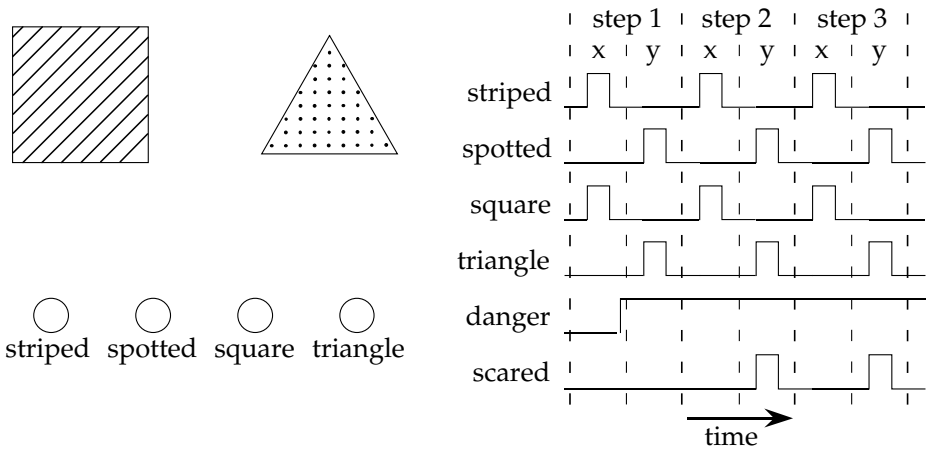


Figure 1.1: An example of temporal synchrony variable binding, and the sequential application of two rules.

pattern-action rules, which test and modify the predications stored in the temporal pattern of activation of the units. Because the same links are present during each variable's phase, these pattern-action rules inherently generalize across variables. Thus the rules of the parser inherently generalize across phrase structure constituents. It is this ability to capture generalizations which distinguishes the S&A architecture from other solutions to the variable binding problem, such as (Smolensky, 1990).

An example of computation in the S&A architecture is also given in figure 1.1. The bottom two lines of the timing diagram show the effects of the application of the following two rules.

$$\forall x, \mathit{striped}(x) \wedge \mathit{square}(x) \Rightarrow \mathit{danger}$$
$$\forall x, \mathit{danger} \wedge \mathit{triangle}(x) \Rightarrow \mathit{scared}(x)$$

First the synchronous activation of *striped* and *square* trigger the activation of the unit representing the predicate *danger*. *Danger* represents a property of the situation as a whole, and therefore it's unit is active during both x and y's phase. Then in y's phase the simultaneous activation of *danger* and *triangle* cause the *scared* unit to change its state. This change shows up in the output activation of the *scared* unit in the subsequent period, in the phase of y. Because the effects of a rule generally show up in the pattern of activation one period after the rule applies, periods can be thought of as steps in the computation, as shown in figure 1.1. By supporting predications over variables and supporting sequential computation with pattern-action rules that generalize over variables, the S&A architecture supports the kind of symbolic computation necessary for syntactic parsing. For more discussion of how the S&A architecture supports symbolic computation, see section 2.2.

1.2.3 Constraints on Syntactic Parsing

While the S&A architecture provides a rather general purpose computing framework, it does have significant limitations. One very significant limitation of this architecture is that it has a bounded memory capacity.[9] The ability to detect whether or not units are pulsing synchronously has

[9]The architecture allows for multiple computing modules, each with its own memory. NNEP is implemented as one of these modules. The memory bounds apply to each computing module independently, so the parser does not have to share its memory resources with other cognitive activities.

bounded precision, and units can only maintain periodic firing for a bounded range of frequencies, so only a bounded number of distinguishable phases can fit in one period. Thus predications can only be stored for a bounded number of variables. From biological evidence this bound is at most ten, probably a little less (Shastri and Ajjanagadde, 1993). For this investigation the bound will be assumed to be ten.[10]

Another significant limitation of the S&A architecture is that it has no explicit representation of logical connectives. Thus only the default logical connective can be used. Conjunction is the most useful connective for syntactic parsing, so it will be used as the default connective. This means the architecture cannot explicitly represent a disjunction of predications. However, it can implicitly represent disjunctive information, and disjunction can be manifested in parallel computations. The rules of the network or an external observer can give a disjunctive interpretation to a predicate, but in terms of the explicit representation, the predications will still simply be conjoined with other predications. Also, the lack of a predication can be interpreted as a disjunction between that predication and other incompatible predications, but again the predications which are specified are treated as conjoined. The parallel computation of pattern-action rules can also manifest disjunction, in that different patterns can be doing tests which pertain to different possible continuations of the computation. Thus if a bounded amount of disjunction is needed for a bounded amount of time, it can be eliminated by compiling all that computation into one pattern-action rule for each of the disjuncts. These computations, however, must be atomic, in that they cannot store intermediate state, and thus cannot be composed of multiple steps. This limits the feasibility of this method to only short computations, although there can be a large number of disjuncts. As will be discussed below, the inability to explicitly represent disjunction means a parser must be deterministic, as Marcus (1980) proposed.

The remaining limitations of the S&A architecture are due to the fact that temporal synchrony variable binding only adds one additional dimension for representing information. This dimension (fine grained temporal synchrony, or phase) allows the efficient storing and processing of information about individual variables, but relations between variables require additional representational dimensions for each additional argument position in the relation. This is analogous to the situation which traditional connectionist architectures are in when they try to represent unary predications. The solution to the need for additional representational dimensions is to use the dimensions which are available more than once. Traditional connectionist architectures (implicitly or explicitly) use the space dimension (i.e. unit identities) more than once. This technique was formalized by Smolensky in his tensor product variable binding mechanism (Smolensky, 1990). This is also the technique used in the S&A architecture for representing relations between variables. This means that the S&A architecture has the same problems with storing and processing relations between variables as traditional architectures have with storing and processing all predications over variables. These problems are characterized in two additional constraints imposed by the S&A architecture, one on storing relations, and one on processing relations.

The problem with storing relationships between variables using multiple spatial dimensions is that it requires more units and it adds complexity in modifying and accessing the stored relations. To control these costs, Shastri and Ajjanagadde (1993) propose that at most three instantiations of a relation can be stored at any one time. This adds one additional dimension with three distinct values. This dimension represents bindings between sets of variables, where each variable in each

[10]For the data addressed and representations used in this dissertation, this bound could be reduced to nine. This is interesting because nine is the maximum of the robust bound on human short term memory of seven plus or minus two (Miller, 1956). However, some additional data needs to be found and used for testing before such a precise claim is made.

set is related to each variable in the other sets. The constraint requires that no more than three such bindings be stored at any one time. For example, $r(a,b) \wedge r(c,d) \wedge r(e,f) \wedge r(g,h)$ could not be stored, but $r(a,b) \wedge r(a,d) \wedge r(e,f) \wedge r(g,h)$ could, because the first two relationships can be represented with a single binding between the sets $\{a\}$ and $\{b,d\}$. In the general case, binary relations would require ten such bindings, since ten is the maximum number of variables in the memory.

It is possible to implement rules that manipulate relationships between variables using one copy of the rule for each of the three bindings for the relation, but this would require an additional mechanism to be added to the architecture. Because rules (unlike storage) have adjustable parameters, and connectionist learning algorithms are spatially local, implementing a rule in multiple copies would require a mechanism for ensuring that the weights of the links in the different copies are the same. While such weight sharing is easy to do from outside the the architecture, currently there is no biologically motivated way to do it within the architecture. Thus a single rule needs to be implemented with a single pattern of links. It is possible to time-multiplex these links across a relationship's bindings (as is done for variables), but such rules would then take up to three times as long to apply, and this would require circuitry to sequence through the bindings. If we don't allow this method either, then rules cannot directly manipulate relationships between variables. Information about variables is available one variable at a time, and rules are implemented with links, which have no memory, so rules must apply to each variable independently. Thus relations must be set and accessed via unary predicates. For example, calculating long distance dependencies requires information about what phrase structure nodes are possible "gap" sites for "trace" nodes. Because (as it turns out) only one trace node needs to be involved in this calculation at any given time, this information can be accessed using one unary predicate identifying this unique trace node, and one unary predicate specifying what nodes might be the gap for the unique trace node. The calculation itself must be done using these unary predicates. Calculations, such as this one, which involve multiple variables can be done despite the fact that individual rules must apply to each variable independently. In addition to information about variables, the network's activation pattern represents information about the situation as a whole. Calculations which involve multiple variables can be done using different rules to set and test predications about the situation as a whole. This was illustrated in figure 1.1, where *danger* was set based on information about x, and then tested to determine information about y. Because communicating information between variables using constant predicates does not allow the identity of a variable to be communicated, this technique still does not permit computations which require the manipulation of pairs of variables. This requirement that rules apply to each variable independently is a form of locality constraint on computations. While this constraint has no precedent in work on syntactic parsing, it turns out to have a number of significant linguistic implications.

A model of syntactic parsing in the S&A architecture needs to comply both with the constraints from the architecture, and with requirements from the nature of the parsing task. The later constraints are imposed by the environment which provides the input to the parser, and by the modules which interpret the output of the parser. The words which are input to the parser become available one at a time, in the order in which they appear in the sentence. Thus the parser must accept incremental input. For spoken language, and to a large degree in reading, the time between the input of each word is bounded. Thus the parser must only take a bounded amount of time per word. In other words, the parser must parse in quasi-real time.[11]

[11] Because of the relationship between the architecture and biological mechanisms, biological evidence can be used to determine whether the parser can parse in real time, not just quasi-real time. Such a claim requires estimates of parsing time in terms of actual seconds, not just computation steps. The link between computation steps and seconds

The modules which receive the output of the parser need to compute the sentence's interpretation incrementally. In order to provide for incremental interpretation, the parser's output must be incremental and monotonic. If the output isn't monotonic, then the interpreter can't make commitments on the basis of the output without risking having to retract those commitments. While such retractions do occur under some circumstances, I assume that there is always some evidence that something has gone wrong. Typically the person will be consciously aware of a problem, although other evidence (such as regressions in eye movements) can also be used to determine these cases. Since we are concerned here with the normal case in which nothing goes wrong, the parser's output must be monotonic.

Combining the externally imposed constraints with the constraints from the architecture, we get the following complete list of the constraints on a model of syntactic parsing in the S&A architecture. The constraints from the architecture (1–4) are discussed in more detail in section 2.2.4, and the externally imposed constraints (5–8) are discussed in more detail in section 2.3. As will be discussed in the next section, constraints 2, 7, and 8 mean the parser must be deterministic (in the sense of (Marcus, 1980)), and in subsequent discussion they will often be referred to as the determinism constraint.

1. at most ten variables stored at a time
2. no explicit representation of disjunction
3. at most three instantiations of relations
4. rules apply to each variable independently
5. incremental input
6. quasi-real time
7. incremental output
8. monotonic output

1.2.4 The Relationship to Previously Proposed Constraints

In additional to being consequences of using a particular independently motivated computational architecture, these computational constraints are interesting because of their relation to previously proposed computational constraints on natural language. The first constraint is an example of a bounded memory requirement. It has generally been assumed that at some level of abstraction the syntactic parser has a bounded memory (Chomsky, 1959). Church (1980) showed that this constraint applies at a level which takes into consideration performance constraints, such as restrictions on the depth of center embedding and on the availability of phrases for posthead modification. The particular form of the bounded memory constraint given above has not previously been successfully applied to syntactic parsing, but it has extensive precedence in other investigations of cognition. Miller (1956) proposed a bound of seven plus or minus two on the number of things which can be stored in short term memory, and this result has been replicated for a surprising number of tasks. The bound given above is precisely the same form of constraint, and although here I'm assuming ten things can be stored, Miller's results are within the resolution of the biological arguments which

is provided by the theory of how the primitives of the architecture are realized in neurons. This kind of extremely fine grained prediction makes the potential of working in this architecture great. The parser presented here complies with this real time constraint, but since the estimates of real speed are rather rough, I will concentrate on the clearer constraint of quasi-real time.

are used to derive that bound. See (Shastri and Ajjanagadde, 1993) for a more extensive discussion of this relationship.

Another interesting correlation with previously proposed computational constraints on natural language is due to the restriction on disjunction and the requirement for incremental monotonic output. These constraints imply that the syntactic parser must parse deterministically. This constraint was first proposed by Marcus (1980), and has been argued for by several researchers since ((Church, 1980), (Marcus *et al.*, 1983), (Berwick and Weinberg, 1984)). It requires that the parser deterministically pursue a single analysis. This means that multiple analyses can't be pursued in parallel, and that once the parser commits to an aspect of the analysis it can't retract that commitment. Explicitly pursuing multiple analyses in parallel is equivalent to having explicit disjunction in the representation of the analysis, which is ruled out by the second constraint above. The retraction of commitments is ruled out because all commitments must be immediately output in order for the output to be maximally incremental, and once information has been output it can't be retracted or the output would not be monotonic. Thus the determinism constraint can be derived from the independently motivated constraints that there be no explicit representation of disjunction and that the parser's output be incremental and monotonic.

1.3 Overview of the Parsing Model

The previous section identified the characteristics of the Shastri and Ajjanagadde connectionist architecture which are significant for syntactic parsing in terms of a set of constraints on symbolic computation. Given this characterization, it is possible to make the argument for the adequacy and significance of this architecture at the level of symbolic computation. This greatly simplifies the discussion of the relevant parsing issues, and it allows results from work in linguistics, computational linguistics, and psycholinguistics (which has almost all been done in terms of symbolic representations) to be applied to this investigation. Because it is the computational constraints that are relevant for determining adequacy and significance, the need to comply with these constraints determines what parsing issues are the focus of this investigation.

The argument for the adequacy and linguistic significance of the S&A architecture is made using a specific example of a parser implemented in this architecture. This section gives an overview of this parsing model, called a Neural-network Node Equating Parser (NNEP). This material is discussed in detail in chapters 3 through 5. The above constraints on the parser require that NNEP's representation of phrase structure trees have certain characteristics. A grammatical framework which has these characteristics is described in the first subsection. Then the design of NNEP itself is outlined. NNEP computes derivations from the grammar formalism, but the above constraints limit what derivations can be computed. The connectionist implementation of this parsing model is briefly characterized in the fourth subsection.

1.3.1 Representing Phrase Structure Trees

The constraints outlined in the previous section place several requirements on the parser's representation of grammatical information.[12] First, because the parser must be deterministic, the

[12]Since the constraints being investigated are computational in nature, the grammatical representation used here is designed primarily to support the needs of the parser. While it would be nice to have a single level of representation for investigating both grammar specifications and parser representations, finding a representation that merges the

representation should allow the parser to avoid saying what it doesn't know. Following Description Theory (Marcus *et al.*, 1983), partial descriptions of phrase structure trees are used to satisfy this requirement. Partial descriptions allow the parser to underspecify phrase structure information, rather than either overcommitting or using a disjunction of more completely specified alternatives. In addition, in order to produce incremental output and only allow syntactically well-formed analyses, the parser must be able to say what it does know. Again the use of partial descriptions is important for this requirement, because they allow different kinds of information to be specified independently of each other. To satisfy both these requirements, the grammatical representation must allow information which the parser does know at a given time to be specified independently of the information which the parser does not know. The grammatical representation used here allows different kinds of grammatical features (e.g. +*nominative*, +*plural*), expectations (e.g. obligatory arguments), iteration restrictions (e.g. one determiner per NP), and structural constraints (e.g. linear order) to all be specified independently of each other.

The locality constraint on rules (constraint number 4 above) and the parser's bounded memory both place another requirement on the parser's representation of grammatical information. Because of the locality constraint on rules, the representation should allow as much information as possible to be local to individual phrase structure nodes. This helps avoid the need to implement computations with multiple rules that set and test predications about the situation as a whole. Thus we want a relatively flat phrase structure representation, provided it still expresses the compositional nature of syntax. This compact representation also makes it easier to stay within the parser's bounded memory, because it reduces the number of nodes in a tree's representation. The grammatical representation used here allows flexibility in the grouping of information into nodes because multiple kinds of expectations and iteration restrictions can be specified for a single node. In many formalisms this is not true. For example, in Context Free Grammars, the node on the left side of a rule cannot have any more nodes attached to it (thereby restricting iteration), and the nodes on the right side of the rule must have other nodes attached to them (thereby expressing expectations). For constituents which can iterate, like optional modifiers, Chomsky adjunction needs to be used. This results in multiple copies of the modified node. Also, in order to control the iteration of words such as determiners separately from controlling the iteration of, for example, head nouns, Context Free Grammars have to have separate nodes for these two purposes (i.e. NP and N, or DP and NP). These problems also apply to the expression of expectations. Optional arguments require two grammar rules, one with the argument and one without, and expressing the expectation for a determiner separately from expressing the expectation for a head noun requires two separate nodes for these purposes.

The locality constraint on rules and the parser's bounded memory interact in another interesting way to constrain the parser's representations. Not only should as much information as possible be local to individual nodes, as little information as possible should be expressed as relationships between nodes. This minimization also helps compliance with the constraint on the number of instantiations of a relation that can be stored. Of the four kinds of information mentioned above, only structural constraints involve multiple nodes. By allowing most ordering constraints to be stated with respect to terminals (rather than other nonterminals), many structural constraints can also be localized to individual nodes.[13] By identifying the minimal set of relations that are needed to parse, special mechanisms which avoid rules that must manipulate pairs of nodes can be devised

requirements of these two activities must be left for later work. Thus the representation discussed here is not being proposed as a replacement for those used in investigations of the nature of grammar.

[13]Only nonterminal nodes are represented as entities in the parser's memory. Information about terminals is represented with features and constraints on the use of grammar entries.

for these few cases. This localization of computation in turn makes it possible to stay within the parser's bounded memory. Because computations which do not directly involve a node are independent of the information about that node, a node which will not be directly involved in any more parser operations can be safely removed from the parser's memory. By removing nodes as they are completed during a parse, the parser can parse arbitrarily long sentences using only a bounded number of nodes at any given time. The grammatical representation used here allow relationships between nodes to be minimized because structural constraints are specified independently of other kinds of grammatical information. Grammar formalisms based on Context Free Grammars do not have this property because expectations and iteration restrictions are specified in terms of a node's structural position in the grammar rule, as discussed above.

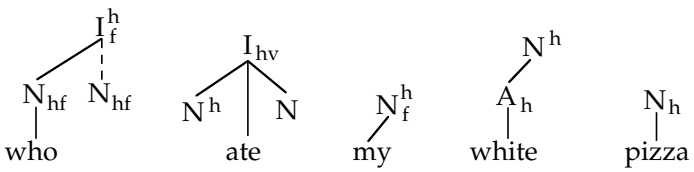
Structure Unification Grammar

In order to comply with the above requirements, NNEP uses Structure Unification Grammar (Henderson, 1990) as its grammar formalism. Structure Unification Grammar (SUG) is a formalization of accumulating partial information about the phrase structure of a sentence until a complete description of the sentence's phrase structure tree is constructed. As such it is similar to other unification-based or constraint-based grammar formalisms. These include Description Theory (Marcus *et al.*, 1983), Head-driven Phrase Structure Grammar (Pollard and Sag, 1987), Construction Grammar (Fillmore *et al.*, 1988), and Segment Grammar (de Smedt and Kempen, 1991), among others. Like these other formalisms, SUG allows multiple kinds of grammatical features to be specified independently of each other. Unlike these other formalisms, SUG allows multiple kinds of expectations, iteration restrictions, and structural constraints to also be specified independently of each other. In addition, SUG's derivations are only constrained by the semantics of the declarative representation, so any valid parsing strategy can be characterized in terms of valid SUG derivations.

The flexibility of SUG derivations is due to its simple mechanism for combining partial descriptions of phrase structure trees. An SUG derivation takes partial descriptions from the grammar (which is simply a set of partial descriptions), conjoins them, and equates some of their nonterminal nodes. Any order of conjoining descriptions and equating nodes is possible, so the parser can use any parsing strategy and still be following an SUG derivation. The only restrictions on derivations are that the final description be consistent and completely describe some phrase structure tree. This means that each equation done in the derivation needs to be between nodes which have consistent descriptions. The grammar can limit the possible equations by specifying inconsistent information about any two nodes which shouldn't be equated. Unlike consistency, completeness is only necessary for the final description. By not satisfying completeness requirements locally, a grammar entry can express expectations about what kinds of information other grammar entries will contribute to the final phrase structure. Because of the complete flexibility of SUG derivations, SUG grammar entries have no procedural import, and the grammar is free to group information into grammar entries in a way which expresses exactly the information interdependencies which the parser needs to know.[14]

The language which SUG provides for specifying partial descriptions of phrase structure trees is illustrated in figure 1.2. As in many formalisms, the grammatical features of nodes are described

[14]The computational constraints on the parser restrict the set of derivations which can be computed. Thus grammar entries do have procedural import when they are interpreted by the parser. At later stages of this investigation it would be good to express this different interpretation in the semantics of the grammar formalism, but here the primary concern is identifying the nature of this difference. Thus flexibility is the desired property for a formalism for this investigation.



key:

- x | y — x **immediately dominates** y
- x ⋮ y — x **dominates** y
- x --➔ y — x **precedes** y
- x^a — x **needs** an a
- x_a — x **has** its a

node label features:

- N = [I:**-**,A:**-**] (noun phrase)
- I = [I:**+**,A:**-**] (sentence)
- A = [I:**-**,A:**+**] (adjective, adverb)
- P = [I:**+**,A:**+**] (prepositional phrase)

superscript and subscript features:

- h = **constituent_head** (noun, inflection, adjective, preposition)
- f = **functional_head** (determiner, complementizer)
- v = **verb**

Figure 1.2: Some example grammar entries. They can be combined to derive the sentence "Who ate my white pizza" by equating the two I's, the second and third N's, and the last four N's.

with feature structures. The use of feature structures allows multiple kinds of grammatical features to be specified independently of each other. For example, "know" can either take a noun phrase or a sentence as its object. Rather than giving "know" a different grammar entry for each case, the entry for "know" can specify its object as $-A$ (noun or sentence) but not specify the value of the I feature (sentence or preposition, versus noun, adjective, or adverb). Expectations and iteration restrictions are specified with a different kind of feature, shown in figures as letter superscripts and subscripts, respectively. Expectations express what information will be specified before the parse is finished. Superscripts specify these expectations in that before a parse can be finished, any node with a superscript must equate with a node that has the same letter as a subscript. For example in figure 1.2, the subject node for "ate" must be equated with a node which has its head noun, thereby expressing the fact that "ate" obligatory subcategorizes for a subject. The object node has no such feature, since the object of "ate" is optional. Iteration restrictions prevent grammar entries from being repeatedly attached at a node, even if the grammatical features of the grammar entries are compatible. Subscripts specify these restrictions in that any node with a subscript cannot be equated with another node which has the same subscript. For example in figure 1.2, "my" has a subscript to prevent other determiners from attaching to the same noun phrase, while "white" has no such subscript, thereby allowing adjectives to iterate.

Like all the above features, structural constraints can be specified partially and independently of other constraints. In addition to the immediate dominance relation for specifying parent-child relationships[15] and the linear precedence relation for specifying ordering constraints,[16] SUG allows chains of immediate dominance relationships to be partially specified using the dominance relation.

[15] In the grammar, the solid lines in figures represent immediate dominance, but when these descriptions are interpreted by NNEP, solid lines do not specify the actual identity of the immediate parent for the dominated node. The reason is that the forgetting operation to be discussed below does not allow such identity information to be kept.

[16] In order to simplify figures, linear precedence constraints will not in general be shown. Most such constraints are between words and either nonterminals or their head terminals. These can be inferred from the lateral position of the nodes relative to the words.

A dominance constraint between two nodes specifies that there must be a chain of zero or more immediate dominance relationships between the two nodes, but it does not say anything about the chain. This relation is necessary to express long distance dependencies in a single grammar entry. For example in figure 1.2, the grammar entry for "who" expresses the fact that its gap is somewhere within its sentence, but does not say where. Because the final description of a derivation must specify a single tree, the N "trace" node in this grammar entry must find a "gap" node to equate with, thereby expressing the fact that the existence of a gap is obligatory.

SUG's ability to describe phrase structure using a minimal number of nodes is due to the above mechanisms for expressing expectations and controlling iteration. Because more than one feature can be used in these mechanisms, more than one type of expectation or iteration restriction can be expressed for a single node. For example in figure 1.2, the N node for "my" can't equate with other determiner nodes but can equate with noun nodes, while the node for "pizza" allows the opposite possibilities. This is not possible when iteration is controlled using structural configuration, as is done in practically all other formalisms. As was discussed for Context Free Grammars above, using structural configuration requires a different node for each type of expectation or iteration restriction. This is why the different features used here correspond to the different projections in grammatical frameworks such as Government Binding theory.

Node Closure

In addition to providing the necessary flexibility in the specification of the phrase structure of the sentence, Structure Unification Grammar localizes information in a way that allows completed nodes to be closed from further access by the syntactic parser. Closed nodes can be removed from the phrase structure representation, thereby reducing the number of nodes which NNEP needs to store information about, and allowing it to stay within its memory bounds. Because NNEP outputs all the information about the phrase structure of the sentence as it computes it, forgetting nodes does not interfere with the interpretation of the output. The mechanism for closing nodes, called the forgetting operation, does not imply any particular node closure strategy; it simply provides a sound mechanism for implementing such a strategy.

For the forgetting operation to be sound, its use cannot allow the forgotten information to be contradicted later in the parse. Because forgetting a node prevents any future parser actions at that node, soundness can be guaranteed as long as all the information about a forgotten node would only be needed to test the consistency of parser actions at that node. The only information in an SUG description which is a problem for this requirement is immediate dominance. Some parser actions need to test whether a node has an immediate parent, but if that parent has been forgotten, then this information would not be available. Since no parser actions need to know the actual identity of the immediate parent, this problem can be easily solved by representing immediate dominance in two parts, dominance (for ordering constraints), and having an immediate parent. The later information is a property of an individual node, so forgetting the parent will not interfere with accessing this information. With this change in representation, forgetting a node will never allow the parser to compute an analysis which would otherwise be impossible. It may, however, prevent the parser from finding an analysis which would otherwise have been possible. Thus the parser wants to avoid forgetting nodes which have a significant chance of being involved in a parser action. In particular, it never wants to forget nodes which must be equated with in order for the parse to be completed.

The Parser's Grammars

Because the constraints being investigated in this dissertation are computational in nature, the primary concern is the nature of the parser, not the nature of the grammar. This was manifested in the design of the notation used to specify grammars, given above. It is also manifested in the grammars themselves. The grammars partition their information according to how the parser wants to access that information. For example, most grammar entries are lexicalized. A lexicalized grammar entry specifies all the information about the phrase structure of the sentence which can be determined given the presence of a single word. This allows the parser to easily access all relevant grammatical information about a word in the input. It does not, however, result in the most compact or most easily learned representation of the grammatical information. Some grammatical investigations have addressed this conflict between the needs of a parser and the needs of simple grammatical specification. The first was Generalized Phrase Structure Grammar (Gazdar *et al.*, 1985), which uses meta-rules to specify the grammar, and then compiles these meta rules into a set of Context Free Grammar rules, which can be easily parsed. Meta-rules have also been used in the specification of Tree Adjoining Grammars (Becker, 1993). Head-driven Phrase Structure Grammar (Pollard and Sag, 1987) addresses this issue using an inheritance hierarchy. The inheritance hierarchy approach would probably be more effective for a unification based grammatical framework such as the one used here, but this issue has not yet been addressed. It would be best addressed in an investigation of grammar learning, since that is when capturing generalizations across lexical grammar entries is most important.

As will be discussed in more detail in section 4.2, NNEP uses grammars specified in Structure Unification Grammar, but not all SUG grammars are supported by NNEP. As just mentioned, the grammars used in this document are for the most part lexicalized. Each lexicalized grammar entry is a rooted tree fragment with exactly one phonetically realized terminal, which is the word of the entry. Different uses of a word are specified using separate grammar entries. All the grammar entries shown above in figure 1.2 are lexicalized. Nonlexical grammar entries are rooted tree fragments with no words. They can be used to express constructions for which no specific lexical evidence is necessary, such as relative clauses without overt wh- words. Other forms of grammar entries are possible, but for the purposes of this investigation, these types are adequate.

Because NNEP imposes some constraints which are not imposed at the level of the SUG grammar, the meaning of a given description in SUG is sometimes more general than the meaning of the same description in NNEP. This difference is just the traditional split between competence and performance. The SUG interpretation of the grammar is the competence grammar, and the possibilities which are ruled out by NNEP are the performance constraints. In general, constraints should be expressed in the competence grammar unless they receive a simpler treatment in terms of parsing constraints. In NNEP the traditional division of data into competence or performance phenomena is for the most part maintained. The only exception is that some constraints on long distance dependencies are enforced through constraints on NNEP's ability to recover these dependencies, while they are traditionally treated at the level of the grammar.

It should also be noted that the information specified in the parser's grammar is only for the internal use of the parser; it is not the output of the parser. These grammar entries and the phrase structure nodes in them are assumed to be linked to structures at other levels of representation. The output of this syntactic constituent structure parser identifies what grammar entries are used in the parse, and how the nodes in these grammar entries overlap in the final phrase structure tree. Given this information, the associated structures at other levels of representation can be combined in the associated ways, producing (or helping to produce) the final structures for those levels. This

assumption about how the syntactic level of representation is related to other levels of representation subsumes the relationship between constituent structure and functional structure in Lexical Functional Grammar (Kaplan and Bresnan, 1982), and the relationship between syntactic structure and predicate-argument structure in Combinatory Categorial Grammar (Steedman, 1987).

1.3.2 Recovering Phrase Structure Trees

The purpose of proposing a parsing model in this dissertation is to use it in the argument that the S&A architecture is computationally adequate and makes linguistically significant predictions for syntactic parsing. This means that the parser must be able to comply with the computational constraints outlined in section 1.2.3, and it must adequately address the linguistic issues which are of particular concern given those constraints. It does not have to be a complete model of syntactic parsing. However, future research on parsing in the S&A architecture will want to develop such a complete model. The model presented here can be seen as a first step in this direction, and as such it provides some indication of what such a complete model of syntactic parsing in the S&A architecture would look like. However, the discussion here will concentrate on the narrower concerns of this particular investigation. The design of this parsing model is discussed more thoroughly in chapter 4.

The basic characteristics of this Neural-network Node Equating Parser (NNEP) were presented above in the discussion of Structure Unification Grammar. NNEP uses SUG's phrase structure descriptions as its representation of phrase structure information, and computes SUG's derivations in recovering that phrase structure information from the words of a sentence. NNEP's parser state represents an SUG description which specifies the information that has been determined so far about the phrase structure of the sentence. NNEP's operations compute the SUG derivation steps which combine this intermediate description with descriptions from the grammar and perform node equations. NNEP outputs each of these derivation steps as they are computed, thereby outputting all the information which NNEP adds to its parser state as soon as the information is inferred. When the parse is done, NNEP checks to make sure it has produced a complete description, thereby ensuring that NNEP will only accept sentences which the grammar specifies as grammatical.

The set of SUG derivations which NNEP can compute is limited by the computational constraints discussed in section 1.2.3. Because NNEP must produce incremental output, the phrase structure information which is implied by the presence of a word must be added to the parser state (and therefore output) when the word is input. This information is precisely the grammar entry for the word, provided there is no lexical ambiguity. If there is more than one grammar entry that could be used for a word, then because no disjunction is allowed in the parser state, one of them must be chosen.[17] In some cases this forced choice can result in a mistake, thereby predicting a garden path.[18] The parse shown in figure 1.3 gives examples of three parser operations which add the information in a grammar entry to the parser state (attaching, double attaching, and equationless combining).

[17] It is possible that predicates could be defined which represent a bounded disjunction between grammar entries, or portions of grammar entries, thus allowing lexical disambiguation to be delayed. However, this would greatly complicate the parser, since such predicates would require a very complex interpretation which is rather different from the node-local features represented by most other predicates. Thus this alternative has not been pursued, although perhaps the constrained use of such predicates would be feasible.

[18] This discussion is a slight simplification. In the complete model (discussed in chapter 4), the parser can wait for information about the immediately following word in cases where it isn't sure which grammar entry to pick. Also, not all grammar entries are associated with words, so some ambiguities can be handled by delaying the addition of one of these nonlexical grammar entries.

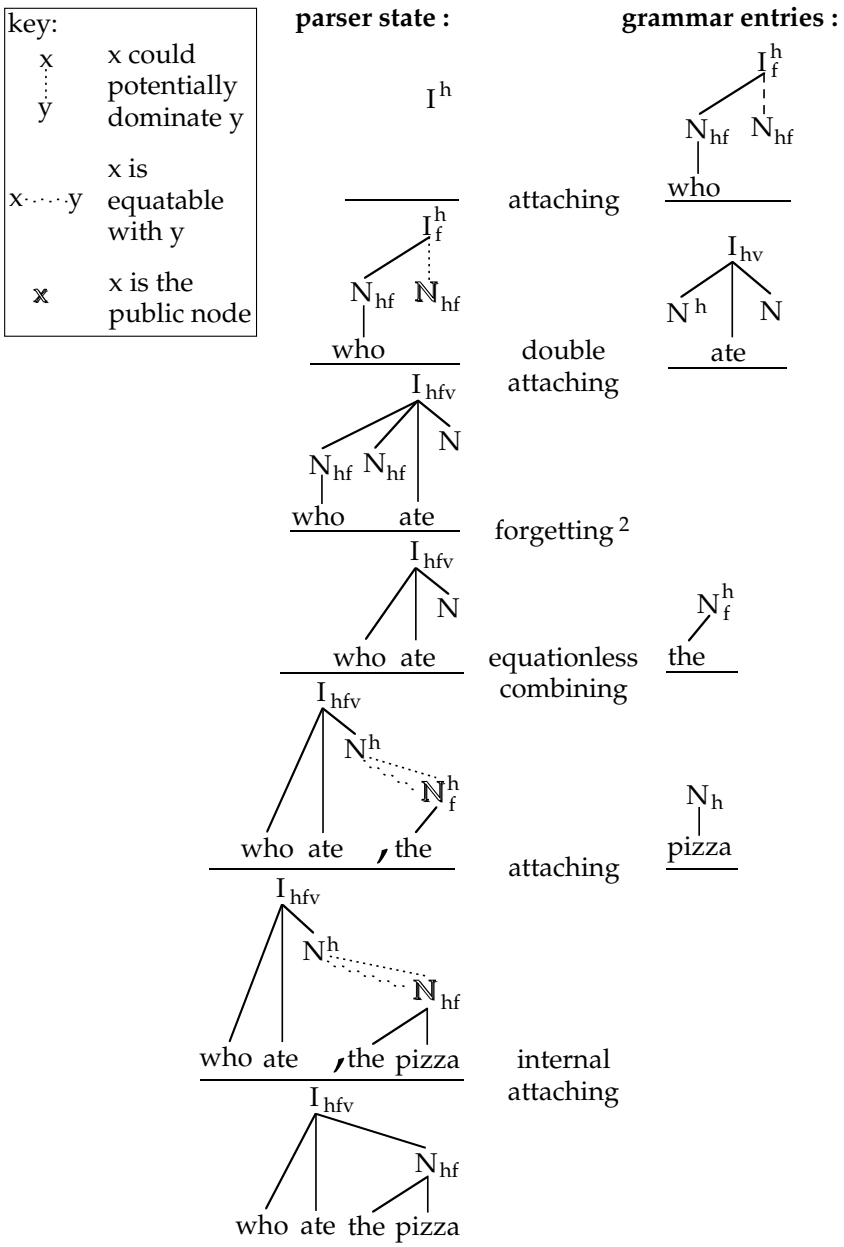


Figure 1.3: An example parse of "Who ate the pizza".

In contrast to the information in grammar entries, the equations between nodes in the grammar entry and nodes already in the parser state do not necessarily have to be specified. For example in figure 1.3, when the grammar entry for "who" is added, its root is equated with the sentence node which initialized the parser state, but when "the" is processed, it's grammar entry is not attached to the tree fragment from the previous portion of the sentence. Such delays in attachment decisions are necessary when there is not enough information available at that time to make a commitment to one equation, since the determinism requirement prevents the retraction of commitments. In this case, NNEP can't be sure that "the" is the start of the object of "ate", since it might also be the start of the possessor of the object of "ate", as in "Who ate the pizza's crust". If an equation decision is delayed, one of the possible equations can be performed later in the parse when there is enough information available. In this example, the equation is done when the end of the sentence is reached, at which point there can be no forthcoming possessive marker. After this equation, NNEP has specified a single immediate dominance tree, and there are no remaining superscripts, so the parse has been completed successfully.

The SUG derivations which can be computed by NNEP are also constrained by the S&A architecture's bounded memory capacity. In NNEP's implementation, variables refer to nonterminal nodes in the current phrase structure description. Thus this description can have at most ten nonterminals in it at any one time. The forgetting operation discussed in the previous section is used to stay within this bound. For example in figure 1.3, after "ate" is processed, the two NP's on the left are no longer on the right frontier of the sentence. Thus no other nodes will be equated with them, and NNEP can safely close them off from further consideration. Since this level of representation is only being used for syntactic parsing, forgetting these nodes does not interfere with processes which might involve their associated nodes at other levels of processing. The resulting parser state only requires two variables; the terminals are only shown for readability. In other cases, such as long right branching sentences, it may be necessary to close nodes which could conceivably be involved in further equations. For the purposes of this investigation, any closure strategy can be used in these cases, as long as the only eliminated parses are for readings which people do not get. The memory capacity of the parser is also bounded by the fact that at most three instantiations of each relation can be stored, and that the set of predicates used in the implementation is fixed. This means that any data structures used by the parser must be of bounded size. As will be discussed in section 6.4, these turn out to be the significant bounds for constraints on center embedding, not the bound on the number of nonterminal nodes.

The constraint that rules must apply to each variable independently also limits the set of SUG derivations that NNEP can compute. It limits the ways that grammar entries can be added to the parser state, the equation of nodes within the parser state, and the calculation of binary relations. In the parser's implementation, grammar entries are represented in rules. For each grammar entry, rule patterns look for ways that the grammar entry can be combined with the phrase structure description in the parser state. Because these rules must apply to each variable independently, the only combinations that can be computed are those whose consistency can be tested based on information about one phrase structure node and the tree as a whole. Other nodes can be involved in the combination, but only if they can be uniquely identified at the time, and therefore can have their information represented as information about the tree as a whole. The simplest example of a combination is given at the top of figure 1.3. The only node in the parser state that is involved in this combination is the matrix root, where the grammar entry for "who" is attached. All the information about the nodes in the grammar entry is compiled into the rule, so the rule does not have to refer to them with variables. The equationless combining operation, shown in the middle of figure 1.3, does not involve any nodes in the parser state. It does need to know about the existence

of certain nodes in the parser state, but since the actual identities of these nodes does not need to be known to choose this operation, this information can be represented as information about the tree as a whole. The only other simple combination that is possible is shown at the bottom of figure 1.4, where a tree fragment in the parser state is attached to a grammar entry. However, the mechanisms that are needed for calculating long distance dependencies makes a fourth combination operation possible. As illustrated in the second line of figure 1.3, this operation (double attaching) simultaneously attaches the grammar entry to a node in the parser state, and attaches a node in the parser state to the grammar entry. This operation involves two nodes in the parser state, but because the lower attached node must be "the public node", all the information about this unique node can be represented as information about the tree as a whole, and therefore this combination can be calculated in terms of one variable referring to the node where the grammar entry is attached. The role of the public node in calculating long distance dependencies will be discussed next.

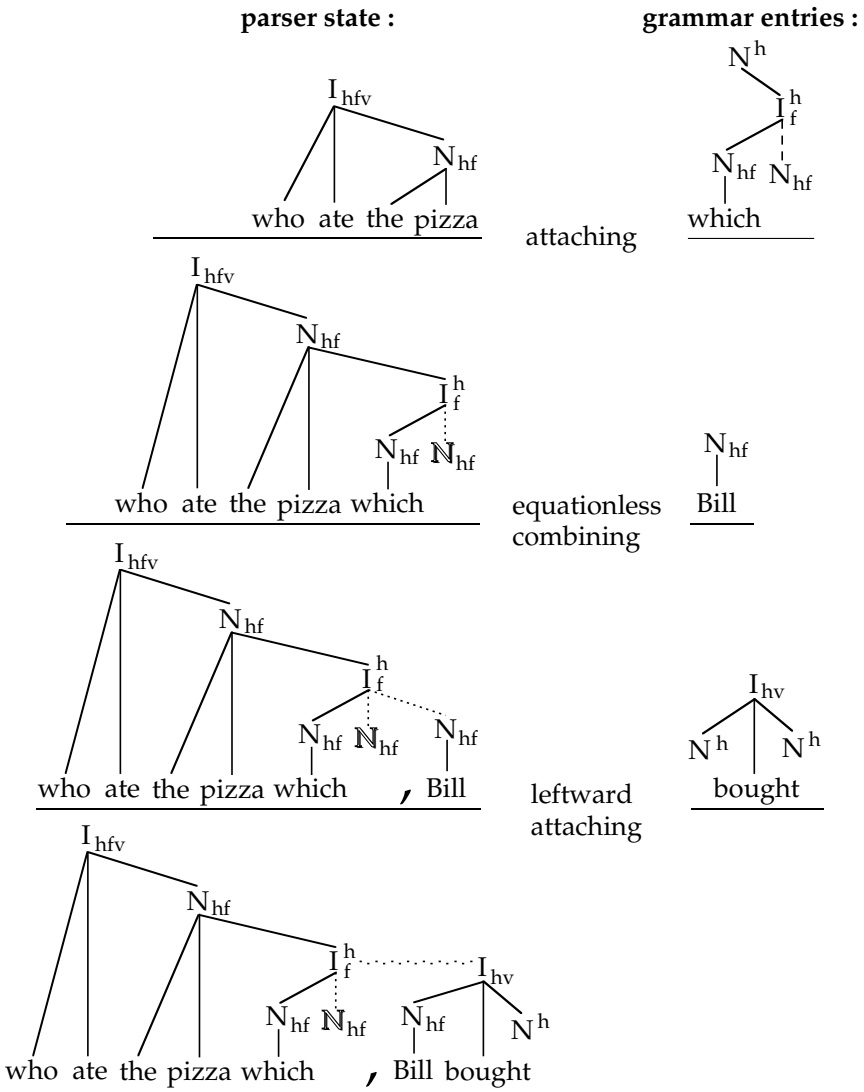


Figure 1.4: The beginning of a parse of the relative clause in "Who ate the pizza which Bill bought".

As it turns out, very few relations need to be explicitly represented in NNEP's parser state, but

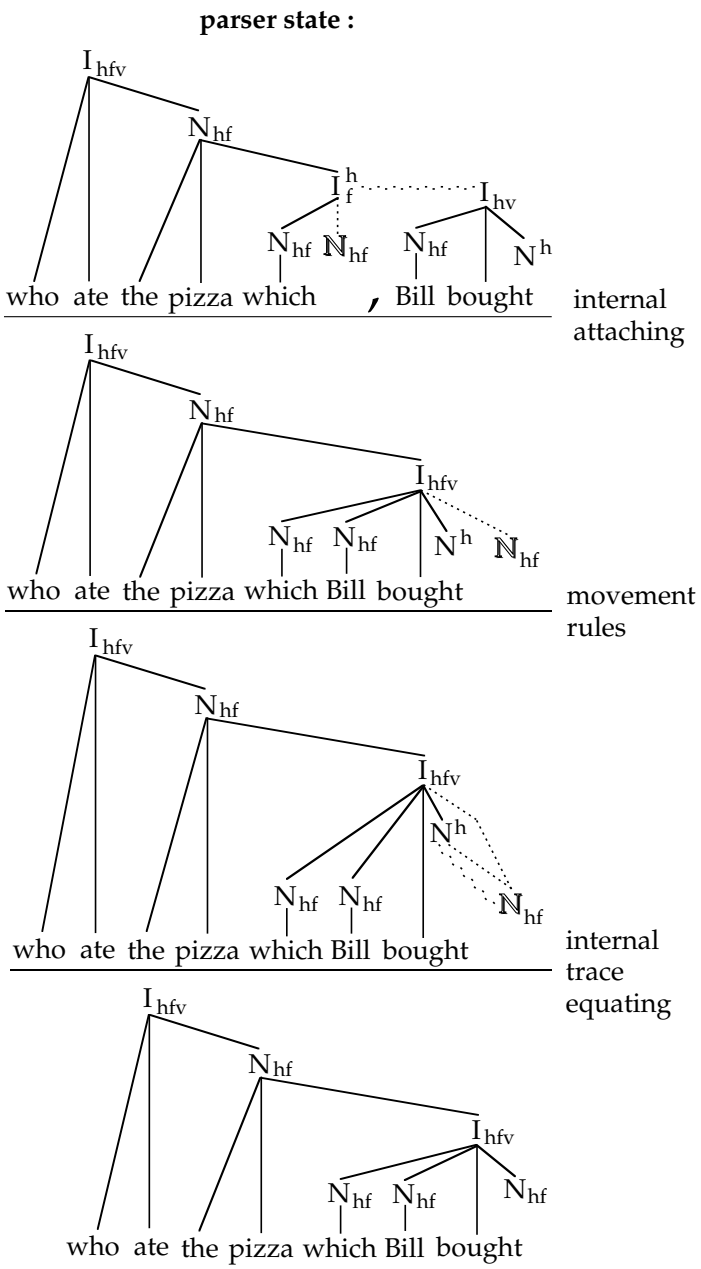


Figure 1.5: The end of a parse of "Who ate the pizza which Bill bought".

the constraint that the rules which manipulate these binary relations must apply to each variable independently still has significant implications. The only explicitly stored relations that are required represent possible equations between nodes and possible dominance relationships between nodes. These are shown in figures with scattered dotted lines and dotted lines, respectively. Both these relations are needed for calculating long distance dependencies. As mentioned in section 1.3.1, long distance dependencies are represented using a trace node, which must equate with the node for the gap site. Finding the correct gap node for the trace node to equate with involves recursively finding what nodes might dominate the trace node and what nodes might be equated with the trace node. These relationships can be stored with NNEP's two binary relations. Figure 1.5 illustrates the application of the rules which calculate new instances of these relationships (the movement rules). In this example, the object of "bought" could either dominate the trace node for "which" or be equated with it. These calculations require information about both the trace node and the candidate node. To do this calculation with rules that apply to each node independently, all the relevant information about one of the two nodes must be represented as information about the tree as a whole, and therefore one of the nodes must be uniquely identifiable. Since a given grammar entry may have multiple possible gap sites, the candidate node is not uniquely identifiable. Thus we must assume that at any given time there is only one trace node for which these relationships need to be calculated. Since testing equatability requires virtually all the information about the trace node to be represented as information about the tree as a whole, this unique node is called the public node (its information is publicly available). In English, the most recently introduced trace node is always the public node, so NNEP needs a stack data structure to keep track of which trace node was introduced most recently. This stack is called the public node stack. The need to restrict access to trace nodes using a stack predicts a number of interesting constraints on long distance dependencies (discussed in section 6.2), and an independently motivated bound on the depth of this stack predicts some constraints on center embedding (discussed in section 6.4). In order to allow the efficient processing of subjects, they are sometimes also placed on the public node stack. This allows the double attaching operation to apply to them.

Possible dominance relationships and possible equality relationships are also needed for the roots of unattached tree fragments in the parser state. Because of the nested nature of English, these relationships only need to be accessed for the rightmost tree fragment root.[19] Thus these binary relations can be accessed using unary predicates identifying the nodes that might dominate or might be equated with this unique tree fragment root. These relationships are set when the grammar entry for the tree fragment root is added to the parser state, as is illustrated when equationless combining is used in figure 1.3.

Ordering constraints are an area that might seem to require binary relations, but unary predicates turn out to be adequate. In most cases ordering constraints can be expressed between a nonterminal and a terminal (e.g. prehead, post-determiner, etc.), thus allowing them to be represented with unary predicates for the nonterminals. For the grammatical representations used here, ordering constraints between nonterminals are only needed for the objects of ditransitive verbs. Using a binary relation to represent this ordering constraint can only be avoided if at any given time there is only one such ordering constraint that needs to be stored. Under these circumstances the relationship can be represented with two unary predicates, *preceding* and *preceded*. This requirement

[19] For languages, such as Dutch, which do not have a nested structure, other access orders could be used, as long as at any given time only one tree fragment needs to be considered. It is also possible that such languages use structures that do not require the use of the tree fragment list in these cases. For example, arguments could be adjoined into a verb's projection (like English prepositional phrases can be), rather than the verb subcategorizing for the arguments. These possibilities also apply to calculating long distance dependencies for other languages. These relationships also do not always respect a nested structure.

predicts a strict constraint on center embedding with ditransitive verbs, which will be discussed in section 6.4. The dominance relationships that are needed to calculate the inheritance of these predicates are also represented with unary predicates. These predicates identify which nodes are in each tree fragment in the parser state, and which nodes are in a newly added grammar entry. Operations which attach a tree fragment or grammar entry also propagate ordering constraints from the site of the attachment to all the nodes in the attached tree fragment or grammar entry.

As discussed above, the need to delay attachment decisions means that sometimes two nodes already in the parser state must be equated. Since the rules that choose and perform these equations must apply to each node independently, only one of the two equated nodes can be chosen by the rule, and the other must be uniquely identifiable. The pairs of nodes which these rules might need to equate are exactly those for which the equatable relationship is stored in the parser state, and the constraints which apply to the calculation of these relationships also apply to these rules. Thus only the rightmost tree fragment can be attached to its adjacent tree fragment, and only the most recently introduced trace node (the public node) can be equated with its gap site. The implications of these restrictions are the same as for the calculation of the relations. The attachment of a tree fragment is illustrated at the bottom of figure 1.3, where the delayed attachment of "pizza" as the object of "ate" is performed. The equation of a trace node with its gap site is illustrated at the bottom of figure 1.5, where the trace node for "which" is equated with the object of "bought".

1.3.3 The Connectionist Implementation

While it is important to be able to investigate parsing issues at the level of symbolic computation, there are some issues which need to be addressed at the level of connectionist computation. Most of these issues were discussed in the analysis of the constraints imposed by the Shastri and Ajjanagadde connectionist architecture, but an actual implementation of the parser is still needed to test the validity of that analysis, and to address issues which are not relevant at the level of symbolic computation. This section gives an overview of NNEP's implementation in the S&A connectionist architecture. Chapter 5 discusses this implementation in more detail.

The S&A architecture was developed for modeling fast common sense reasoning (called reflexive reasoning) (Shastri and Ajjanagadde, 1993), and here it is used to implement a special purpose module for syntactic constituent structure parsing. It is a module in that the predicates and variable bindings of the network are specific to the network's particular task. It is not currently clear whether this network is part of a larger module which includes things such as the calculation of predicate-argument structure, or whether it is a distinct module that interacts highly with other stages of language processing.

The network has three basic parts, input units, predicate units, and grammar units. The temporal pattern of activation over the predicate units represents the information the parser needs to know about its parser state. The phases in this pattern of activation represent variables that refer to nonterminal nodes in the phrase structure of the sentence, and the predicates represent properties of these nodes and of the phrase structure tree as a whole. Since these predicate units represent a feature decomposition of the types of nodes, and these features are only interpreted within this module, the predicate units are analogous to hidden units in Parallel Distributed Processing (Rumelhardt and McClelland, 1986) networks.

The network's input units are just a stand-in for the word recognition component of a complete

system. There is one input unit per word.[20] When the next word is input, that word's unit becomes active and stays active (across all phases) until a grammar entry for that word is combined with the parser state. Links from these units go to units for the grammar entries of the word. These primary links are inhibited by links from the units that represent predicates. These inhibitory links filter out all the phases for phrase structure nodes which cannot be sites for a combination with the grammar entry. Thus primary links from the input units and inhibitory links from the predicate units implement the patterns for the pattern-action rules that calculate what action the parser should take given the parser state and the next input word. For example, when the parser is in the state shown in the second line of figure 1.3, the input unit for "ate" is active across the phases for all three nonterminal nodes, but an inhibitory link from the $-I$ predicate (among others) blocks this activation in the phases of the two N nodes. The activation for the I node makes it through the filter, because this node is compatible with the requirements on possible sites for double attaching with the grammar entry for "ate". Note that since the trace node is the public node, the relevant information about it is represented with predicate units that are active across all phases (i.e. predications about the tree as a whole), so that if that information was not compatible with the combination, then no phases would make it through the filter. Other such sets of links implement the patterns for parser actions that do not require the input of a word (such as internal attaching, or using a nonlexical grammar entry). Using link weights and continuous valued unit output, these patterns of links can be used to estimate the probability that the pattern's action is the right thing to do.[21]

The input activation provided to the grammar units by the above patterns is used to choose what parser action to perform next. The nature of the arbitration network that should be used to make this disambiguation decision has not been significantly addressed in this work, but see (Stevenson, 1994) for an investigation of these issues. Once this choice has been made, the grammar unit for the chosen parser action fires in the phase of the chosen site. This is the output of the parser, since this is the earliest indication of what the parser has decided, and the sequence of parser actions completely determines the information that the parser recovers about the phrase structure of the sentence. This output is also used to trigger the action component of the pattern-action rule that calculates the chosen parser action. This action changes the states of the predicate units to reflect the new information that is implied by the chosen parser action. An action is implemented with links from the grammar unit to the units for new predications about the site of the parser action, plus a unit that gates (using inhibitory links) links to units for the new predications about other nodes in the grammar entry. These gated links may introduce new nodes into the parser state, and may add information about uniquely identifiable nodes. Continuing the above example, the output of the unit for the grammar entry of "ate" would set the new information about the I node, and cause its gate unit to temporarily become inactive. The inactive gate unit would allow activation to flow across links that add the new information about the trace node and introduce a new node for the object of "ate".

The forgetting operation, which removes nodes from the parser state, is implemented with a pattern-action rule that looks for nodes which have at most a small chance of ever being involved in any

[20] The complete model, discussed in chapter 5, also includes input units for the word after the next word. These units are used to allow one word lookahead to help with some cases of lexical disambiguation.

[21] Estimating this probability is sometimes more complicated than this description would suggest. Since some of the actions that the parser can perform (for example equationless combining) are more general than multiple other possible actions, the probabilities for the later actions need to be summed to get the probabilities for the former actions. This sum then needs to be adjusted for the probability that the more general action will lead to an overflow of the parser's resource bounds. Only preliminary work on estimating these probabilities will be presented in this document.

future parser actions, suppresses all the predications about these nodes, and makes their phases available for future use.

In addition to the pattern-action rules for parser actions, there are some rules for calculating the indirect implications of the predications that are directly added by the above rule actions. These are implemented with links that propagate activation from the directly set predicate units to the units for the implied predications. These rules include those that calculate possible long distance dependencies, and rules that transfer information about the public node to predications about the structure as a whole.

1.4 Overview of the Evaluation

The previous section described a model of syntactic parsing (NNEP) which is designed to comply with the constraints imposed by the S&A architecture and the nature of the parsing task. In this section, NNEP will be used to argue that the S&A architecture is computationally adequate for syntactic parsing, and that it makes linguistically significant predictions. These arguments and the tests used to make them are discussed in more detail in chapters 6 and 7. To argue for the adequacy of an architecture, it is not sufficient to perform tests simply on a small set of examples, or on phenomena which the architecture is well suited for. The phenomena which are likely to be difficult for the architecture need to be identified, and empirical tests need to be performed on these phenomena. Because the limitations of the S&A architecture which are significant for syntactic parsing have been characterized at the same level of abstraction (symbolic computation) as has traditionally been used in the study of linguistic phenomena, it is fairly easy to identify the phenomena which are of particular concern. To argue that an architecture makes linguistically significant predictions, it is simply necessary to provide examples of such predictions. Since the predictions are due to the computational constraints imposed by the architecture, they pertain to the same phenomena that need to be tested to determine the architecture's adequacy. Thus both these arguments will be made by testing NNEP's ability and inability to parse sentences pertaining to the phenomena that are of particular concern given the computational constraints imposed by the S&A architecture and the nature of the parsing task.

While it is important to pay particular attention to phenomena where the limitations of the architecture are likely to be significant, it is also necessary to guard against errors in the identification of these phenomena. There is no complete proof that the relevant limitations of the S&A architecture and the relevant phenomena for these limitations have been completely identified. For this reason, NNEP is also tested on a set of randomly selected, naturally occurring sentences. This provides an essentially unbiased test of the parser's ability to handle the diversity of phenomena in natural language. While it is not necessary for the parser to handle every phenomena in this test set, specific arguments need to be made that any excluded phenomena can be handled by extensions to the parser.

The primary concern of this investigation is to demonstrate the adequacy and significance of the S&A connectionist architecture for syntactic parsing, but the parsing model developed here has some additional characteristics that may be of significance for future work in this area. These characteristics are summarized at the end of this section, and are discussed in more detail in section 7.3.

1.4.1 Phenomena of Particular Concern

Four types of phenomena are of particular concern given the limitations of the S&A architecture. As discussed in section 1.2.3, the parser must be deterministic, must be implemented with rules that apply to each variable independently, cannot store more than three instantiations of any relation, can use only a bounded number of variables, and must parse in quasi-real time. The determinism constraint requires testing with locally ambiguous sentences. The parser needs representations which don't use disjunction but still allow the resolution of a local ambiguity to be delayed long enough for disambiguating information to be found. The issue of whether the parser can make the right choice given disambiguating information is not of particular concern here, given the general success of connectionist networks in disambiguation tasks.[22] This representational requirement in turn places constraints on the representation of grammatical analyses. Thus we also need to test the parser's ability to express phrase structure analyses that accurately characterize the language. If the parser doesn't have accurate information about the language, the inaccuracies may introduce false ambiguities that mislead the parser.

In NNEP, binary relations and the rules that manipulate them always involve phrase structure nodes which are looking for an immediate parent. These nodes include trace nodes, so testing on long distance dependencies is necessary. They also include subject nodes and delayed attachment decisions, but these phenomena are adequately covered in the local ambiguity and phrase structure analysis data. The data structures which are used to comply with the locality constraint on rules also introduce constraints. These data structures are of bounded size, so they require testing on center embedded sentences. These bounds guarantee that no more than three instantiations of any relation ever need to be stored. There can be at most three unattached tree fragments (other than the matrix root), and there can be at most two nodes on the public node stack. Since the rules that manipulate the relationships involving these two types of nodes are disjoint, these relationships are also stored in disjoint relations. Thus at most three bindings for the tree fragments' relations and at most two bindings for the public node stack's relations need to be stored. This guarantees that the constraint on the storage of relations will never be violated, so no additional testing is needed for this constraint.

The bound on the number of variables that the parser can use at any one time also requires testing on center embedded sentences. These are the sentences which necessarily involve remembering a relatively large number of nodes, and therefore require using a relatively large number of variables. The question of how many nodes could possibly, but not necessarily, need to be remembered has not been tested, since no suitable set of data has been found. For example, according to the grammar, a long right branching sentence could have a modifier attached to any one of the nodes on the right frontier, and thus all of these nodes would need to be stored. However, as discussed in section 4.1.3, it is well known that the set of nodes which are available for such modification is severely restricted ((Church, 1980), (Gibson *et al.*, 1993)). Theories of these performance constraints indicate that the parser's bounded number of variables is not a problem for such sentences.[23]

Each operation that NNEP performs takes a bounded amount of time, so to show that NNEP can

[22] NNEP does have a plausible mechanism for making disambiguation decisions, but because it is not a central issue for this dissertation, and because fully addressing this problem would be a large project in and of itself, the proposed mechanism has not been adequately developed or tested. See sections 4.5 and 7.4 for discussion of this mechanism and future work in this area.

[23] This is an example of how the resources necessary for maintaining local ambiguities can interact with the bounds on these resources. In general, these interactions have not been investigated here. Acquiring the data necessary for such an investigation would be difficult, since it is hard to control when people make disambiguation decisions and when they delay them.

parse in quasi-real time, it is sufficient to show that only a bounded number of operations need to be performed between combining any two words. The only operations which can be performed between two words are internal attaching, internal trace equating, and combinations that involve nonlexical grammar entries. Both the internal operations remove an unparented node from the parser state. Since at most five such nodes (two nodes on the public node stack and three non-matrix tree fragment roots) can be in the parser state, at most five of these operations can be performed between any two words. In practice, the number of such operations can probably be bounded to one or two, but this question hasn't been adequately addressed. Nonlexical grammar entries are only needed for certain cases of ambiguity, so locally ambiguous sentences need to be used to test this constraint as well.

From this analysis we see that NNEP needs to be tested on center embedded sentences, local ambiguities, long distance dependencies, and phrase structure analyses. This analysis is presented in more detail in chapter 7.

1.4.2 Testing for Adequacy and Significance

To test NNEP on the specific phenomena identified above, papers were selected from the literature which discuss a representative sample of the data on these phenomena. NNEP was then tested on its ability to at least parse the data in the papers, if not adopt the same analyses.[24] For an approximately unbiased sample of sentences, a set of sentences randomly selected from the Brown Corpus was used. All of these tests deal only with English data, except to the extent that the analyses inherited from the papers generalize to other languages. Chapter 6 gives a detailed presentation of the results of these tests.

To test NNEP's ability to express phrase structure analyses that accurately characterize the language, the phrase structure analyses in (Kroch, 1989) were used. In (Kroch, 1989), constraints from Government Binding theory are expressed in the Tree Adjoining Grammar (TAG) framework. The similarity between SUG and TAG made this paper particularly appropriate for this task. As is the case in the examples given above, NNEP's grammars represent each lexical projection and all its associated functional projections as a single SUG nonterminal node. All the information about these projections are expressed in the feature structures which label the node. This compact representation is possible because SUG can represent multiple types of expectations and iteration restrictions on a single node. Ordering constraints within these projections can be specified with respect to the distinguished terminal nodes (constituent head, functional head, and verb).[25] Any schematized local collection of nodes can be represented in this way, so the test was successful. Section 6.1 discusses this test further.

The analyses in (Kroch, 1989) were also used to test NNEP's ability to recover long distance dependencies. Again the use of TAG in this paper was useful, because NNEP's rules for calculating long distance dependencies factor the local component of that dependency from the recursive component, just as is done in TAG. The local rule calculates what constituents could be the gap for a trace node, and the recursive rule calculates what constituents could have the gap somewhere within them. Because the relevant trace node is always uniquely identifiable as the public node,

[24] While each of these areas deserve a more detailed analysis, a broad and shallow test is appropriate for this stage of the investigation. A demonstration of the feasibility of addressing all of these issues is necessary to justify the detailed investigation of any one of them.

[25] Ordering constraints involving subjects are enforced by limiting the parser operations that can be used with the grammar entry, since the presence of a subject in the parser state changes how the subject's sentence can be attached to.

all the relevant information about this node can be represented as information about the phrase structure tree as a whole. In this way the two rules can apply to each candidate node independently, and thus the locality constraint on rules does not need to be violated. Most of the constraints on long distance dependencies are simply compiled into features on nodes in grammar entries (i.e. *extractable* and *not_barrier*), and enforced by the long distance dependency rules. However, some of the constraints are predicted by the need to restrict these rules' access to trace nodes. Since the public node is the top node on the public node stack, these rules can only access the most recently introduced trace node. This constraint is used to explain the that-trace effect, the cases of subject islands that precede inflection, and the limited possible extractions out of wh- islands. The later phenomena are particularly interesting, because accounting for this data required Kroch to go outside the power of TAG. Thus by accounting for this phenomena with a computational constraint, the competence theory of long distance dependencies can be simplified. This explanation for wh-island constraints is also interesting in that it subsumes Pesetsky's path containment condition (Pesetsky, 1982). In summary, all the data in (Kroch, 1989) was correctly categorized, mostly by adopting the same analyses, and some of the phenomena were predicted by the computational constraints imposed by the S&A architecture. Section 6.2 gives an extensive discussion of these results.

To test NNEP's ability to handle local ambiguities, the data from the chapters on ambiguity resolution in (Gibson, 1991) were used. (Gibson, 1991) is particularly well suited for this purpose because Gibson surveys the literature on ambiguity resolution and discusses the relevant data. If a given sentence prefix can be continued in more than one way, then at that point NNEP needs a representation of the sentence's phrase structure which is compatible with both continuations. Because NNEP can delay attachment decisions, ambiguities in the way two grammar entries are connected can be represented for as long as necessary. Because SUG's partial descriptions allow NNEP to avoid specifying information which it doesn't yet know, NNEP can use grammar entries which are compatible with all the acceptable continuations after a lexical ambiguity. This may involve leaving some structure unspecified. For example, the ambiguity between a sentential complement and a relative clause requires that the relative clause's modification relationship and trace node not be specified until the gap is found. This can be handled with grammar entries which specify the delayed structure information, but which are not associated with a word. The addition of such nonlexical grammar entries can be delayed until there is disambiguating information. In the test sentences, no more than one such combination ever needed to be done between any two words, so this mechanism does not pose a problem for parsing in quasi-real time. Because SUG's partial descriptions allow NNEP to specify the information it does know independently of the information it needs to leave unspecified, the information which is necessary to resolve an ambiguity is available for decision making. In a few cases the information which NNEP must have for making a disambiguation decision also includes the word immediately following the current word. If the parser has to make a decision and it can't decide based on the left context and the next two words, then a garden path is predicted in one of the alternatives. There is one pair of sentences for which this prediction may be a problem, given below. Since "found" is obligatorily transitive, looking at the immediately following word to see if it could be the start of an object would ordinarily allows this reduced relative/main verb ambiguity to be resolved, but because of the heavy NP shift, this is not possible for these sentences. Gibson (personal communication) agrees that experiments are needed to determine whether in any given context one of the sentences is a garden path, as this model predicts. With this one caveat, all the acceptable data is parsable. NNEP accounts for the unacceptability of "The horse raced past the barn fell" (Bever, 1970), but otherwise no attempt was made to account for the unacceptable data. Some of the unacceptable data is probably due

to the disambiguation mechanism's efforts to conserve resources, but this possibility has not been investigated. Section 6.3 discusses the results of this test in detail.

(247a) The bird found in the room was dead.
(249a) The bird found in the room enough debris to build a nest.

The test of NNEP's ability to handle center embedded sentences used the data from the chapters on processing overload in (Gibson, 1991). Again, (Gibson, 1991) is particularly well suited for this purpose because it surveys the literature. In addition, some example sentences involving nested ditransitive verbs were constructed and used. Since the interaction between ambiguity and resource requirements is not being tested here, nodes were closed as soon as possible, using the forgetting operation. None of the acceptable sentences required more than ten nonterminals to be stored at any one time, so the architecture's bounded memory was not a problem. In fact, the maximum number of nonterminals required was nine, given the compact phrase structure representation used here. This is interesting because nine is the maximum of the robust bound on human short term memory of seven plus or minus two (Miller, 1956). The data structures which are used to handle the locality constraint on rules also result in some bounds. The public node stack can be at most two deep, there can be at most three unattached tree fragments in the parser state, and there can be at most one first posthead argument node for a ditransitive verb (usually the direct object). None of these constraints need to be violated to parse any of the acceptable sentences in this data set. In addition, much of the unacceptable data is ruled out, mostly due to the bound on the depth of the public node stack and a particular (independently motivated) strategy for when to specify a tree fragment root as the public node. Not all the unacceptable data, however, is predicted by these constraints. Some of this data is probably due to interactions between the resources necessary for maintaining ambiguities and these resource bounds, but this possibility has not been investigated. Section 6.4 presents the results of this test in more detail.

To make sure the above phenomena-specific tests did not miss anything which would be difficult for the S&A architecture, NNEP was also tested on an essentially unbiased sample of sentences. This set of fifty thirteen word sentences were randomly selected from the Brown corpus by Ezra Black in 1991. Since the issue being addressed is the adequacy of the S&A architecture, and not the adequacy of NNEP as it is currently designed, incompleteness in NNEP's coverage of the phenomena in this data set is only a problem to the extent that extensions to NNEP aren't likely to be able to handle the phenomena. Indeed there are some phenomena which NNEP is not yet equipped to handle, but none of these are expected to be any more difficult for this architecture than they are in general. In particular, NNEP cannot parse coordinations, or gapping in comparatives, and it cannot make many disambiguation decisions. As argued above, this architecture is well suited for doing disambiguation. I expect that the relationship between SUG and Combinatory Categorial Grammar (Steedman, 1987) will make the analyses of coordination and gapping easier for this parsing model than for most phrase structure based parsers. Due to the scope of these topics, they will have to be left for future work. These topics are discussed further in section 6.5.

1.4.3 Advantages of the Architecture

Because the primary concern here is the adequacy of the S&A architecture for syntactic parsing, this work has concentrated on the problems associated with parsing using such a connectionist network. However, there are several advantages to parsing in this computational architecture. Some of these advantages are manifested in the current parsing model. First, the biological motivations for the

S&A architecture provide independent motivations for its computational constraints, and therefore give them explanatory power. This architecture's relationship to the biological substrate of human language processing also gives work in this architecture predictive power, since the link between an abstract model and real time and space data can be made on the basis of independent biological evidence. NNEP is compatible with the real time constraints on language processing given this relationship, but work taking full advantage of this predictive power has only begun. The S&A architecture's use of massively parallel computation is also important for NNEP. Because NNEP's grammar is implemented with a set of pattern-action rules that compute in parallel, NNEP's speed is independent of the size of its grammar. This property is very important given the size of any real grammar for a natural language. Also, the ability to interpret the units of an S&A architecture at the level of symbolic computation makes it possible to structure the network to take advantage of known generalization about the task. In NNEP, different words whose grammar entries have the same effect on the parser state share the same grammar units. This means the number of units in NNEP's implementation is proportional to the number of different types of descriptions in the grammar, not the number of words. The number of different description types in a grammar is much smaller than the number of words. Also, the ability to capture such generalizations would be important for learning new words, since (in most cases) only link weights to existing units need to be learned. These characteristics are discussed further in section 7.3.

Most of the advantages of parsing using the S&A architecture will only be utilized in future work. By demonstrating the feasibility of syntactic parsing in the S&A architecture, the work done here justifies future work that uses this architecture to investigate issues for which connectionist networks are particularly well suited. In particular, previous connectionist investigations of grammar learning and ambiguity resolution have been hampered by representational inadequacies. With the above work, the success connectionist models have had on similar problems in other areas is likely to be repeated for natural language. The emphasis in this work on segmenting grammatical information into independent components (by kind of information and by phrase structure node) is likely to be significant in this investigation, since statistical independence is very important in controlling the complexity of the grammar induction task. These issues are discussed further (although still tentatively) in section 4.5.

1.5 Summary

This chapter has given an overview of this dissertation, which discusses syntactic parsing using a model of symbolic computation in a connectionist network recently proposed by Shastri and Ajjanagadde (Shastri and Ajjanagadde, 1993). This connectionist model of computation extends previous connectionist architectures by using temporal synchrony variable binding to represent the identities of entities in a way that allows rules to generalize over entities. Because of this added ability, the architecture can take advantage of the systematicity of natural language, while keeping the properties of connectionist networks which have made them important tools for cognitive modeling and applications. However, the S&A architecture has some limitations. These limitations can be characterized as a set of constraints on symbolic computation, thereby allowing the significance of using the S&A architecture to be studied at the same level of abstraction as other investigations of syntactic parsing. Most of these computational constraints have previously been proposed on the basis of linguistic and psychological evidence.

The primary claim of this dissertation is that the Shastri and Ajjanagadde connectionist computational architecture is adequate for recovering the syntactic constituent structure of natural language

sentences. In addition, it is claimed that the computational constraints of this architecture make significant predictions about the nature of language. To make these arguments, a specific parsing model has been implemented in the architecture which is designed to address the architecture's computational constraints. The central characteristic of this parser which allows it to comply with these constraints is its extensive use of partial descriptions of phrase structure trees. This parser has been tested on all the phenomena which are of particular concern given the limitations of the architecture (phrase structure analyses, long distance dependencies, local ambiguities, and center embedding). The results of these tests and a test on a random sample of sentences indicate that the S&A architecture is powerful enough for recovering the syntactic structure of natural language sentences, and that the computational constraints imposed by the architecture make some significant linguistic predictions. These predictions are mostly in the areas of long distance dependencies and center embedding.

In the remaining chapters of this dissertation, the argument given above will be made in more detail. The first four chapters investigate parsing in the S&A architecture. They start with Chapter 2, which characterizes computation in the S&A architecture in terms of symbolic computation under a set of constraints. Chapter 3 presents a grammatical formalism which is designed to help a parser comply with these constraints. Chapter 4 then uses this formalism as the grammatical framework for a specific parsing model which complies with the constraints. Chapter 5 describes how this parsing model has been implemented in the S&A architecture. The last two chapters use the parsing model developed in the previous chapters to demonstrate the adequacy and linguistic significance of the S&A architecture for syntactic parsing. First Chapter 6 tests the parser's ability to handle various natural language phenomena, and argues that some linguistic constraints are explained by the constraints from the architecture. Then Chapter 7 summarizes the computational and linguistic characteristics of the parsing model, and discusses their significance.

Chapter 2

The Computational Constraints

This dissertation discusses parsing in the Shastri and Ajjanagadde connectionist computational architecture. While the parser presented here has been implemented in the primitive computational devices provided by this architecture, that level of description is too specific to provide a useful basis for discussing the motivations for and implications of aspects of a parser's design. The level of abstraction which has been found to be the most useful for discussing parsing issues is that of symbolic computation. By symbolic I do not mean categorical. Many problems in parsing require the use of probabilistic information, continuous valued parameters, and soft constraints. By symbolic I mean the use of abstract representations of entities, their properties, and the generalizations about these properties. The prototypical symbolic representation is predicate calculus, which uses variables as abstract representations of entities, predicates as abstract representations of properties, and formulae as abstract representations of generalizations. One of the things that makes the S&A architecture so interesting for this investigation is that there is a clear mapping from the level of activations, units, and links to the level of variables, predicates, and formulae. Because of this relationship, the important characteristics of the architecture can be expressed at the level of symbolic computation, thereby allowing the investigation of parsing issues to be done at the appropriate level of abstraction. While it is important to make sure that the resulting parser design can be implemented at the level of units and links, the irrelevant details of this level should not be allowed to interfere with the investigation of parsing issues.

Symbolic computation by itself makes no assumptions about the computational architecture. If we made no such assumptions, then any desirable computational properties which we want would be available to us, but we would have no motivation for imposing any computational constraints. Because this investigation is about the implications of assuming the S&A architecture, it is the architecture's limitations, and not its desirable characteristics, which are our primary concern. Thus this chapter is primarily concerned with characterizing the computational constraints imposed by the S&A architecture. These constraints, plus the constraints imposed by the nature of the parsing problem itself, form the basis of this investigation. By demonstrating that these constraints do not prevent the syntactic parsing of natural language, we demonstrate that the S&A architecture is computationally adequate for this process. By demonstrating that these constraints make interesting predictions about the nature of language, we demonstrate that the S&A architecture makes such predictions. While a parser has been implemented in the primitives of the S&A architecture, most of the discussion will be at the level of constrained symbolic computation.

In order to further motivate the departure from nonsymbolic connectionist architectures, this chapter starts with a discussion of previous work on connectionist natural language parsing. The computational constraints imposed by the architectures used in these investigations are characterized, along with the implications of these constraints for parsing. None of these investigations have done the empirical investigations which would be necessary to argue that these constraints do not prevent such parsers from ever being adequate. After this motivation, this chapter discusses how computation in the Shastri & Ajjanagadde architecture can be characterized as symbolic computation under a set of computational constraints. Other proposals for doing symbolic computation in a connectionist network are also discussed, along with the advantages of the S&A model. The computational constraints imposed by the architecture are then combined with constraints imposed by the input to the parser and the needs of the modules which interpret the parser's output, to form the set of computational constraints on parsing to be investigated in this dissertation. Chapters 3 and 4 discuss how a parser can comply with these constraints, chapter 5 demonstrates that such a parser can indeed be implemented in the S&A architecture, and chapter 6 provides the empirical investigations necessary to show that the proposed techniques allow such a parser to be adequate for syntactic parsing.

2.1 Constraints on Previous Connectionist Parsers

Some previous investigations of connectionist natural language parsing have claimed that connectionist architectures are adequate for parsing, but they all use connectionist architectures which impose formidable constraints on any such parser. To date, none of these investigations have adequately addressed the problems which arise due to these constraints. The fundamental source of these difficulties is that these architectures do not distinguish between constituent identity information and constituent feature information. In symbolic representations, this distinction is made by using different variables to refer to different constituent identities, and using different predicates to refer to different constituent features. This distinction is important because these two types of information have very different properties. First, the set of features which a parser needs to represent during the course of a parse is fixed, while the set of constituents needs to be dynamically determined. Making use of this difference is crucial to a connectionist parser's ability to comply with its inherently bounded memory. Second, investigations of natural language have consistently found that it is important to express generalizations over constituents in terms of subsets of their features.[1] For example, adjectives can modify noun phrases, regardless of where the noun phrase occurs in the sentence. To express these generalizations it must be possible to determine whether a given constituent has a given set of features, without having to make use of other features to do so. This is the role that variables play in symbolic representations. They are used to represent which predicates apply to which entities, without themselves specifying any information about the entities. Thus the use of variables to express the bindings between different predications allows the expression of exactly the generalizations which need to be stated.

The reason the connectionist architectures used in previous investigations of connectionist parsing do not distinguish between constituent identity information and constituent feature information is that they represent these two types of information in the same way. These architectures use

[1] The existence of these kinds of generalizations is the property that Fodor and Pylyshyn (1988) call systematicity. It is directly related to compositionality in that, for representations to be productively constructed from separately specified representations, the later representations can't know everything about the resulting representation. Thus they must be able to specify only a subset of the features of the resulting representation. This can't be done if they have to uniquely identify the resulting representation.

different units to represent both different features and different constituents. Localist networks concentrate on representing what constituents are in the phrase structure tree, but (in the extreme case) they only represent one feature for each constituent. This one feature has a rather complex interpretation, but it is not decomposed in the network's representation. Distributed networks concentrate on representing what features the phrase structure tree has, but (in the extreme case) these features all apply to a single constituent, namely the whole tree. Again, that constituent may be interpreted as a rather complex structure, but that structure is not decomposed in the network's representation. In practice, networks mix these two dimensions of information, but because they don't distinguish between them, they have difficulty taking advantage of their different properties.

An early investigation of connectionist parsing was done by Cottrell in his dissertation on word sense disambiguation, published in (Cottrell, 1989).[2] While it was not the central concern of this work, he did implement a syntactic parser. This parser's grammatical framework uses context free grammars which include both syntactic category and semantic role information. The network uses a localist representation of this information. There are dedicated units for each possible category-role pair, and dedicated units for each list of roles which can form a constituent of a given category. There are multiple copies of each of these types of units, to allow for multiple constituents of the same type. While this representation specifies each of the context free productions used in the parse tree of the sentence, it is possible for this representation to be ambiguous as to how these pieces fit together in the complete structure. Cottrell tries to use the temporal order in which these units become active to represent which constituents have already been finished and bound to a position in the phrase structure tree, but he reports errors. He refers to (Fanty, 1985) for a solution to this problem. In addition to this problem, this parser does not represent features of constituents other than their major category. Representing these features would require the network to be intractably large. He also doesn't address the issue of reclaiming copies of constituent recognizers when the sentence has too many constituents of the same type. Since syntactic parsing is not Cottrell's major focus, he only tests his parser on its ability to handle simple sentences.

Fanty (1985) also investigated context free parsing, using the chart parsing method. He recognized that if the input is limited to sentences of less than a fixed length, then a fixed chart could be specified which includes the parses for all possible input sentences. He used a localist representation of this chart. A combination of bottom up and top down activation are used to determine which constituents in the chart are parts of possible parses, and winner-take-all networks are used to select among the possible parses. Because the chart shares common constituents between parses, this representation is more efficient than listing each possible parse separately. However, it still requires a very large network for sentences of reasonable length, and is inherently unable to deal with arbitrarily long sentences. The size of the network will also increase as finer distinctions are made between different nonterminal categories, which is exactly the problem mentioned above with representing features besides major categories.

Selman and Hirst (1987) also use a localist connectionist representation, and they also map the input sequence into space. Their network uses the Boltzmann machine method of connectionist computation, also known as simulated annealing. While this parser was the first to make systematic use of connectionist networks' ability to do evidential reasoning, it suffers from the same basic problems of the previous two. Because the input sequence is mapped onto space, the parser is inherently limited to sentences of a bounded length. Because a localist representation is being used, fine distinctions between constituent categories will make the network intractably large.

[2]The dissertation itself was finished in 1985, which is why I count this work as early, even though the book came out four years after Fanty's work (Fanty, 1985).

The remaining connectionist parsers to be discussed here solve the problem of representing the ordering of input words using recurrent links. These links put cycles in the spread of activation through the network, thereby allowing the network to store state information. Words are then input to the network sequentially, and the parser's state is used to represent how the previous words constrain the processing of the current word. In principle this should give these parsers the ability to parse arbitrarily long sentences, but in practice this ability has not been achieved. These parsers also differ from the previous ones in that they use distributed, rather than localist, connectionist representations. This allows them to express generalizations in terms of subsets of features, but these features tend not to generalize across embedded constituents. The researchers claim this as an advantage, because these networks take the "context" of each embedded constituent into consideration. However, it is the difficulty these networks have with abstracting away from irrelevant context information which prevents them from being adequate for parsing natural language.

Elman (1991) applies his work on simple recurrent networks to natural language parsing and gets reasonable results for simple sentences. Simple recurrent networks are a form of Parallel Distributed Processing (PDP) model (Rumelhardt and McClelland, 1986). They have a layer of input units, a layer of output units, and a layer of hidden units, but unlike plain PDP models, they have an additional set of input units whose values are the values of the hidden layer units after the last word was processed. An extension of the backpropagation learning algorithm is used to train the network to predict the subsequent word given the next word of the sentence and the previous state of the hidden units. He trains and tests the network on sentences with up to three embeddings of relative clauses. The results show that the network has learned to expect gaps in relative clauses, to expect verbs to agree with their extracted subjects, and to expect one clause to be completed once an embedded clause is finished. In addition, an inspection of the internal representations of the network reveals that there is some generalization from one level of embedding to another. However, this generalization is not sufficient to allow the network to parse large numbers of embeddings. People are not able to parse large numbers of embeddings when they are nested, but they have no trouble parsing arbitrarily many embeddings when they are not nested. While in theory a sufficiently large network like Elman's could learn to generalize in the way necessary for parsing arbitrarily long sentences, the work done so far does not provide evidence that this would be tractable.[3] In addition, the sentences which are used have a very restricted syntax, and a very small set of words. While he does address some linguistic phenomena (such as verb subcategorization, subject-verb agreement, and limited cases of wh- movement), it is the shear quantity of grammatical information that a parser must know which poses the greatest challenge to parser's implemented in Elman's architecture. Because these networks have a hard time abstracting away from contextual information in their representations, they will end up representing a much greater quantity of grammatical information than the already large quantity which is needed. The training sentences did not require any such contextual information, and yet the network still represented it. As Elman (1991, page 220) puts it "even when the network's behavior seems to ignore context ([...]), the internal representations reveal that contextual information is still retained." While Elman regards this property as an asset, it is likely to prevent such networks from being able to scale up from the simple linguistic domains which they have been tested on so far.

St. John and McClelland (1992) do not technically do parsing, since the input to the network is

[3]Servan-Schreiber, Cleeremans, and McClelland (1991) show that simple recurrent networks can be trained to mimic finite state automata. However, encoding a natural language grammar as a finite state automata would result in a huge automata, so these results do not demonstrate that training such a network would be tractable for natural language.

position-constituent pairs and not ordered words, but since they use terms like "sentence comprehension" their work will be discussed here. Their network takes constituents for up to four positions in a clause and maps them to a representation of the semantics of the clause. Another network is then used to decode the first network's output representation into semantic role-filler pairs. Both these components are PDP networks. The first component processes the constituents of the clause sequentially. It has input units for each word that can be in a constituent, for the four positions (preverbal, verbal, first-post-verbal, and other-post-verbal), and for the output representation produced by the last constituent. These input units are connected to a layer of hidden units, which are connected to a set of output units. The output units do not have a fixed interpretation. Instead, the second component of the system is used to translate whatever representation the first component comes up with into the fixed representation of semantic role-filler pairs. Along with the output units from the first component, the second component has input units for the semantic roles and fillers. The output of the second component is the role-filler pair for the input role or filler in the input representation. The two networks are trained together to produce the correct role-filler pairs for the input roles and fillers after processing each constituent of the sentence.

The only syntactic phenomena which St. John and McClelland (1992) tested was the ability to differentiate between active and passive sentences. In a domain of 10 names, 10 transitive verbs, "was", and "by", after 100,000 training sentences the network produced the correct result on 97% of the testing data. This test leads them to claim (page 115): "As demonstrated by learning the rule for the passive voice construction, it can learn syntactic regularities". The network was also trained on a slightly more complicated set of sentences which also included both active and passive sentences, but the result was not tested on its ability to generalize to new sentences. While it is understandable that they would need to restrict their attention to such simple single clause sentences given their desire to map (pregrouped) words directly to semantic structures, such demonstrations hardly merit the use of terms like "sentence comprehension". Indeed, the very intractability of doing more complicated testing is indicative of the intractability of doing broad coverage natural language parsing with the kind of system they propose. In addition, as they themselves say (page 118), their representation of both input constituents and output role-filler pairs are inadequate for sentences with multiple clauses. They propose using head-role-filler triples to solve the later problem, but even this is inadequate for sentences such as "John said that Mary said to go", where John and Mary have the same head and role. This problem is precisely the problem of representing constituency information, which distributed connectionist representations can only do by integrating it with feature information. The space of features necessary to map words to semantic representations is already so large that compiling constituency information into it would make training the network intractable. Solving this problem requires partitioning the task into multiple pieces, most likely along the lines found to be useful by researchers in symbolic natural language processing, namely separating constituent identity information from constituent feature information, and separating syntactic information from semantic information.

There are a number of other investigations which relate to connectionist parsing, but they do not directly address the issue of the adequacy of connectionist networks for natural language parsing. Some of these systems perform computations which are related to parsing, but use a more structured form of input than is available to a parser ((McClelland and Kawamoto, 1986), (Miikkulainen and Dyer, 1991)). Others use hybrid architectures, which are only part connectionist. Some of the later investigations may be significant for future work on the parsing model developed in this dissertation. These investigations complement this one, in that they investigate the strengths of connectionist networks, but not their weaknesses. Kempen and Vosse (1989) use a grammatical framework very similar to that used here (Segment Grammar, see section 3.3.3), and they use simulated

annealing to make attachment decisions. Their system successfully models several psycholinguistic results. Jain (1991) uses a combination of PDP networks and C code to parse spoken language. The system is hand structured to reflect knowledge about the nature of the task. PDP networks are used for the various stages of processing. These modules are trained separately to perform those computations that require the particular abilities of connectionist networks. Simple, well understood computations, such as transformations from the format appropriate for the output of one module to the format appropriate for the input to another module, are computed with C code. By taking this approach, Jain is able to demonstrate that connectionist networks generalize well from training examples, tolerate noise well, and can effectively use multi-modal input. These results should also apply to an entirely connectionist parser, such as the one developed here, as discussed in section 7.4. Stevenson (1994) makes use of activation decay and parallel constraint satisfaction in a hybrid connectionist/marker passing network to model a variety of psycholinguistic results. As a sentence is input, a network is constructed which encodes the ways the principles of Government-Binding theory apply to the sentence. Marker passing and parallel constraint satisfaction are used to converge on a single (partial) analysis. If evidence is received which contradicts the chosen analysis, the network can reconverge to a new analysis. Activation decay is used to model recency effects in both attachment and gap filling ambiguities. This research helps answer questions about the nature of the human parser's disambiguation mechanism, and should be applicable to future research on such a mechanism for the parser developed here.

2.2 Connectionist Symbolic Computation

The Shastri and Ajjanagadde connectionist architecture differs from those used in previous investigations of connectionist parsing in that it supports symbolic computation. Symbolic computation involves abstract representations of entities and their properties. For our purposes, the entities are the constituents of the phrase structure of the sentence, and the properties are the features of those constituents. In order to allow the representation of constituent identities to be factored from the representation of constituent features, the S&A architecture uses the fine grained temporal synchrony of unit activation to represent the binding between different properties of the same entity. This mechanism is called temporal synchrony variable binding, and it is the core feature of the S&A architecture. This mechanism also allows the set of constituents to be dynamically specified. Other connectionist architectures have been proposed for representing the bindings associating properties with entities, but none of them have all the properties which make the S&A architecture so well suited for this investigation. In addition to supporting symbolic computation, the S&A architecture supports the massively parallel use of knowledge, supports evidential reasoning, is biologically plausible, and has psychologically plausible limitations. The limitations of this architecture are the primary computational constraints to be investigated in this dissertation. The architecture has a bounded memory capacity, no explicit representation of logical connectives, and in the general case has difficulty in representing, and computing with, relationships between entities.

2.2.1 Temporal Synchrony Variable Binding

In order to support symbolic computation, an architecture needs to be able to represent the truth values of multiple predicates for multiple entities. The architectures used in previous investigations of connectionist parsing have not been able to do this effectively because they only have two dimensions for representing state information, unit activation and unit identity. The key insight of

the S&A architecture is that in addition to activation and identity, neurons have available a third dimension for representing information. When a neuron is active, it is not simply on, it pulses. A pulse has an activation, a neuron identity, and a time. This temporal dimension is what is used to provide the third dimension which is needed for symbolic representations. Activation levels are used to represent truth values,[4] unit identities are used to represent predicates, and the time of a unit's activation pulse is used to represent which entity the predicate is true of. Typically, symbolic representations use variables to represent which entity a predicate is true of. The identity of a variable does not itself specify any information about the entity, but it serves to bind together the different predicates which are true of the entity. This process is called variable binding. The S&A architecture uses the temporal synchrony of unit activation pulses to represent these bindings. If two units are firing synchronously, then they are representing predications over the same entity, and if they are firing out of phase, then they are representing predications over potentially different entities. This technique is called temporal synchrony variable binding, and it is the core feature of the S&A architecture.

Figure 2.2 illustrates how temporal synchrony variable binding is used to represent information about the phrase structure of a sentence. The predicates listed on the left of the figure are represented with sets of units. The collective output of these units are shown in the timing diagram. The left side of the timing diagram (labeled "step 1") shows the representation of the first parser state shown in figure 2.1. In the first time slice, units are active for all the predicates which are true of the node labeled x in figure 2.1, namely the root node of the sentence. Similarly, the second and third time slices represent the bindings between predications that are represented with the variables y and z in figure 2.1, respectively.

All of these predicates in figure 2.2 are unary predicates, except the last one (*complete*), which is a constant predicate. This last predicate does not apply to individual nodes, but to the phrase structure description as a whole. Thus its output is constant across all the time slices. Binary (or higher arity) predicates can be represented using multiple pairs (or triples, etc.) of unary predicates. These unary predicates represent which nodes are in the different roles of the binary predicate, and the fixed pairings between the unary predicates represent which nodes in the first role go with which nodes in the second role. Just as variables only represent the bindings between predications, the different pairs do not represent any information in and of themselves, but only the bindings between the different entities involved in the relation. In the general case for binary predicates, this method requires as many pairs of unary predicates as there are variables in the memory. For higher arity predicates, it requires as many tuples as the number of variables times one minus the arity of the predicate.

The information represented with temporal synchrony needs to be maintained across the steps of a computation. To simplify the maintenance of synchronous firing, units will be assumed to fire periodically at a fixed frequency. For example, the activation shown in figure 2.2 for the predicate $+I$ is the activation of a single unit. This assumption means that the activation of these units can be described with a phase and a range of periods. Just as a time slice represents a variable within a period, a phase represents a variable across periods. Thus figure 2.2 labels the different time slices for the same phase with the same variable name.

Temporal synchrony variable binding uses fine grained temporal distinctions to represent variable bindings, which leaves course grained temporal distinctions to represent the sequence of steps in a computation. The architecture cycles through variables, and each period of this cycle is

[4] Actually, activation level will be used to represent the probability of (or evidence for) a predicate being true. In the categorical case this reduces to the truth value itself.

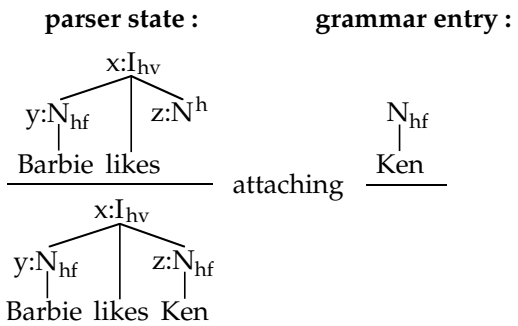


Figure 2.1: The parser combination action whose processing is shown in figure 2.2. The grammar entry for "Ken" is added to the parser state as the object of "likes".

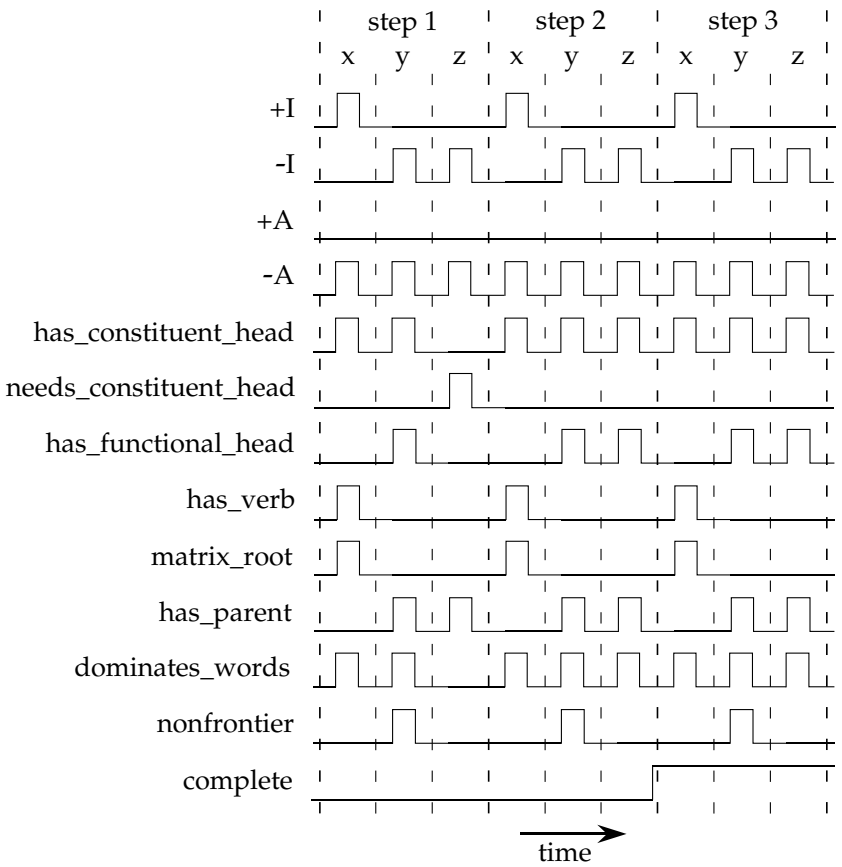


Figure 2.2: An example of how temporal synchrony variable binding supports computation. New information about z is set in step 1, the information begins being output in step 2, and this results in the *complete* predicate being true of the whole tree in step 3.

analogous to a computation step. Thus figure 2.2 labels the different periods with different step numbers.[5] The information represented in the temporal pattern of activation can be modified from one period to another using links between units. As in other connectionist architectures, activation flowing across links produce new patterns of activation from old ones. Patterns of links have the effect of testing the pattern of activation, and modifying it if a match is found. Thus links implement pattern-action rules. Because the architecture cycles through variables, the links are time-multiplexed across variables, and thus their pattern-action rules inherently generalize across variables. For the same reason, links only have access to information about one variable at a time. However, links also have access to information about the structure as a whole, which is represented with constant predicates. Pattern-action rules which involve more than one variable can be implemented by setting and testing these constant predications. Thus constant predications are used to communicate information about one variable or set of variable to all the other variables. This allows any pattern-action rule to be implemented, but some rules are significantly harder to implement than others.

Figure 2.2 shows the effects of the rule that implements the parser action shown in figure 2.1. The pattern of this rule matched in the phase of z, and in "step 1" the rule's action sends activation to the units which represent the predicates which are newly true (or false) about z. In the subsequent period (step 2) these units begin outputting activation in this phase, thereby representing the new information about z which resulted from the parser action. After this new state of the parser is represented in the parser's memory, another rule detects that none of the nodes in the tree have any more expectations that need fulfilling. In step 2 this rule sends activation to the unit for constant predicate *complete*, which starts being active in step 3.

2.2.2 Other Characteristics of the S&A Architecture

Shastri and Ajjanagadde (1993) give a number of important characteristics of their architecture other than its ability to do symbolic computation. Perhaps most importantly, the architecture supports the massively parallel use of knowledge. This property is crucial for parsing because of the huge size of any real grammar for a natural language. In order to parse with the incredible speed necessary for real time speech understanding, the human parser must access this knowledge in a massively parallel fashion. The parsing model developed here takes advantage of this ability. The parser's speed is independent of the size of its grammar.

Another crucial property for natural language processing is the ability to do evidential reasoning. Problems such as lexical disambiguation, syntactic disambiguation, thematic role disambiguation, pronoun reference, filling in implicit arguments, and others require representing and reasoning with probabilistic information, continuous valued parameters, and soft constraints. Connectionist networks are good at this kind of reasoning, because link weights, unit thresholds, and activation levels can all be continuous valued. Thus both long term and short term knowledge can be probabilistic or soft. Long term knowledge includes parsing rules and grammatical information. Here, short term knowledge is the information about the phrase structure of the sentence. It is expected that these abilities will be very important in the long range success of parsing models such as the one developed in this dissertation. This dissertation takes a more categorical approach to parsing issues in order to simplify the investigation, and because the ability of connectionist networks to deal with

[5] The choice of x as the start of a period is arbitrary. It could have been y or z. In this sense there aren't really discrete steps in the computation. The key point is that temporal distinctions at the resolution of periods correspond to distinctions between steps in the sequence of computation.

soft constraints is fairly uncontroversial, while the ability of connectionist networks to deal with the issues associated with symbolic processing is much more controversial.

Although they are less directly relevant to the study of language, two other characteristics of this architecture help motivate adopting it. First, there is a plausible and fairly direct relationship between the primitives of the architecture and neurons. Because of this relationship, the work in this dissertation can be viewed as part of an effort to explain how higher level linguistic generalizations are manifested in biological mechanisms, although the scope of such a study goes far beyond what is discussed here. For a detailed argument for the neurological plausibility of this architecture see (Shastri and Ajjanagadde, 1993). That document also argues for the psychological plausibility of the architecture's various limitations. The limitations which are relevant for syntactic parsing will be discussed in section 2.2.4..

2.2.3 Other Models of Connectionist Symbolic Computation

Other models have been proposed for doing symbolic computation in a connectionist network, but none of them have all the advantages of the Shastri & Ajjanagadde architecture. Each of these solutions can be characterized in terms of what physical dimensions they use for the various representational dimensions needed for symbolic computation. The physical dimensions of a connectionist network are time, unit identity (i.e. space), and activation level. All these solutions use different unit identities to represent different properties, those that consider sequential computation use time to represent the different stages in the computation, and those that consider evidential reasoning use activation level to represent different degrees of evidence. Where these proposals differ is in how they represent entities. Tensor product variable binding (Smolensky, 1990) and relative-position encoding (Barnden and Srinivas, 1991) use the space dimension; signatures (Lange and Dyer, 1989), CONSYDERR (Sun, 1992), and pattern-similarity association (Barnden and Srinivas, 1991) use the activation level dimension; and temporal synchrony variable binding (Shastri and Ajjanagadde, 1993) uses the time dimension. In each of these architectures, the space dimension is used to represent whatever distinctions remain after entities have been represented.

Smolensky's work on tensor product variable binding formalizes an approach to representation that has been used in a number of connectionist investigations (see (Smolensky, 1990)). Tensor product variable binding organizes the space dimension into multiple representational dimensions, one for properties, and the rest for entities. In general, representing properties which have multiple roles requires as many additional representational dimensions as there are roles. Thus tensor product variable binding uses as many dimensions for entities as the maximum number of roles for any property (i.e. the maximum arity of any predicate). Barnden and Srinivas (1991) generalize this method by using relative spatial information, rather than absolute spatial information, and call this form of representation relative-position encoding. As discussed in section 2.1, the use of space to represent different entities has the problem that links can't help but be sensitive to spatial distinctions, and thus rules can't generalize across entities. Such generalizations must be explicitly encoded using multiple sets of links. This poses a problem for connectionist learning algorithms, which are spatially local unless an external mechanism is used to enforce the equality of weights.

Several researchers have proposed variable binding mechanisms that use activation level to represent entities. Lange and Dyer (1989) call different activation levels "signatures", and use them to represent variable bindings. Sun uses the same method in his CONSYDERR system (1992). Barnden and Srinivas (1991) call this form of representation pattern-similarity association. In contrast to the way space is used to represent entities, none of these proposals compile both degree

of evidence and variable binding into the same activation levels. Because these two dimensions would have to be separated to be interpreted, such a representation would require more complex computation by units than most researchers are willing to assume. Instead, some units' activation levels represent degree of evidence, and others' represent variable bindings. This requires the use of spatial associations to represent which degrees of evidence are for which variable bindings, and it requires two parallel computation paths for the two kinds of information. The need to coordinate computation at these two levels complicates the network significantly. Otherwise, the ability of links and unit computations to generalize over activation levels makes this representation similar to temporal synchrony variable binding. Each of the above researchers recognize the similarity between the two approaches, and suggests that temporal synchrony might be an alternative implementation for the variable binding technique they propose. The advantage of using time rather than activation level to represent variable bindings is that compiling both stage in the sequential computation and variable binding into the time dimension does not require very complex unit computations. Units simply need to be sensitive to both relative phase (precisely synchronous or not) and relative period (occurring within a certain time or not). By representing both degree of evidence and variable binding in the temporal pattern of activation, a single level of the network can process all the relevant information, and there is no need to coordinate computation at different levels.

As mentioned in the discussion of tensor produce variable binding, in general adding a single additional representational dimension in not adequate. Representing properties that have multiple roles requires as many additional dimensions as there are roles. Rather than having a different representational dimension for each role's variable binding (as in tensor product variable binding), temporal synchrony variable binding represents each role's variable binding separately using the time dimension, and then uses space to represent the different groupings of these bindings into propositions. While this doesn't reduce the size of the representational space in principle, in practice the number of such groupings that need to be stored at a given time is rather small. The use of the space dimension to represent these groupings means that rules do not generalize across different groupings, but in practice this is also not a problem. These two claims will be made more precise in the next section, and, for the case of syntactic parsing, will be argued for in the rest of this dissertation.

2.2.4 Limitations of the S&A Architecture

Using temporal synchrony variable binding, the S&A architecture supports variables, predicates, and pattern-action rules. This allows the abstract representation of multiple entities and their properties, and the statement of generalizations over entities in terms of arbitrary subsets of these properties. These abilities provide a rather general purpose framework for symbolic computation, but the architecture does have some limitations. By making clear the relationship between symbolic computation and computation in the S&A architecture, the above analysis allows these limitations to be characterized and analyzed in terms of a set of computational constraints which are conceptually distinct from the mechanisms of symbolic computation. In this way our investigation of parsing natural language can capture the abstract generalizations about the nature of language, and still express the exceptions imposed by the computational abilities of the architecture. Although it is the generalizations plus the exceptions that determine the ultimate interpretation of the parser's representations, making this conceptual distinction allows this investigation to incorporate results from investigations which abstract away from computational constraints, which make up the majority of the work on natural language. That said, it is the implications of the computational constraints

that are the primary concern of this investigation, and not the instantiation of any particular theory of linguistic competence. This section continues the analysis of the relationship between the S&A architecture and symbolic computation by characterizing the set of computational constraints which it's limitations impose.

One important limitation of the S&A architecture is that the memory capacity of any module is bounded. Neurons are only capable of sustaining periodic firing for periods within a certain range, so the length of a period is bounded. Also, the ability of a neuron to distinguish between synchronous and asynchronous input pulses has fixed precision. Thus there is a bound on the maximum number of distinguishable phases that can fit in one period, and thus a bound on the number of variables that a module can store information about at any one time.[6] Biological evidence on the maximum period length and the width of an activation spike can be used to estimate this bound. Shastri and Ajjanagadde (1993) do this estimation and determine that the bound on the number of variables is at most ten, probably a little less. For the data addressed in this dissertation, any bound of nine or greater would be fine, but to avoid the impression that I am making a more precise claim than I am warranted in making, I will assume this bound to be ten. Some additional linguistic data needs to be found and addressed before a precise bound can be determined on that basis. Nevertheless, it is interesting to have this degree of agreement between the biological data, the linguistic data, and the robust bound on human short term memory of seven plus or minus two (Miller, 1956).

Another important limitation of the way the architecture stores information is that there is no explicit representation of logical connectives. Thus only the default logical connective can be used in the representation of phrase structure information. Either conjunction or disjunction could be used as the default connective, but using disjunction would require a very large number of primitive predicates. Conjunction is used as the default connective so that feature decompositions can be used to represent node types. Representing categories as a conjoined set of more primitive features is a common practice in grammatical representations. This form of representation has also been widely adopted in connectionist work, where it is called a distributed representation.[7] Such representations have been found to be quite effective in capturing relevant similarities between the properties of different entities. Because only one logical connective can be the default, the use of conjunction means that the parser cannot explicitly represent a disjunction of predications. This constraint on parsing has been proposed previously and was argued for on the basis of linguistic evidence. Marcus (1980) proposed that natural language parsing can be done deterministically, in the sense that the parser does not have to simulate a nondeterministic machine. Marcus (1980) proposed a parsing model which complied with this constraint, and showed that it could handle a variety of linguistic phenomena. The model also predicts some difficulties with disambiguation which people have. There are two techniques which can be used to simulate a nondeterministic machine: pursuing multiple analyses in parallel, and using backtracking to pursue multiple analyses serially. The prohibition against pursuing multiple analyses in parallel is equivalent to not allowing the use of disjunction in the representation of phrase structure information. As will be discussed in the next section, the prohibition against backtracking falls out of constraints on the parser's output. Thus these constraints together require that the parsing model developed here be deterministic. This is another interesting convergence between previously proposed computational constraints and the limitations of the S&A architecture.

[6] The architecture allows for many different computational modules. Storing information in the constituent structure parsing module does not use up resources in other modules.

[7] Normally distributed representations use a feature decomposition which is produced by a learning algorithm, and which does not correspond to the feature decompositions which have been manually devised by other investigators. I take this to be an orthogonal issue to the distinction between using an atomic predicate or a conjunction of predications to describe an entity.

The remaining limitations of the S&A architecture are due to the fact that temporal synchrony variable binding only adds one additional dimension for representing information. This dimension (phase) allows the efficient storing and processing of information about individual variables, but relations between variables require additional representational dimensions for each additional argument position in the relation. As discussed in the last section, the solution to the need for additional representational dimensions is to use the dimensions which are available more than once. Tensor product variable binding explicitly uses the space dimension more than once, and most other connectionist architectures also use this method, although there may not be any clear mapping from units to points in the multidimensional space. Although the S&A architecture uses time instead of space for the representation of information about individual variables, it uses space more than once for representing relations between variables. This means that the S&A architecture has the same problems with storing and processing relations between variables as traditional architectures have with storing and processing all predications over variables. These problems are characterized in two additional constraints imposed by the S&A architecture, one on storing relations, and one on processing relations.

The problem with storing relations between variables using multiple spatial dimensions is that it requires more units and it adds complexity in modifying and accessing the stored relations. As discussed in section 2.2.1, relations can be stored in the S&A architecture using multiple tuples of unary predicates, where the different unary predicates represent the different roles of the relation and the different tuples represent which entities in the roles are related to which entities in the other roles. Because the different tuples do not represent any information in and of themselves, the allocation of tuples can't be determined by features of the entities, but must be determined by specific control structures. To control the complexity of these control structures and to control the number of units required to implement the multiple unary predicates, the implementation of relations is constrained to use at most three tuples of unary predicates. This is essentially equivalent to the constraint proposed by Shastri and Ajjanagadde (1993), who require that at most three instantiations of a relation can be stored at any one time. It is slightly different in that multiple instances of a relation can be stored in a single tuple, as long as all the entities in each role are related to all the entities in each other role, but in the case where no such distributivity holds it reduces to the same constraint. For this investigation only binary predicates will be needed, which in the general case would require ten pairs of unary predicates, since ten is the maximum number of variables in the memory.

Processing with relations between variables is difficult because such rules involve more than one entity. As discussed in section 2.2.1, since rules only have access to information about one entity at a time, information that the rule needs about other entities must be represented as properties of the structure as a whole. If the identity of the other entities isn't important, then this is not a particular problem. Such entities appear either only in the antecedent or only in the consequent of the rule. If the identity of another entity is important, then to avoid losing the identity information in the conversion to information about the structure as a whole, the other entity must be uniquely identifiable at the time when the rule applies. Thus it must be possible to refer to such entities with constants, rather than variables. Thus rules are constrained to only use at most one variable that appears in both the antecedent and consequent of the rule. It is possible to simulate a rule which involves multiple variables by introducing arbitrary distinctions between entities, but this requires multiple copies of the same rule. Introducing completely arbitrary distinctions is undesirable, and the duplication of the rule violates the locality assumptions of connectionist learning algorithms. It would also be possible to time-multiplex rules across the tuples which store relations, as is done with individual variables. For a rule involving a single relation, the rule's application would take

up to three times as long, since three is the maximum number of tuples. This would slow down the system and result in additional complexity for sequencing through the tuples. These additional costs turn out not to be necessary for syntactic parsing. While the constraint that at most one variable appear in the antecedent and consequent of a rule has no precedent in work on syntactic parsing, this investigation shows that it has a number of significant linguistic implications.

The locality constraint on rules just discussed is stated in a different form from the locality constraint on rules given in chapter 1, but the class of computations that are allowed is the same. In chapter 1, this constraint was stated as requiring that all rules apply to each variable independently, but accessing and setting relations was allowed, as long as this was done through unary predicates. The above form of the constraint allows the accessing and setting of relations to be incorporated into the notation for rules. This makes it easier to write down rules, but it glosses over a cost associated with such rules that should be kept in mind. The core computation in these rules takes place in the phase of the one variable that occurs in both the antecedent and consequent, and this computation is independent of information about other variables. The remaining computations necessary to implement such rules take place independently in the phases of the other phrase structure nodes mentioned in the rule, and they communicate with the core computation using predications about the structure as a whole (i.e. constant predications). While there is no theoretical limit to the number of constant predicates that can be used to coordinate computation between variables, each such predicate and the rules necessary to manipulate it does constitute a cost in both network size and complexity. Also, learning these predicates and computations is relatively complicated compared to unary predicates and computations over them, since they involve information about multiple nodes, and thus the space of possible computations is much larger. Thus, although it is not a strict constraint, this investigation will be concerned with minimizing the number of constant predicates.

The above discussion concludes the characterization of computation in the S&A architecture. The S&A architecture performs symbolic computation using variables, predicates, (probabilities of) truth values, and pattern-action rules, under the following set of computational constraints.

1. information about at most ten entities stored at a time
2. no explicit representation of disjunction between predications
3. at most three tuples of unary predicates for storing relations
4. at most one variable can appear in both a rule's antecedent and consequent

2.3 Constraints on the Parsing Model

As discussed above, the most effective level of abstraction for investigating natural language parsing is that of symbolic computation. In order to investigate parsing in the S&A architecture at the level of symbolic computation, the nature of computation in this architecture needs to be expressed at this level. The previous section did this by specifying a set of constraints on the architecture's ability to represent and reason with variables, predicates, truth values and rules. To complete the set of requirements on a syntactic parser implemented in the S&A architecture, we have to consider the constraints imposed by the nature of the task of recovering the constituent structure of a sentence. This section discusses the constraints imposed by the nature of the input to the parser and by the requirements of the modules which receive the output of the parser, then it summarizes these constraints and the constraints from the previous section.

The nature of the input to the parser imposes two constraints on the parser design. First, the words which are input to the parser become available one at a time, in the order in which they appear in the sentence. Thus the parser must accept incremental input. Second, the time between the input of each word is bounded, as is the time between the input of the last word and the completion of the parse. This is certainly true for speech, and in the normal case is also true in reading. Thus the parser must only take a bounded amount of time per word. In other words, the parser must parse in quasi-real time. The actual bound on the amount of time the parser should take can be determined, but in order to test compliance with this constraint there has to be a clear relationship between the number of computation steps performed at the level of symbolic processing and the number of seconds required at the level of biological processing. The relationship between the primitives of the S&A architecture and neurons provides a connection to biological processing, and biological evidence provides an estimate of the speed of this processing, but both these steps are rather approximate. Because precise estimates of the real speed of a biological implementation of a parsing model can't be made at this stage of our understanding, I will concentrate on the clearer quasi-real time constraint, rather than the stricter real time constraint. However, even the imprecise estimates which can be made provide some guidance for the design of the parser. In chapter 5 it is argued that the parser developed below is consistent with the real time constraint, given what we do know about biological processing. As our understanding of biological processing improves, these estimates will become more precise, and future work can use them to confirm and refute many aspects of parsing models.

The modules which receive the output of the parser need to incrementally compute the sentence's interpretation. Thus the output of the parser needs to be as incremental as possible, and immediately interpretable. To be as incremental as possible, the parser needs to output everything it thinks it knows about the phrase structure of the sentence. To be immediately interpretable, the output has to be monotonic. If the output isn't monotonic, then the interpreter can't make commitments on the basis of the output without risking having to retract those commitments. While such retractions do occur under some circumstances, I assume that there is always some evidence that something has gone wrong. Typically the person will be consciously aware of a problem, although other evidence (such as regressions in eye movements) can also be used to determine these cases. Since we are concerned here with the normal case in which nothing goes wrong, the parser's output must be monotonic. The requirement that the parser produce incremental monotonic output prevents it from backtracking. Once the parser determines something about the phrase structure of the sentence, it must be output, and once information is output, it can't be retracted. Thus the parser can't backtrack without something going wrong. As discussed in section 2.2.4, the combination of this prohibition against backtracking and the inability to use disjunction in the representation of phrase structure information implies that the parser must be deterministic. Thus Marcus' (1980) determinism constraint can be derived from the requirements on the parser's output and the limitations of the S&A architecture.

In summary, the design of a parsing model implemented in the S&A architecture needs to comply with four constraints from the architecture, two constraints from the input, and two constraints from the modules which interpret its output. This set of eight computational constraints, listed below, can now be used to investigate the feasibility and implications of using the S&A architecture for constituent structure parsing.

1. information about at most ten nodes stored at a time
2. no explicit representation of disjunction between predications
3. at most three tuples of unary predicates for storing relations
4. at most one variable can appear in both a rule's antecedent and consequent
5. input is incremental
6. parse in quasi-real time
7. produce maximally incremental output
8. produce monotonic output

Chapter 3

The Grammatical Framework

The problem of syntactic parsing can be divided into two parts, characterizing what syntactic structure (or structures) should be produced for each sentence, and characterizing how that syntactic structure will be produced. The first part is the grammar, and the second part is the parsing algorithm. Both these parts need a notation for representing information about syntactic structure. The formal characterization of this notation is called a grammar formalism. The choice of a notation is important both for specifying a grammar and for specifying a parser's data structures. Because the constraints identified in the last chapter are computational in nature, the grammar formalism that will be used is designed primarily to be good for representing a parser's data structures.[1] This is in contrast to most work on grammar formalisms, which are designed primarily to be good for representing grammars. Fortunately, these concerns are similar enough that there is a significant amount of overlap in the desired properties for a grammar formalism. Nonetheless, the grammar formalisms which are most similar to the one used here are those that have taken into consideration the concerns of parsing.

To investigate the implications of the computational constraints derived in the last chapter, the grammar formalism needs to provide the parser with the expressive abilities that it needs to comply with the constraints. The grammar formalism presented in this chapter (Structure Unification Grammar) does this using partial descriptions of phrase structure trees. This makes SUG similar to other unification based or constraint based formalisms, but none of these formalisms have all the properties which are needed to comply with the computational constraints. In particular, SUG allows each grammatical feature, expectation, iteration restriction, or structural constraint to be specified independently, SUG has very flexible derivation structures, and there is a simple abstraction operation ("forgetting") for SUG which allows unneeded information to be forgotten. The parser defined in the next chapter will use SUG plus the abstraction operation as its grammatical framework.

To simplify the analysis of the implications of the computational constraints derived in the last chapter, the grammar formalism should not impose any additional constraints. This means we want a relatively unconstrained grammar formalism at this stage in the research. Once the implications of the constraints have been determined, the grammar formalism should manifest all the constraints that can be simply stated declaratively. This is because the empirically testable interpretation of

[1] For this same reason, the discussion of grammars in this dissertation will be minimal. The analyses given in this chapter are only for illustrative purposes. Chapter 6 discusses how some results from the investigation of grammars can be used by the parser presented in the next chapter. What significance the results from this investigation do have for grammatical theory will also be discussed in chapter 6.

a grammar is the interpretation provided to it by the parser, not the interpretation provided to it by the grammar formalism. Thus a grammar writer who (or which) uses the grammar formalism's interpretation is being deceived about the true interpretation of their grammar. Sometimes this imprecision is justified because it greatly simplifies the grammar formalism, but when the difference can be specified declaratively, it should be included. This redefinition of the grammar formalism has not yet been done, but the differences are not very significant. Mostly the differences are in the range of grammars which can be defined by the formalism, versus the range that can be parsed. These differences will be discussed in the next chapter.

The rest of this chapter begins with a discussion of what properties the grammatical framework used in this investigation should have, and what existing approaches to grammatical representation have some of these properties. The second section describes a grammar formalism (SUG) which is specifically designed to comply with all the requirements, and which will be used for grammatical representations in this dissertation. The third section discusses how SUG can be used to express grammatical information. For a more extensive discussion of expressing grammatical constraints in SUG, including formal and informal comparisons with a variety of other investigations into natural language, see (Henderson, 1990). The fourth section defines the abstraction operation that can be used with SUG. This operation allows information to be forgotten without risking the violation of the forgotten information. The combination of SUG and this abstraction operation is the grammatical framework which will be used in the next chapter to specify the parser's grammars and how the parser recovers the syntactic structure of natural language sentences.

3.1 Related Approaches to Grammatical Representation

In the last chapter, computation in the Shastri and Ajjanagadde architecture was characterized as symbolic computation under a set of constraints. This allows the investigation of parsing using the S&A architecture to be done at the level of symbolic computation. Many symbolic grammatical frameworks have been developed that can be used for parsing, but the particular set of constraints we are investigating place important requirements on the grammatical framework which should be used here. This section characterizes these requirements, and discusses previous work on grammatical representation that has addressed these requirements. The grammatical framework presented in the next section and used in later chapters synthesizes ideas from all of these approaches.

3.1.1 Requirements for the Grammatical Framework

Chapter 2 identified the following set of constraints on parsing in the S&A architecture.

1. information about at most ten nodes stored at a time
2. no explicit representation of disjunction between predications
3. at most three tuples of unary predicates for storing relations
4. at most one variable can appear in both a rule's antecedent and consequent
5. input is incremental
6. parse in quasi-real time
7. produce maximally incremental output
8. produce monotonic output

As was discussed, constraints 2, 7, and 8 mean the parser must be deterministic, in the sense of (Marcus, 1980). Also, some effort should be made to minimize the use of rules that involve multiple phrase structure nodes.

Because the parser must be deterministic, the representation should allow the parser to avoid saying what it doesn't know. Following Description Theory (Marcus *et al.*, 1983), partial descriptions of phrase structure trees are used to satisfy this requirement. Partial descriptions allow the parser to underspecify phrase structure information, rather than either overcommitting or using a disjunction of more completely specified alternatives. For example, "she" has nominative case, and "her" has accusative case, but "Barbie" is ambiguous as to its case. Rather than having two different grammar entries for "Barbie", one for each case, partial descriptions allow a single grammar entry which simply does not specify the case. On the other hand, in order to produce incremental output and only allow syntactically well-formed analyses, the parser must be able to say what it does know. Again the use of partial descriptions is important for this requirement, because they allow different kinds of information to be specified independently of each other. In the same example, there is no problem with stating the "Barbie" is a noun, even though all nouns have case, and we don't yet know "Barbie"'s case. To satisfy both these requirements, the grammatical representation must allow information which the parser does know at a given time to be specified independently of the information which the parser does not know. What the basic units of grammatical information are such that this independent specification can be done is an empirical issue, but previous investigations of this issue give us a good idea about the nature of these representational primitives.

The locality constraint on rules (constraint number 4 above) and the parser's bounded memory both place another requirement on the parser's representation of grammatical information. Because of the locality constraint on rules, the representation should allow as much information as possible to be local to individual phrase structure nodes. This helps avoid the need for computations that manipulate pairs of nodes. It also minimizes the need for computations which involve multiple nodes. Thus we want a relatively flat phrase structure representation, provided it still expresses the compositional nature of syntax. This compact representation also makes it easier to stay within the parser's bounded memory, because it reduces the number nodes in a tree's representation. Previous investigations of grammatical representation give us a good idea about what domains in their phrase structure representations have large amounts of dynamically interacting information, even though they tend not to use the compact representation needed here.

The locality constraint on rules and the parser's bounded memory interact in another interesting way to constrain the parser's representations. Not only should as much information as possible be local to individual nodes, as little information as possible should be expressed as relationships between nodes. This minimization also helps compliance with the constraint on the number of instantiations of a relation that can be stored. As will be argued in chapter 6, each of the few parsing issues that actually require the use of relations can be handled with special mechanisms that avoid the use of rules that manipulate pairs of nodes. Isolating the few cases when relations are required also makes it possible to stay within the parser's bounded memory. Given that the use of relations has been minimized, the only information represented in relations is about how a change in the information about one phrase structure node affects future computations involving another node. Thus if no future computations are going to be necessary for a node, then relations with it are no longer needed. Information specifically about a node also isn't needed if there will be no more computations directly involving it, so under these circumstances all the information about the node can safely be forgotten. Thus such nodes can be removed from the parser's memory without risking contradicting the information about these nodes. By removing nodes as they are completed during a parse, the parser can parse arbitrarily long sentences using only a bounded

number of nodes at any given time, as is required by the S&A architecture's bounded memory. One particular class of grammar formalisms uses representations that stay bounded in size even for unbounded input (Categorial Grammar based formalisms), and these formalisms show what relations can be used to allow this property.

Because this investigation is interested in the implications of precisely the above set of constraints, the grammatical framework should not impose any additional constraints on either the representation or processing of grammatical information. This ensures that the limitations of the parser that is developed in this framework are due to the constraints, and not due to the framework. By using a very flexible representation, the parser can use whatever strategies it needs to compensate for the constraints, without having to use representations that are outside the framework. Unfortunately, this approach is in contrast to most work on grammar formalisms, which seeks the most restricted representation that still allows the specification of the range of phenomena in natural language. As discussed above, developing such a tight match between the formalism power and the empirically required power will have to wait for later investigations. However, previous work on grammar formalisms does point to certain expressive abilities the grammatical framework should have. In particular, work on Tree Adjoining Grammar (Joshi, 1987) shows that the grammar formalism should allow grammar entries which specify grammatical information over a fairly large domain of the tree. This domain is called the formalism's domain of locality, and it should include the ability to specify information across multiple levels in the phrase structure tree, and should include the ability to specify long distance dependencies.

3.1.2 Specifying Independent Information Independently

Grammatical representations that allow partial information from different sources to be specified independently and then combined in the final representation have been studied by quite a number of researchers. Such representations are often called unification-based or constraint-based grammar formalisms. They include Description Theory (Marcus, Hindle, and Fleck, 1983), Head-Driven Phrase Structure Grammar (Pollard and Sag, 1987), PATR-II (Shieber, 1986), Feature Structure Based Tree Adjoining Grammars (Vijay-Shanker, 1987), the system defined in (Rounds and Manaster-Ramer, 1987), Construction Grammar (Fillmore *et al.*, 1988), and Segment Grammar (de Smedt and Kempen, 1991). Shieber (1986) discusses other such formalisms. In this section, Head-Driven Phrase Structure Grammar and Description Theory will be discussed, because they embody the features to be developed in the discussion of the grammar formalism used in this work (Structure Unification Grammar, (Henderson, 1990)). After the presentation of Structure Unification Grammar, section 3.3.3 discusses most of the other formalisms listed above.

Of the well-developed linguistic theories, Head-Driven Phrase Structure Grammar (HPSG) most directly embodies the unification-based approach. The data structures of the theory are all feature structures, and unification is the central operation for combining these data structures. Some parts of these feature structures, such as the SUBCAT list, have special interpretations by virtue of a small set of universal principles, but these can be interpreted as notational mechanisms that allow the specification of constraints, such as linear order, that are not directly specifiable in feature structures alone. The use of feature structures allows each of a phrase structure node's grammatical features to be specified independently of the others. This allows a wide variety of phenomena to be handled using the interaction of grammatical constraints that are specified independently in lexical entries. The mechanisms for specifying long distance dependencies make it possible to also handle these phenomena with information specified independently in lexical entries (although see the discussion in section 6.2). However, the mechanisms for specifying constituent structure inherit

most of the problems of Context Free Grammars (CFGs) from the preceding work on Generalized Phrase Structure Grammar (Gazdar *et al.*, 1985). These formalisms each use structural configuration to specify when a constituent needs more information specified about it (expectations), and when a constituent can't have more information (directly) specified about it (iteration restrictions). Thus expectations and iteration restrictions cannot be specified independently of the specification of constituent structure. Since the S&A architecture's computational constraints place requirements on how constituent structure is specified, we want a grammar formalism which allows these requirements to be satisfied without interfering with the specification of expectations and iteration restrictions, and therefore we want to be able to specify these different kinds of information independently. Also, although some mechanisms, such as the Head Feature Principle, have been added to extend HPSG's domain of locality beyond that of CFGs, there are still constraints on what groups of structural constraints can be specified in a single grammar entry. Thus different constituent structure specifications are not completely independently of each other.[2]

Description Theory (D-Theory) also embodies the idea that partial descriptions are central to grammatical specification, but it takes a more computational perspective than HPSG. In particular, D-Theory is concerned with parsing deterministically. This leads Marcus, Hindle, and Fleck (1983) to address the problems that were just discussed for CFG-based formalisms. Rather than using a representation of constituent structure that directly parallels that structure, they use unstructured sets of statements about the nodes in the constituent structure. This representation allows the specification of grammatical information about different nodes to be independent of the structural relationship between the nodes. To allow such a representation, they use variables to refer to nodes, rather than feature structures to represent nodes. It is possible to delay the statement of equality between two variables, thus allowing the resolution of many ambiguities to be delayed. This technique will be very important in the work presented here. (Hence the name "Neural-network *Node Equating* Parser".) Another change they make is to specify structural constraints with dominance relationships, rather than immediate dominance relationships. Dominance is the recursive, transitive closure of immediate dominance, so every node dominates itself, and any node that x dominates is dominated by all the nodes that dominate x. This allows the direct specification of long distance dependencies, and it allows the resolution of many attachment ambiguities to be delayed by attaching high and lowering later. While dominance relationships will be used below to specify long distance dependencies, attachment ambiguities will be handled by delaying the equation of variables. This points to a property of D-Theory that will not be adopted here. They take the concept of partial description to the point where there is no concept of a description ever being complete. The parser simply finds out as much information as it can about the structure, and leaves the rest up to later stages of language processing. While this may be an appropriate model for parsers that need to handle the ungrammaticalities and incompleteness of real natural language utterances, it makes it impossible for a formal characterization of the parser's grammars to express expectations. Without having a notion of the parse not being finished, there can be no notion of something being needed for the parse to be acceptable. For example, there can be no specification of obligatory arguments. Thus the formal characterization of the parser's grammars requires that there be a definition of when the description of the sentence's phrase structure tree can be called complete. For the parser, the only significance of this notion of completeness is the expectations

[2]These problems should not be taken as criticisms of HPSG, since the objectives of that investigation are very different from the objectives of the one discussed here. HPSG is a linguistic theory which is predominantly concerned with competence phenomena. This investigation is far from a linguistic theory, and it is predominantly concerned with computational issues in language. There are certainly aspects of the formalism used here which would need to be changed to make it appropriate for an enterprise such as HPSG's. However, given the large amount of overlap between the two frameworks, future work should be able to synthesize insights from both of them.

that it has. If we allow these expectations and other grammatical constraints to be of varying degrees, then we are again in a position to handle "unexpected" input, such as ungrammaticality and incompleteness.

3.1.3 Computational Locality

Given a grammatical formalism that allows constituent structure information to be specified independently of expectations and iteration restrictions, it is possible to associate grammatical information with phrase structure nodes in a way that allows as much computation as possible to be local to individual nodes.[3] Because the information about a single node is unstructured, the desire to localize computation needs to be balanced against the need to express how the different words in the sentence relate to each other (i.e. the sentence's compositional structure). Thus each node should represent a portion of phrase structure which other grammatical investigations describe with a consistent structure. In this section, Government Binding theory will be used to to determine the largest such domain. The other issue discussed above that involves computational locality is the need to be able to remove nodes from the parser's bounded memory. This requires a particular representation of the relationships between nodes. To determine this representation, Combinatory Categorial Grammar (Steedman, 1987) will be discussed.

In Government Binding theory (GB), X-bar theory provides a consistent structure for all projections. Thus given predicates for each of the distinguished roles in this structure, each projection can be represented in an unstructured fashion. The X-bar structure can be unambiguously recovered from the unstructured representation, given the roles and the universal principles of X-bar theory. Similarly, the relationship between a given lexical projection (headed by a noun, verb, adjective, adverb, or preposition) and its associated functional projections (headed by a determiner, complementizer, or inflection) is universal, so these relationships can also be unambiguously determined from an unstructured representation. Beyond this domain there is much more variability in the possible structures. For example, a noun's projections may be immediately dominated by either a verb's projections or a preposition's projection, and it may be in the specifier role or the complement role. Therefore, the largest domain of phrase structure information that can be associated with a single node in the parser's grammatical representations includes a lexical projection and all its associated functional projections. This is the representation that is used in this investigation. It results in a much flatter phrase structure tree than is used in most investigations. While flatter representations than that used in GB are common, it is very rare to go so far as to not even represent the distinction between the sentence and the verb phrase. While this distinction may be important at other levels of representation, it does not need to be made at the level of syntactic constituent structure, for the reasons just discussed. These issues are discussed in more detail in section 6.1.

As discussed above, localizing most computation to individual nodes allows those nodes that will not be directly involved in the rest of the parse to be forgotten, provided the right set of relations is used. To determine what relations should be used, we can look at a grammar formalism which is

[3] Because it is only the information that needs to be dynamically manipulated that is important for the requirements imposed by the S&A architecture, this computational domain of locality is different from the domain of locality for grammatical specification. Grammar entries need to specify all the grammatical dependencies that exist in the language, but, as will be argued in this section, violations of these dependencies can be checked much more locally using multiple independent computations. For example, the grammar entry for "of" needs to specify both that it modifies a noun phrase and that it takes a noun phrase object, but checking each of these constraints does not require the information about the other.

already capable of representing parses of arbitrarily long sentences using structure specifications of bounded size, namely Combinatory Categorial Grammar (Steedman, 1987). Like other formalisms based on Categorial Grammar, Combinatory Categorial Grammar (CCG) represents grammatical information in categories which can be viewed as abstract types for tree fragments.[4] For example, the category NP is the type for tree fragments which have an NP node as their root and have no missing constituents. The categories of the form X/Y are types for tree fragments which would be tree fragments of type X if they were to combine with a tree fragment of type Y. Because these categories only represent the root node and the missing constituents, they abstract away from the completed constituents which tree fragment contains. CCG even provides a mechanism for abstracting away from nodes before their constituent is completed, as long as the node itself is complete. This mechanism is the composition operation, which combines a category of the form X/Y with a category of the form Y/Z to produce the category X/Z. For example, the category for "Barbie said" (S/S) can be composed with the category for "Ken likes" (S/NP) to produce a single category for "Barbie said Ken likes" (S/NP). Because the internal structure of this phrase is not represented, this single category is no larger than the category for "Barbie said". In this way, CCG can represent the parse of an arbitrarily long sentence using only categories of bounded size. Section 3.4 discusses these issues further.

Since CCG has this property, its categories must show how the relationships between nodes in a phrase structure tree can be represented while still allowing completed nodes to be removed from the representation of that tree. Since the innermost result of a category is the root of any tree fragment it describes, and the arguments are missing constituents within such tree fragments, these categories represent dominance relationships between nodes. They do not represent immediate dominance, since there may be completed nodes between the result node and the argument nodes. However, they do represent whether a node has an immediate parent. The innermost result of a category is the tree fragment root, so it does not have an immediate parent. The arguments of a category (or their innermost results) do not need to be taken as arguments to another category, so they have an immediate parent. As discussed in (Henderson, 1992), arguments of arguments do not have immediate parents, and higher order functions do not need to be considered for our purposes. The property of having an immediate parent is a property of an individual node, not a relationship between nodes. Relationships in the semantics of the category do not have to be represented at the level of processing being investigated here. The only other relationships represented in CCG categories are ordering constraints. Thus the only information that actually needs to be represented as explicit relationships is dominance and ordering constraints, and these relationships do not prevent removing completed nodes from the parser's representation of the sentence's phrase structure tree. As will be discussed in section 3.4, the grammatical representation also needs the ability to leave nodes that appear to be completed in the representation, which makes CCG itself inappropriate for this investigation.

3.2 Structure Unification Grammar

Structure Unification Grammar is a formalization of accumulating information about the phrase structure of a sentence until this structure is completely described. This section will expand the

[4] In some Categorial Grammar based formalisms, the categories can't strictly speaking be viewed as types for tree fragments. However, I take Lambek Calculus (Lambek, 1961) to define the core meaning of categories, and under this definition they can be viewed in this way. See (Henderson, 1992) for a formal explication of this relationship between Lambek Calculus categories and phrase structure tree fragments.

description given in the introduction by giving the details of SUG's definition.[5]

The first subsection below discusses the language which SUG uses to describe phrase structure trees. These trees are ordered trees of feature structures. The tree relations are immediate dominance, linear precedence, and dominance. Immediate dominance is the relationship between a node and each of its immediate children. Linear precedence is the ordering relation used here. Dominance is the recursive transitive closure of immediate dominance. Its addition is necessary in order to express long distance dependencies in a single grammar entry. The nodes of the trees are feature structures. They are divided into nonterminals, which are arbitrary feature structures, and terminals, which are atomic instances of strings. These feature structures are allowed to share values, including having the value of a feature be another node. Examples of how this descriptive language is used to express grammatical information are given below and in section 3.3.

The second subsection below specifies what constitutes an SUG derivation. The objects used in these derivations are partial descriptions in SUG's language for specifying phrase structure trees. Each step in a derivation combines one or more descriptions by conjoining them and equating zero or more pairs of nonterminal nodes. These equations are not allowed to make the resulting description unsatisfiable. A derivation must start with entries from the grammar, and must finish with a complete description. A complete description implies all and only what is true about a unique phrase structure tree. This tree is the result of the derivation. The tree set generated by a grammar is the set of trees resulting from some derivation for the grammar. The language generated by a grammar is the yields of these trees. Examples and more complete definitions will be given below.

To make the definition of SUG precise, the third section below gives a concise formal specification of SUG. The reader may want to skip that section.

3.2.1 Describing Phrase Structure

The central concept in Structure Unification Grammar is the partial description of phrase structure trees. It allows for great flexibility in both the specification of grammatical information and the processing of that information. This section presents the language which SUG uses to describe phrase structure trees.

The Notation

In recent years many linguistic formalisms have been developed which use partial descriptions of linguistic information. These formalisms usually use feature structures to represent this information. The problem with feature structures is that the relationships which they can represent are restricted to being functional, in the sense that a feature structure label must represent a function from feature structures to feature structures. This causes trouble when specifying information about phrase structure, since many of the relations which we wish to state, such as linear precedence and dominance, are not functions. Formalisms like PATR-II (Shieber, 1986) and FTAG (Vijay-Shanker, 1987) solve this problem by using separate mechanism for specifying phrase structure. PATR-II uses a context free skeleton for this purpose, and FTAG uses Tree Adjoining Grammar (Joshi, 1987) skeleton. Description Theory (Marcus *et al.*, 1983) takes a different approach. It

[5]The description of SUG given in this section has been taken from chapter 2 of (Henderson, 1990), with only minimal modification. It is included here for completeness.

extends feature structures to allow structural relations to be expressed in the same manner as the information usually expressed in feature structures.[6] This later approach gives the description of the structural relations the same degree of partiality given the other information. For this reason this is the approach which will be taken here.

There have been several suggestions for how to add arbitrary relations to feature structures. One was proposed in (Rounds, 1988), where set values are added to feature structures. This would allow linear precedence, for example, to be expressed by giving a node a feature with a set value containing all the nodes which precede it. However, this approach would force an unwanted asymmetry in the representation between preceding and being preceded by. Instead I will not use the automata based conception of feature structures used by Rounds, but use a representation espoused by Johnson (1990). In this representation feature structures are specified using quantifier-free first-order formulae with equality. In these formulae, variables range over feature structures, atoms are represented as constants, and labels are specified as unary functions from feature structures to feature structures. In (Johnson, 1990), the characteristics of atoms and a treatment of incomplete information are axiomatized. This axiomatization will be discussed below. The advantage of this system over Rounds' representation of feature structures is that quantifier-free formulae already have a mechanism for specifying arbitrary relations, namely predicates. For example, if node x precedes node y this can be expressed as *precedes(x,y)*[7].

The shift to using quantifier-free formulae as the notation for feature structures suggests a few changes which I will adopt. Since a typical formula will contain many variables, none of them distinguished from the others, I will treat a formula as describing a set of entities, rather than a single one. This has the consequence that our phrase structure descriptions no longer need to be root centered. Given that we are talking about sets of entities, it is also natural to remove the restriction that they all be connected. These choices increase the formalism's flexibility, and thus make it more appropriate for investigating parsing strategies.

First-order formulae not only provide us with a natural representation for our descriptions, they also provide a way to axiomatize the characteristics of the relations we wish to add. Stating relations between nodes will have no causal role in a parse if we do not restrict these relations in accordance with their intended meaning. These axioms can simply be added to the set already introduced by Johnson to define the nature of atoms and undefined information. In order to do this the notation will have to be expanded to first-order formulae with quantifiers. The only problem with this is that the satisfiability problem for first-order formulae with quantifiers is undecidable. However, Vijay-Shanker (1987) showed that the unrestricted use of feature structures combined with the ability to generate arbitrarily large structures results in an undecidable system, so we already know that SUG is in general undecidable. Quantifiers will still be excluded from grammar entries. Constraints imposed by the parser ensure that the parser's grammars are decidable.

[6] Rounds and Manaster-Ramer take a similar approach in (Rounds and Manaster-Ramer, 1987). This will be discussed in section 3.3.3.

[7] The problem with Johnson's representation of feature structures is that he uses the usual classical semantics for first-order formulae. This means that, unlike in Rounds' system, in his system subsumption does not respect entailment, where subsumption is as defined in (Rounds and Kasper, 1986). In other words, given two feature structure models, A and B, such that the nonnegative information in A is a subset of that in B (A subsumes B), there may be descriptions which are satisfied by A but not by B. This is because a description may have a negative constraint which is incompatible with information which is in B but not in A. This will not be a problem here because the use of negation is limited to axioms in the definition of SUG which either are true in all phrase structure tree models, or are simply predicating something's existence. Thus this problem can not arise, and in SUG subsumption does respect entailment, with the models restricted to those specified in the next section.

The Structure Models

Before discussing how to describe phrase structure trees, it is necessary to specify the objects to be described. I will restrict the set of models for the descriptions to ordered trees of feature structures. The nodes of these trees are divided into two types, terminals and nonterminals. The nonterminals are models of arbitrary feature structures.[8] Terminals are all instances of strings. The terminals must be instances of strings rather than strings because otherwise the phrase structure of a sentence with the same word occurring twice would not be a tree. Values in the feature structures can corefer, both within a single node and between the feature structures for different nodes. This includes the ability to have a node be the value of a feature in another node.

The only components of the allowable structures other than the above feature structures are the two ordered tree relations, immediate dominance and linear precedence. Immediate dominance is the relationship between a node and each of its immediate children. The graph of the immediate dominance relation must be a single tree. Linear precedence is the ordering relation. It is a partial order on nodes which is transitive and antisymmetric. Also, if a node x linearly precedes a node y, then everything in the subtree below x linearly precedes everything in the subtree below y.[9]

The Descriptions

As discussed above, the language SUG uses to describe models of phrase structure trees uses first-order logic as its notation. In this representation variables range over feature structures and the constant $\perp$, constants represent atomic feature structures, unary function symbols and equality are used to represent feature-value relationships, and binary predicates are used to represent tree relations. Johnson (1990) shows how to represent the feature structures in this way. If a feature structure x has y as its f feature's value, this is represented as the statement $f(x) \approx y$. The constant $\perp$ is used to represent nonexistent values of functions, since first-order logic requires functions to be total.[10] The use of functions to specify feature values enforces the fact that a given feature structure can have only one value for each of its features. The characteristics of constants are enforced with the following axioms, taken from (Johnson, 1990).[11]

1. For all constants c and feature labels f, $f(c) = \perp$
2. For all distinct pairs of constants c_1 and c_2, $\neg(c_1 = c_2)$

The characteristics of $\perp$, which represents nonexistent information, are axiomatized as follows, also taken from (Johnson, 1990).

[8] Any models of simple feature structures will do here, as long as they must be single feature structures and must be connected. One such set of models is given in (Rounds and Kasper, 1986).

[9] There are a couple other constraints which could be imposed on the allowable models, which I have not chosen to include. One is that the root of the tree have category S, but this seems better incorporated at the level of a linguistic theory. Another is that the linear precedence relations completely order the terminals, since the words of a sentence are always completely ordered in either time or space. I have not included this constraint because there seem to be sentences in some languages for which some of this ordering is not significant to the sentence's phrase structure.

[10] Johnson says this symbol is for undefined information, but I will use the term "nonexistent" because it is less easily confused with the term "unspecified". A feature structure can be completely unspecified and yet still exist.

[11] Unlike in Johnson's system, the fact that terminal nodes are instance unique atomic feature structures means that there are an infinite number of constants. That means that there are an infinite number of axioms of the form 1 and 2. Thus these axioms should really be universally quantified for c, c_1, and c_2.

3. For all feature labels f, $f(\bot)=\bot$

4. For all constants c, $\neg(c=\bot)$

Finally, when the value of a feature is specified then it must exist. This means that the specification can not be done simply using equation, since $f(x)=y$ is consistent with $y=\bot$. Thus Johnson defines another operator "$\approx$" to be used for specifying features, which is defined as follows.

For all terms u,v, $u \approx v \Leftrightarrow (u=v \wedge \neg(u=\bot))$

This is not an axiom, since there are an infinite number of instantiations of it, but a definition of what $\approx$ is an abbreviation for.

The axiomatization of the tree relations are done similarly to the above axioms, only tree relations are specified using predicates rather than functions. The predicates *idom* and *prec* specify immediate dominance and linear precedence relations between nodes, respectively. Formulae may also specify dominance relations between nodes using the predicate *dom*. Dominance is the recursive, transitive closure of immediate dominance. Thus a node x dominates a node y either if x equals y or if there are a series of nodes $z_1, \ldots, z_n$ such that $x=z_1$, $y=z_n$, and for all i, $1 \leq i \leq n-1$, z_i immediately dominates z_{i+1}. Nodes are distinguished from other feature structures using the predicate *node*, and terminals are distinguished from nonterminals using the predicate *terminal*. These predicates are axiomatized as follows, where *strings* is the set of instances of strings.

$\forall\ x,y,z,w,$

5. $\neg(idom(x,x))$
6. $idom(x,y) \wedge \text{idom}(z,y) \Rightarrow x \approx z$
7. $idom(x,y) \Rightarrow dom(x,y)$
8. $\neg(prec(x,x))$
9. $prec(x,y) \wedge prec(y,z) \Rightarrow prec(x,z)$
10. $prec(x,y) \wedge dom(x,w) \wedge dom(y,z) \Rightarrow prec(w,z)$
11. $dom(x,y) \wedge dom(y,z) \Rightarrow dom(x,z)$
12. $dom(x,y) \wedge dom(y,x) \Rightarrow x \approx y$
13. $dom(x,y) \Rightarrow (x \approx y \vee \exists z(idom(z,y) \wedge dom(x,z)))$
14. (a) $prec(x,y) \Rightarrow (node(x) \wedge node(y))$
    (b) $dom(x,y) \Rightarrow (node(x) \wedge node(y))$
    (c) $terminal(x) \Rightarrow node(x)$
15. $node(x) \Rightarrow dom(x,x)$
16. $terminal(x) \Leftrightarrow (\exists s \in strings,\ x \approx s)$
17. $terminal(x) \wedge dom(x,y) \Rightarrow x \approx y$

Figure 3.1 gives an example of how phrase structure is specified in this descriptive language. Not all the information about the structure is explicitly specified in the formula, but the rest is derivable given the above axioms.

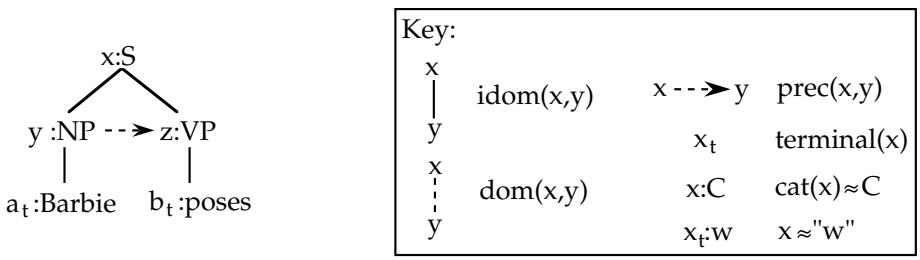


Figure 3.1: The structure specified by $(cat(x) \approx S \wedge cat(y) \approx NP \wedge cat(z) \approx VP \wedge a \approx$ *"Barbie"* $\wedge\, b \approx$ *"poses"* $\wedge\, idom(x,y) \wedge idom(x,z) \wedge idom(y,a) \wedge idom(z,b) \wedge prec(y,z))$.

The above axioms complete the definition of the language SUG uses to describe phrase structure trees. The grammar specifications can only use a subset of the expressive power of this language, but the full power is necessary in order to express and reason with the axioms. In particular, all the variables in a grammar entry must be existentially quantified and the only logical connective which can be used is conjunction; universal quantification, disjunction, and negation cannot be used. This will be discussed more in the following section.

3.2.2 Accumulating Phrase Structure

With the above language for describing phrase structure trees, it is now possible to define the process of deriving phrase structure trees in Structure Unification Grammar. An SUG derivation starts with partial descriptions from the grammar, and sticks them together using node equations, until a complete description of some phrase structure tree is constructed. That tree is the result of the derivation. This process can be visualized as taking a set of transparencies, each with a grammar entry on it, and placing them on top of each other[12] until the resulting picture is of a complete phrase structure tree.[13] As this depiction implies, the descriptions in the grammar are not arbitrary formulae in the language for describing phrase structure trees. This would allow negative facts, disjunctive facts, and universal facts to be expressed in the grammar entries, all of which cannot be depicted in this simple way. Grammar entries are restricted to being conjunctions of facts with only existentially quantified variables. The restrictions on what is a complete description of a phrase structure tree are defined by the need to have a unique phrase structure tree as the result of the derivation.[14] As with any partial description, the description resulting from a derivation has an infinite number of phrase structure trees which are compatible with it. One way to find a

[12] Of course, this will only work if the original transparencies have their information spatially laid out in a way compatible with the total resulting picture.

[13] This characterization is slightly misleading, since there will be information about the resulting description, as a consequence of the axioms, which is not depicted in any of the original transparencies. New dominance and precedence relationships between nodes are an obvious example of this, although there are other less obvious possibilities. Nonetheless, all the information about the resulting description can be recovered from the resulting depiction using the axioms. In any case, this characterization is a useful way to think about SUG derivations.

[14] The requirement that the result be a tree and not a description is a requirement at the formalism level. It is necessary in order for derivations to have some criteria for completion. Otherwise there would be no reason to do any equations during a derivation. Whether the result produced by the syntactic parser should be viewed as a tree or as a description is not clear. For the purposes of this investigation it will be assumed that the result is a tree and if a tree is not produced the parser has failed. The ability of people to understand incomplete sentences, sentence fragments, and sentences with unresolved ambiguities seems to indicate that an investigation with a less idealized view of language might want to treat the result of the parser as a description. See (Hindle, 1983) for an example of such an approach.

unique tree for a description is to take the circumscriptive closure. In other words, assume that anything which is not entailed by this description is false. This definition can only find such a tree for a subset of the descriptions, called complete descriptions. All descriptions in this subset must specify a single immediate dominance tree which includes all the nodes, and must specify the string associated with every terminal. This section will go into the above discussion in more detail.

Grammar Entries

An SUG grammar simply consists of a set of partial descriptions of phrase structure trees. These descriptions specify what configurations of information are allowed by the grammar. If a particular description is in the grammar, then that description's information can be added to a description in a derivation, as long as all its information is added and the result is satisfiable and complete. For example, the grammar entry $(cat(x) \approx S \wedge cat(y) \approx NP \wedge cat(z) \approx VP \wedge idom(x,y) \wedge idom(x,z) \wedge prec(y,z))$ allows two nodes whose *cat* features are compatible with NP and VP, respectively, to attach under a node with a *cat* feature compatible with S, but in the resulting description the NP node must precede the VP node. This example could equally well be described with the precedence information being the precondition and the category information being the result, but regardless the requirement is the same; all the information can be included as long as all the information is included. Other examples will be given throughout the rest of this document. This meaning of grammar entries may be clearer in the case of a lexicalized grammar. In this case the presence of a word in a sentence can "license" the portion of the complete structure which is specified in one of the word's grammar entries, as long as the rest of the structure is compatible with this portion.

The entries in an SUG grammar are not arbitrary partial descriptions of phrase structure trees. They are restricted to a subset of SUG's language for describing phrase structure trees. First, SUG grammar entries must be satisfiable, since using an unsatisfiable grammar entry in a derivation will always result in an unsatisfiable resulting description. More interestingly, SUG grammar entries can only have existentially quantified variables and the only logical connective allowed is conjunction. They cannot use universally quantified variables, disjunction, or negation. Because all variables in a grammar entry are existentially quantified, the quantifiers are not explicitly specified. These restrictions are imposed for several reasons. First, they ensure that in SUG subsumption respects entailment. Second, they greatly simplify determining if a description is complete. If negation or disjunction were allowed in the grammar entries, then a grammar entry could specify grammar specific characteristics which need to be uniquely determined for the description to be complete.[15] Third, it restricts the domain of locality of grammar entries. If universal quantification was allowed in grammar entries then they could directly constrain nodes which are not mentioned in their description. Fourth, the intuitive characterization of SUG as simply constructing a picture of the derived phrase structure tree by overlaying pictures of the grammar entries, would not be possible without these restrictions on the language used to specify SUG grammar entries. In addition to these formalism level motivations, these restrictions are necessary in order to store SUG descriptions in the Shastri and Ajjanagadde connectionist computational architecture, since these constraints on SUG grammar entries are also restrictions on the S&A architecture's memory mechanism.

Grammar entries are the leaves of SUG derivation trees. However, if the same grammar entry is used twice in the same derivation, then the two instantiations of the grammar entry cannot be

[15] Other than this complication there are no problems with allowing disjunction in grammar entries. Not permitting disjunction does not restrict the languages generable by SUG, since any disjunction can be specified with a grammar entry for each possible choice in the disjunction.

identical. First, the two instances must use disjoint sets of variables. This is simply a technique for avoiding variable capture during the derivation due to changing the scope of the implicit existential quantifiers. Second, the two instances must have distinct terminals. When the same word occurs twice in a sentence it must be manifested as two distinct terminals in the phrase structure, otherwise the phrase structure is not a tree. Thus whenever a grammar entry is introduced into a derivation, all its terminals are replaced with new unique instances of their words. This has the effect of preventing any two terminals with their words specified from ever equating.

Combining Structure Descriptions

The combination operation in SUG derivations is very simple. A set of descriptions are combined by conjoining them and adding zero or more statements of equality between their nonterminal nodes. Simply taking the conjunction of the descriptions would not be sufficient, since the fragments would never become connected, and thus would never form a complete description of a tree. Permitting arbitrary information to be added would not permit the grammar to constrain the set of derivable phrase structure trees. By only allowing coreference information to be added SUG avoids both these problems, and it conforms to the intuitive characterization of SUG as simply constructing a picture of the derived phrase structure tree by overlaying pictures of the grammar entries. An example of this combination operation is given in figure 3.2. The only restriction on what equations can be added is that the resulting description be satisfiable. This is exactly analogous to unification in normal feature structures, which is also specified in this notation as equation under the condition that the result be satisfiable. Thus the equation of two nodes results in their feature structures being unified in the resulting description. It is worth noting that the set of equations used in combining two descriptions is not determined uniquely. The definition of a combination is nondeterministic. Descriptions S and T can combine to produce a satisfiable description U if there exists a conjunction of equations of nonterminals, E, such that $U = S \wedge T \wedge E$. Also note that the fact that only the equation of nonterminals can be added does not prevent terminals from equating, since the unification of features in nonterminals can cause the equation of terminals as a side effect.

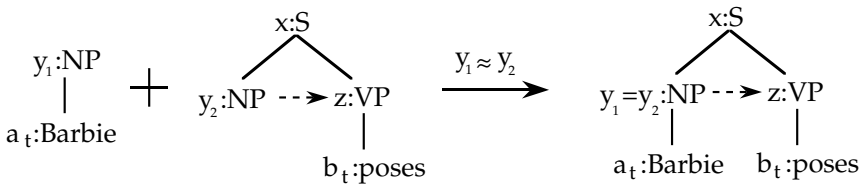


Figure 3.2: The combination of $(cat(y_1) \approx NP \wedge a \approx$ "*Barbie*" $\wedge\ idom(y_1, a)) = F_1$ with $(cat(x) \approx S \wedge cat(y_2) \approx NP \wedge cat(z) \approx VP \wedge b \approx$ "*poses*" $\wedge\ idom(x,y_2) \wedge idom(x,z) \wedge idom(z,b) \wedge prec(y_2,z)) = F_2$ using the equation $y_1 \approx y_2$ to form $(F_1 \wedge F_2 \wedge y_1 \approx y_2)$.

Complete Structure Descriptions

When a derivation is done there needs to be a phrase structure tree which it derives. However, the results of the above combinations are descriptions, not trees. The question is, what phrase structure tree does the description resulting from a derivation specify? Given a partial description, the usual way to to make it a complete description is to invoke the closed world assumption, a process also called taking the circumscriptive closure. In other words, the description is assumed to

specify everything which is true about the thing being described. Under this assumption, anything which is not entailed by the description is false. However, this will not produce a satisfiable set of constraints if the original description contains disjunctive information. If the description entails $f \vee g$ but does not entail f and does not entail g, then this assumption will produce a description which entails $(f \vee g) \wedge \neg f \wedge \neg g$, which is unsatisfiable.

In SUG descriptions the above problem arises in two ways. First, if a node x is dominated by a distinct node and x does not have an immediate parent specified, then there is an ambiguity as to what the immediate parent of x is, as is manifested in axiom 13 in section 3.2.1. This ambiguity means that after taking the circumscriptive closure there will be no tree models which satisfy the description. In other words, for any description which has some nonroot node without its immediate parent specified, the closed world assumption will produce an unsatisfiable description. The other way this problem arises is when a terminal is specified to exist but no word is specified for it. When the circumscriptive closure is applied to such a description, the terminal will be assumed to be unequal to every instance of a string. Because in phrase structure tree models all terminals are instances of strings, no models will satisfy the resulting description. These facts imply the only way circumscriptive closure will produce a satisfiable description is if all the terminals which are known to exist have their word specified and all nodes except the root have an immediate parent specified. Thus in order to use this method for determining the resulting phrase structure tree, the resulting description must have all the terminals' words specified and must specify a single immediate dominance tree which includes all the nodes. In SUG such a description is called a complete description, because it completely specifies a unique phrase structure tree under the assumption that anything which is not entailed by the description is false.

The above approach to finding a unique phrase structure tree for a given description only works for complete descriptions. Since we do not want to make arbitrary choices when determining the resulting tree of a derivation, the only derivations which can be allowed are those which result in such a complete description. This is precisely the requirement for finished SUG derivations; the resulting description must be complete.

The fact that there are conditions on the final resulting description of a derivation allows grammar entries to express what needs to be true in the final description by violating the completion conditions locally. A simple example is that the root(s) of any grammar entry do not have immediate parents, and thus must either be the sentence root or find some place in the rest of the sentence's structure to attach. Not specifying the immediate parent of a node can also be used with nodes which are dominated by some other node in a grammar entry. Such nonredundant dominance constraints express the need for a chain of immediate dominance constraints in their place. For long distance dependencies this can be used to express the need for a gap in the subsequent sentence, as is illustrated in the structure for 'who' in figure 3.6 on page 71. It can also be used to attach a subject to its S node while still expressing the need for a verb which subcategorizes for the subject. The need to have a string specified for all terminals provides a more flexible mechanism for expressing the need for information. In figure 3.3 below, and throughout this document, nodes have a feature *head* which specifies a terminal. If this terminal's string is unspecified then it needs to equate with another terminal which has a string specified. Since only nonterminals can be equated in derivations, the only way for this underspecified terminal to equate is if the node whose head it is equates with another node which has a word specified for its head. When the feature structures of the two nonterminals are unified, the *head* features will cause the terminals to be equated, and thus the underspecified terminal will receive a string. By using underspecified head terminals the grammar can express the requirement that a given node must get a head. If such an unheaded node is a leaf in the structure, then this expresses the obligatory subcategorization for a phrase of

the node's type. If such an unheaded node is the root of the grammar entry, then this specifies that the structure must attach as an adjunct to a phrase of the root's type. In this way unheaded roots are used to express modification relationships. Underspecified terminals can be used with other features, such as *determiner*, to express similar constraints. This discussion will be expanded in section 3.3.

The Derivations

As discussed above, an SUG derivation starts with descriptions taken from the grammar, combines them by conjoining them and adding equations between nodes, and ends with a complete description which specifies the resulting tree of the parse. Each of these components of a derivation are discussed at length in the previous sections. Such a derivation can be described as a tree, the leaves of which are the initial descriptions, the internal nodes of which are the intermediate descriptions, and the root of which is the resulting description. The leaves of an SUG derivation tree are entries from the grammar, except their variables have been replaced with fresh variables and their instances of strings have been replaced with fresh instances of strings. This replacement is done in such a way that all the leaves of a derivation tree have disjoint sets of variables and disjoint sets of instances of strings. This process is done to prevent two instantiations of the same grammar entry from getting their variables or terminals unintentionally conflated. Each internal node of an SUG derivation tree is the result of a combination of its children. Thus an internal description is the conjunction of its children, plus a conjunction of zero or more equations between nonterminal nodes in its children. The sets of equations are limited to those which result in satisfiable descriptions. There are no other restrictions on these equations. Because the grammar entries must be satisfiable and the result of each combination must be satisfiable, all the descriptions in an SUG derivation tree will be satisfiable, including the resulting description. The root of an SUG derivation tree is the resulting description and thus must be a complete description. This means this description must specify a single immediate dominance tree which includes all its nodes, and must specify an instance of a string for all its terminal nodes. This requirement guarantees that the resulting description will specify a unique phrase structure tree after taking the circumscriptive closure. This unique tree is the resulting tree of the derivation. The sentences whose words and ordering are compatible with the terminals of the resulting tree are the resulting sentences of the derivation. Note that there may be more than one such sentence, since the ordering of the terminals may be underspecified.

An example derivation is shown in figure 3.3. The leaves of the derivation tree are toy grammar entries for 'Barbie', 'dresses', and 'fashionably', and are given at the top of the figure. The first step of the derivation combines the first two structure descriptions with the equation $y_1 \approx y_2$. The second step combines the resulting structure description with that for 'fashionably' with the equation $z_1 \approx z_2$, thus forming the complete description shown at the bottom of the derivation. This final description is then interpreted as implying everything which is true about the resulting tree, thus specifying a unique tree, namely the tree depicted at the bottom of the figure. The only sentence compatible with the ordering constraints on this resulting tree is "Barbie dresses fashionably". Note that many other derivation structures are possible, including the one step derivation which combines all three structures with both equations at the same time. In fact, all derivations will have an equivalent derivation for each possible way of combining the grammar entries.

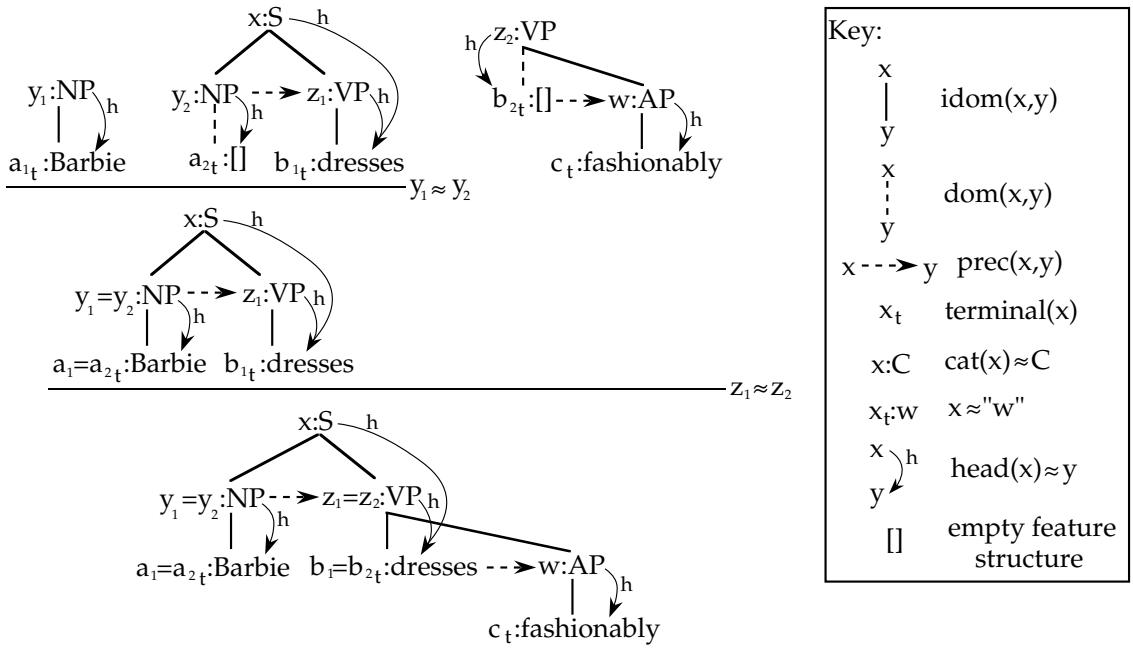


Figure 3.3: A derivation of the sentence "Barbie dresses fashionably". The descriptions shown in the top row are grammar entries and the tree depicted at the bottom is the result of the derivation.

3.2.3 A Formal Specification of SUG

To clarify the above discussion, the following is a formal specification of an SUG grammar and the sentences it generates. An SUG grammar is a tuple $\langle S,\ L,\ A,\ V\rangle$, where V is the variables, $A \cup strings$ is the constants, L is the function symbols, $\{idom,\ prec,\ dom,\ terminal,\ node\}$ is the predicates, and S is a finite set of first order formulae in these primitives. The formulae in S do not use disjunction or negation, and all their variables are implicitly existentially quantified. The arity of all functions is one. *Strings* is a set of instances of strings. The arities of *terminal,* and *node* are one. The arities of *idom, prec,* and *dom* are two. What satisfies a formulae and what a formulae entails are always determined with respect to the axioms given in section 3.2.1.

A description F is generated by a grammar $\langle S,\ L,\ A,\ V\rangle$ if and only if F is satisfiable, F is complete, and $F = F_1 \wedge \ldots \wedge F_n \wedge E$, where the variables and instances of strings in F_1 through F_n are disjoint, there exits a substitution θ for variables and instances of strings such that $F_1[\theta],\ldots,F_n[\theta] \in S$, and E is a conjunction of equations between nonterminals in F. A formula F is complete if for every terminal x in F, F entails $x \approx s$ where s is an instance of a string, and for every node x in F, F either entails $x \approx r$, or there is a node y such that F entails *idom(y,x)*, where r is a unique node in F. x is a terminal in F if F entails *terminal(x)*, x is a node in F if F entails *node(x)*, and x is a nonterminal in F if x is a node in F but not a terminal in F.

A tree is generated by a grammar if it is the subsumption minimal phrase structure tree for some description generated by the grammar. A tree T is the subsumption minimal phrase structure tree for a description F if T satisfies F, and, for all trees T' which satisfy F, T subsumes T'. Such a tree will always exist and be unique for any description generated by a grammar, since all such descriptions are complete descriptions. A tree T subsumes a tree T' if and only if all the descriptions which T' satisfies are also satisfied by T (Rounds and Kasper, 1986). This definition of the resulting tree is equivalent to the one using circumscriptive closure given above.

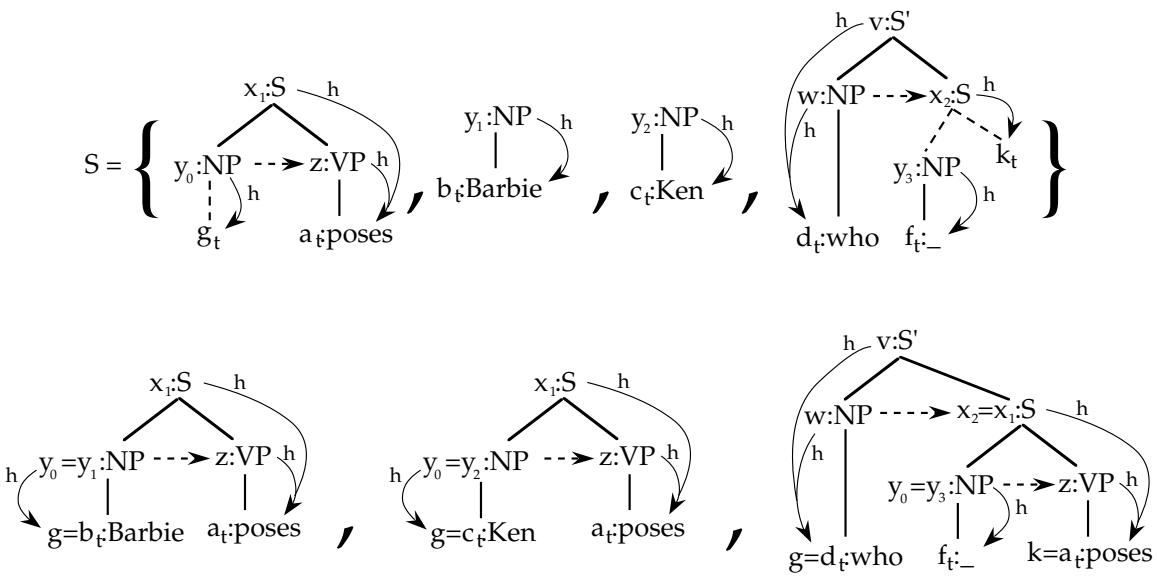


Figure 3.4: The second row of structure descriptions are those generated by the grammar $G=\langle S,\ L,\ A,\ V\rangle$ with S as shown in the first row, $L=\{cat,\ head\}$, $A=\{S',S,NP,VP\}$, and $x_1, x_2, y_0, y_1, y_2, y_3, z, w, v, a, b, c, d, f, g, k \in V$.

A list of strings s is generated by a grammar $G=\langle S,\ L,\ A,\ V\rangle$ if and only if s is a sentence for a tree generated by G. A list of strings s is a sentence for a tree T if there is a bijection g from words in s to nonempty terminals in T such that, $g(w)$ is an instance of w and, if $g(u)$ precedes $g(v)$ in T then u precedes v in s. Nonempty terminals are those which are not instances of the empty string. An example of a simple grammar and the formulae generated by it is shown in graphical form in figure 3.4.

3.3 Expressing Grammatical Information in SUG

Structure Unification Grammar is well suited for the investigation of parsing in the Shastri & Ajjanagadde architecture because it has the properties discussed in section 3.1. SUG has a large domain of locality for expressing grammatical information, which makes it expressive enough to allow the necessary grammatical information to be directly stated. SUG's use of instance unique terminals allows expectations and iteration restrictions to be specified independently of structural constraints. SUG's use of partial descriptions allows known information to be specified independently of that which isn't yet known. In this section, the significance of these characteristics for expressing grammatical information in SUG are demonstrated through a series of examples.[16] The first subsection gives several examples of how SUG's large domain of locality allows the perspicuous representation of lexically specific information within a word's grammar entry. The second subsection discusses how the partial specification of information can be used to express ambiguities.

[16]Throughout this section I will be giving particular analyses, but these analyses are not the point of this section. The objective is to demonstrate how to express in SUG analyses of the types discussed. Many other analyses are possible within SUG, with varying degrees of naturalness. In particular, these analyses give each phrase multiple bar levels, $\overline{\mathrm{X}}$, $\overline{\overline{\mathrm{X}}}$, and sometimes $\overline{\overline{\overline{\mathrm{X}}}}$. This makes the analyses more similar to some standard linguistic analyses. In later sections these different bar levels will be collapsed into a single node.

In the final subsection the importance of these characteristics is supported with a short discussion of some other grammar formalisms which can be viewed as accumulating partial descriptions of phrase structure trees, but which are missing some of these characteristics.[17]

Before discussing specific examples some general techniques for expressing grammatical information in SUG need to be discussed. Any grammatical formalism needs mechanisms for expressing in grammar entries both information which must not be contradicted and information which must be provided by other grammar entries in the derivation. The former is needed for stating things like "this word is a verb", and the later is needed for expressing things like "this word must have an object". All information stated in an SUG grammar entry is of the type which cannot be contradicted. By leaving certain information out, an SUG grammar entry can also express expected information. This is because there are restrictions on what descriptions can be the result of an SUG derivation. Recall that the final description of a derivation must be a complete description of some phrase structure tree, and such complete descriptions must specify a single immediate dominance tree which includes all mentioned nodes and specify words for all mentioned terminals. By not satisfying these restrictions locally, a grammar entry can require that some other grammar entry provide the necessary information to satisfy the restrictions. This was mentioned in the last section and, as promised, it is expanded on here.

The most straightforward mechanism for expressing expectations is to introduce a terminal which does not have its word specified. Because all terminals must have their words specified before the derivation is finished, such an underspecified terminal must equate with a terminal which has its word specified. But, derivations cannot equate terminals directly; they can only equate nonterminals. Terminals can, however, be equated as a side effect of the unification of the feature structures of two equated nonterminals. Thus an underspecified terminal must be referred to in the feature structure of some nonterminal, and the terminal which it equates with must be referred to in the same way in the feature structure of some other nonterminal. An example of this specification is the *head* feature used in the examples of the previous section and throughout the rest of this document (later renamed *constituent head*). A nonterminal node with a *head* feature which refers to an underspecified terminal expresses the nonterminal's expectation for a head. A nonterminal with a *head* feature referring to a word expresses that the nonterminal has a head which can be used to fulfill other nodes' expectations for a head. As a consequence of using this technique, any two nodes which can both fulfill the same expectation cannot equate, since this would require the unification of two distinct terminal strings, which is never possible.[18] Thus, in this example, a given node can only have one head. Nodes which expect heads are used to express two different kinds of grammatical requirements. If the unheaded node is a leaf, then this expresses the subcategorization for an obligatory argument of the node's type. If the unheaded node is a root, then this expresses the modification of a phrase of the node's type. The unheaded root can equate with a node without interfering with other unheaded roots also equating to the same node, thus allowing the iteration of modifiers. The optionality of such modifiers follows from the fact that the unheaded root does not

[17] The subsections of this section are taken with little modification from (Henderson, 1990). That document also illustrates how grammatical information can be expressed in SUG through a series of comparisons with other grammar formalisms. These comparisons take analyses and insights from other investigations into natural language grammar, and show how they can be perspicuously translated into SUG. The investigations discussed are Lexical Functional Grammar (Kaplan and Bresnan, 1982), Description Theory (Marcus *et al.*, 1983), Abney's licensing parser (Abney, 1986), Tree Adjoining Grammar(Joshi, 1987), Lexicalized Tree Adjoining Grammar (Schabes, 1990), and Combinatory Categorial Grammar (Steedman, 1987).

[18] When SUG is used for parsing, the property of having a word as the value of a feature can be specified probabilistically. This allows a node with a nonzero nonone probability of having this property to fulfill an expectation and yet also equate with a node which also has this property, contrary to the generalization just stated for the categorical case.

fulfill any expectations on the node with which it is equated. The other feature which is used in this way in this chapter is the *determiner* feature. In the next chapter this feature will be generalized to the *functional head* feature, and the *verb* feature will also be used in this same way.

The other way to express an expectation is to mention a node which does not have an immediate parent. Since the final description of a derivation must specify a single immediate dominance tree, all nodes must either be the root or have an immediate parent. Thus an unparented node expresses its expectation to either be the root or be equated with another node which has a parent. This means that the root or roots of any grammar entry express the requirement that their grammar entry be incorporated into the phrase structure of the sentence. Another circumstance when a node is unparented is when it is dominated but not immediately dominated. In this case there is a nonredundant dominance constraint between the lowest node which dominates the unparented node and the unparented node. Such nonredundant dominance constraints are shown in figures as dashed lines. This constraint must be made redundant by a chain of immediate dominance constraints in its place. Thus the unparented node must find a parented node to equate with in the phrase of its lowest dominating node. This constraint is primarily used for expressing long distance dependencies. The unparented node is the trace, the dominating node is the root of the phrase out of which the extraction occurs, and the node which the trace node equates with is the gap.[19] The category of the gap can be restricted with the category of the trace node. The fact that the gap's argument position becomes filled is expressed by giving the trace node a head. Assuming the analyses of modifiers which was given above, if the extracted item is an adjunct rather than an argument, then the trace node has the category of the modified node (rather than the modifier) and is not headed. In this case the trace equates with the node which the extracted item "would have" adjoined to. Nonredundant dominance constraints can also be used to require immediate dominance chains which are not expected to be arbitrarily long. For example, a subject can be attached to an embedded S before the embedded verb by stating that the S dominates but does not immediately dominate the subject. This places the subject within the structure but still expresses the requirement that the verb subcategorize for the subject. The same technique can be used to incrementally combine other prehead constituents with the partial structure of the sentence. Many examples of each of these mechanisms will be given in the subsequent subsections.

3.3.1 Using SUG's Large Domain of Locality

Structure Unification Grammar's large domain of locality for expressing grammatical information permits an interdependent set of grammatical information to be expressed in a single grammar entry.[20] To demonstrate this ability this section will give examples of lexicalized grammar entries. Each of these entries will include a terminal for the word item and the fragment of structure necessary to express the grammatical information associated with that word. This grammatical information is simply what we know about the phrase structure given the presence of the word.

The significance of SUG's domain of locality can be demonstrated by contrasting it with that of Context Free Grammars. In CFGs, even the enforcement of subcategorization constraints needs to be coordinated between multiple grammar entries. The structure for 'rolls' in the middle of

[19] As is well known, there are other constraints on long distance dependencies other than that expressed by the dominance constraint here. Ways of further limiting the domain of extraction for a trace will be discussed in section 6.1.

[20] SUG's domain of locality is very similar to that of Tree Adjoining Grammar. The development of SUG was heavily influenced by work on TAG, especially that concerning the importance of a formalism's domain of locality ((Joshi, 1987), (Vijay-Shanker, 1987), (Schabes, 1990)).

figure 3.5 shows how several such constraints can be expressed in a single SUG grammar entry. The whole projection of the verb is present, the subcategorization for an $\overline{\overline{N}}$ subject is expressed,[21] and the agreement information is expressed on this subject. To express the interdependence between the lexical item and these constraints in a CFG would require introducing several features in node labels whose sole purpose would be to enforce this interdependence across the boundaries of grammar entries. There will be several other examples in this chapter which demonstrate how such node features can be eliminated given SUG's large domain of locality.

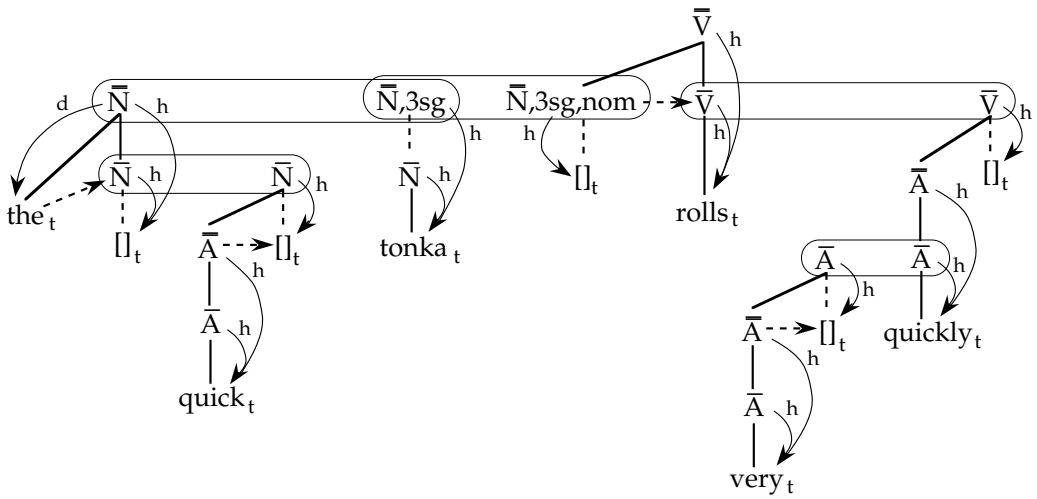


Figure 3.5: Some example grammar entries used to derive the sentence "the quick tonka rolls very quickly".

The structures for 'quick', 'quickly', and 'very' in figure 3.5 show how modification information can also be expressed within SUG's domain of locality. The roots of these structures all have underspecified terminal heads, and thus must be equated with some headed node. This way of expressing modification relationships permits the constraints on the modified node to be expressed in the same grammar entry as the modifier which is imposing the constraints. As for subcategorization relationships, this ability eliminates the need to introduce node features to coordinate the constraints across grammar entries. For example, features are no longer needed to distinguish between the categories adjective and adverb, since the distinction can be expressed within the word's structure by specifying the category of the modified node. Adjectives are A's which modify N's and adverbs are A's which modify either V's or sometimes other A's. This in turn permits a single entry for 'very' which can modify both adjectives and adverbs simply by modifying A's. By specifying adjuncts in this way, multiple adjuncts can attach to a single node. The root node of each adjunct structure can equate with the node being modified without interfering with the attachment of the

[21] Remember that the presence of the underspecified *head* terminal for the $\overline{\overline{N}}$ makes it an obligatory argument.

other adjuncts. This iteration is possible because each adjunct brings with it the link necessary to be attached. In contrast, subcategorized arguments cannot iterate because the link for attachment is supplied by the subcategorizing structure, not the argument, and thus only one argument can attach. This technique for attaching adjuncts eliminates the need for Chomsky adjunction.[22]

The structure for 'the' in figure 3.5 is like the modification structures in that the terminal is not the *head* of the root, but it cannot iterate because the terminal is the *determiner* of the root. The link between the $\overline{\overline{\text{N}}}$ and the $\overline{\text{N}}$ is there in case the head of the $\overline{\overline{\text{N}}}$ is not a full $\overline{\overline{\text{N}}}$ by itself, such as is the case for 'tonka' in figure 3.5. The structure for 'who' in figure 3.6 has a similar basic configuration. Again in these structures, SUG's ability to express information within the structure associated with a word, rather than just in its category, permits node features to be eliminated. Given this analysis there is no need for the nonterminal category determiner. In fact the only categories which appear to be needed are the major categories, N, V, A, and P, with their bar levels.[23] This is an indication of how much more expressive a formalism with a large domain of locality, like SUG, is than a formalism like CFGs, in which much of the work in writing a grammar is working out a system of features to enforce constraints across grammar entry boundaries.

Because SUG allows the specification of dominance relations, long distance dependencies can also be expressed in its domain of locality. An example of this is given in the structure for 'who' in figure 3.6. In this structure, node w_1 acts as a trace, since it needs to find an argument position to give it an immediate parent, and it will fill an obligatory argument position by giving the argument node a filled head. The dominance relation restricts w_1 so it must equate to a node within the lower $\overline{\overline{\text{V}}}$, thus enforcing that 'who' must c-command its trace,[24] but it also allows w_1 to move arbitrarily far from 'who'. Other constraints on where a trace can equate can be enforced using node features, as will be shown in section 6.1.

Gerunds are a particularly good test for the domain of locality of a formalism because they act like noun phrases but have the internal structure of verb phrases. Figure 3.7 gives one possible structure for the gerund 'riding'. This structure includes the usual structure of a $\overline{\overline{\text{V}}}$, including the subcategorization for the object of 'ride'. However, the root of the structure is an $\overline{\overline{\text{N}}}$, thus making it fill $\overline{\overline{\text{N}}}$ argument slots.

The two possible structures for 'wants' in figure 3.8 give another example of the advantages of SUG's domain of locality. The verb 'wants' is followed by an $\overline{\overline{\text{N}}}$ and an infinitival $\overline{\overline{\text{V}}}$. The $\overline{\overline{\text{N}}}$ is semantically the subject of the $\overline{\overline{\text{V}}}$, but the $\overline{\overline{\text{N}}}$ gets its Case[25] from 'wants'. This leads to two possible structures for 'wants', one which follows the semantic structure and one which follows the Case

[22] If Chomsky adjunction is desired, then it can be accommodated by splitting $\overline{\text{X}}$ nodes into two $\overline{\text{X}}$ nodes with a dominance link between them. This allows a series of intermediate $\overline{\text{X}}$ nodes to be inserted between them to produce a Chomsky adjunction structure. If nothing is inserted the two $\overline{\text{X}}$ nodes can simply equate, since dominance is recursive, giving the usual unmodified structure. I do not adopt this analysis because I think Chomsky adjunction is an artifact of the inadequacies of CFG, not an insight of that investigation. In terms of the adjunct/argument distinction just discussed, CFGs only have the ability to specify subcategorized arguments.

[23] There are other features, such as tense and agreement, which could be argued to be part of the category of a node. Nevertheless, all nodes can be subcategories of N, V, A, or P and there is no need for any "extracategorial" nodes, such as determiner.

[24] C-command is a relationship often used in Government Binding Theory. The exact definition is not always agreed upon, but it always involves there existing a node which is a short distance above the c-commanding node and an arbitrary distance above the c-commanded node. In this case this node is the $\overline{\overline{\overline{\text{V}}}}$.

[25] In Government Binding Theory, Case is a formal notion closely related to case. All overt $\overline{\overline{\text{N}}}$'s must receive Case, even if their case is not overtly marked. 'Wants' is an exceptional Case marking verb because it assigns Case to the semantic subject of its object, rather than having the subordinate verb assign Case, as is true for 'said'.

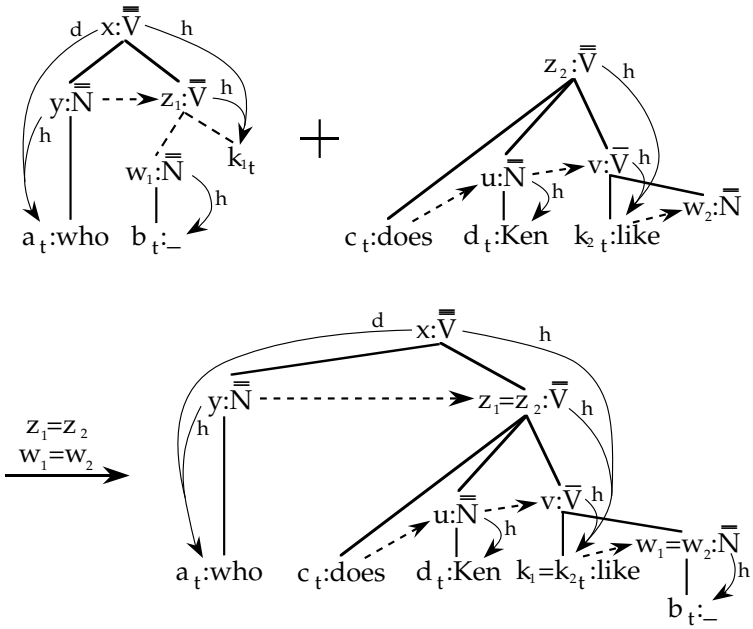


Figure 3.6: An example of using dominance to express long distance dependencies.

structure, as shown in the first and second structures in figure 3.8, respectively.[26] In either structure the relationship not expressed in the structure can be expressed over the nodes in the structure, and the case of the $\overline{\overline{N}}$ can be expressed. I will use the second of these structures because semantic information will have to be expressed separately anyway,[27] so it seems unnecessary to have the syntactic structure mimic the semantic structure. Also, Case seems like an inherently syntactic phenomena, so it is not clear what role it would play if not to determine the syntactic structure. Thus it is natural to assume that providing a node with an immediate parent acts analogously to Case assignment, only extended to all the categories. This is the definition of the constituent structure which NNEP produces. Under this interpretation the adjunct structures discussed above can be interpreted as saying that adjuncts assign themselves Case. This interpretation of Case is similar to Abney's (1986) notion of licensing, as is discussed in (Henderson, 1990).

3.3.2 Trading Ambiguity for Partial Specification

SUG not only provides the domain of locality necessary to state what constraints are known where they are known, it also allows those things which are not known not to be said. This is a natural consequence of using partial descriptions. By only partially specifying information, what would

[26] There are other possible analyses. One common analysis is to express the subcategorization for the subject with the infinitival verb, in the same way as would be done for tensed verbs, except the subject would be marked as needing Case. Given this, the structure for 'wants' would still need to mention the subject in its structure in order to say that it gives the subject Case. In accord with the idea that the grammar entry should say everything known, the subcategorization and subjecthood relationship would also be expressed in the structure for 'wants'. Given this and the fact that infinitival verbs do not always have overt subjects, it is not clear why the subcategorization for the subject should also be in the structure for the infinitival verb. For this reason I have not included this analysis in the example, but that is not to say it could not be done.

[27] The need to express semantic information separately from syntactic structure relations is argued for in section 3.2 in (Henderson, 1990), which compares SUG to Lexical Functional Grammar.

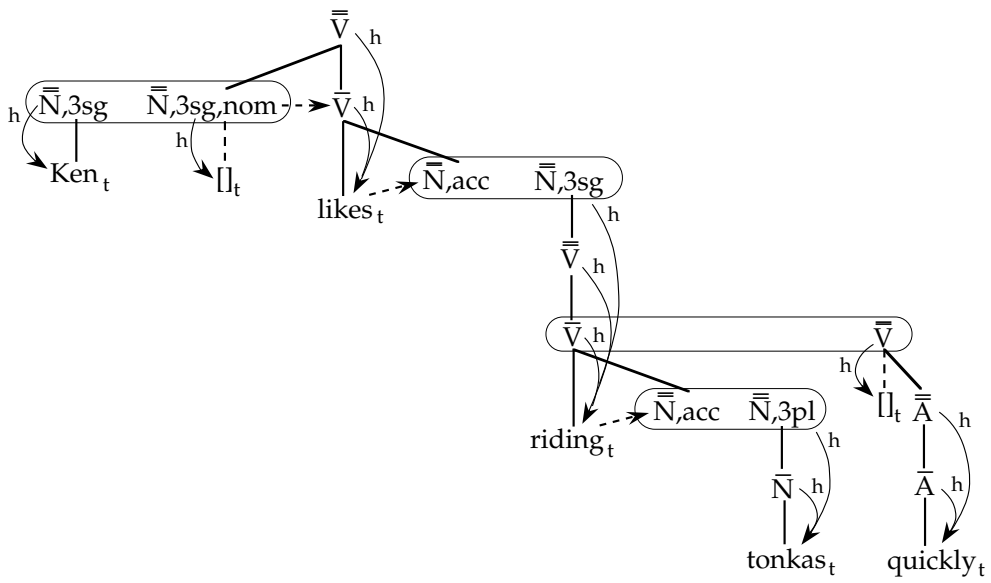


Figure 3.7: One possible grammar entry for the gerund 'riding' used to derive the sentence "Ken likes riding tonkas quickly".

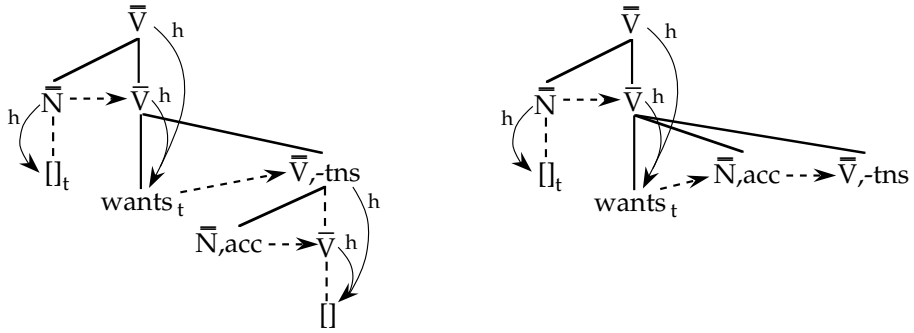


Figure 3.8: Two possible grammar entries for the exceptional case marking verb 'wants'.

otherwise be an ambiguity between multiple grammar entries can be expressed in a single grammar entry. This section will give a few examples of this ability.

Figure 3.9 shows how partial specification of node labels can be used to express ambiguities. Because feature structures are being used to label nodes, it is possible to partially specify these labels, and thus express ambiguity between multiple labels. To do this, however, it is necessary to use a feature decomposition of node labels which allows the desired ambiguities to be expressed. In the examples in figure 3.9, I use a feature decomposition of the major categories which differs from the Chomskian feature decomposition. N is represented as [−V,−A], V as [+V,−A], A as [−V,+A], and P as [+V,+A]. This allows one structure for 'knows' which allows for either an $\overline{\overline{\mathrm{N}}}$ or a $\overline{\overline{\mathrm{V}}}$ object, which would not be possible with the Chomskian feature decomposition. The second structure in figure 3.9 allows 'always' to attach to either a $\overline{\mathrm{V}}$ or a $\overline{\mathrm{P}}$.

Another kind of ambiguity was expressed in the second possibility for the structure for 'wants' given in figure 3.8. This structure expresses the fact that the objects of 'wants' are both optional. Remember that underspecified terminal heads are used to express obligatory arguments. Because the heads of the two objects of 'wants' are not mentioned, these objects do not have to be equated

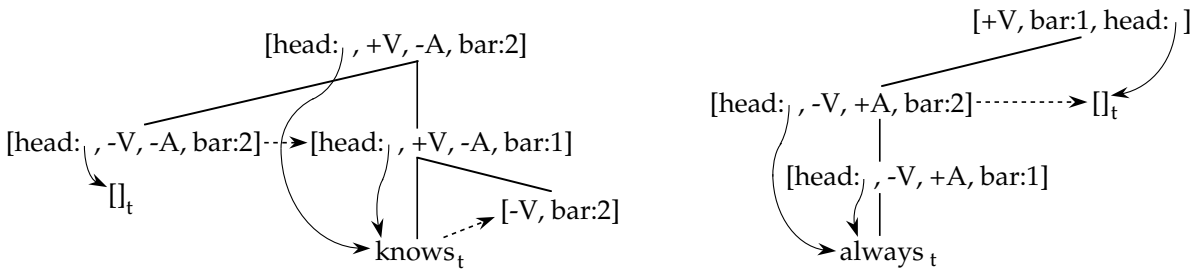


Figure 3.9: Examples of using feature structures to partially specify node labels.

with for the structure to be complete, and thus are optional. On the other hand, there is still nothing preventing a node with a head from equating with these nodes, so they are still subcategorized arguments.

3.3.3 Partial Descriptions of Phrase Structure in Other Formalisms

Many other formalisms can be viewed as combining partial descriptions of phrase structure trees to produce a complete description, but they do not have the properties which have been argued for in this section. This subsection discusses a few such formalisms: CFGs, PATR-II (Shieber, 1986), the system defined in (Rounds and Manaster-Ramer, 1987), Segment Grammar (de Smedt and Kempen, 1991) and FTAG (Vijay-Shanker, 1987). Some other formalisms which can be viewed in this way are discussed and compared with SUG in (Henderson, 1990).

One simple formalism which can be viewed as using partial descriptions of phrase structure is Context Free Grammars. Each rule in a CFG specifies a possible tree fragment of depth one. The expansion of a nonterminal in one rule by another rule corresponds to equating a leaf of one fragment with the root of another. In this sense each rule used in a derivation describes a fragment of the final tree. The problem is that the ways grammatical information can be grouped into CFG grammar entries is severely constrained. The division of grammatical information in a CFG is completely determined by structural configuration, with no influence from the interdependencies between constraints. One effect of this problem is that CFGs force all the information about a node to be specified in every grammar entry that mentions the node. Because node labels are atomic, partial specification of these labels is not possible, thus requiring different rules for each way of completely specifying the node label which should be partially specified. The same problem occurs due to the fact that the children of each fragment are completely ordered. Another effect is that the domain of locality of CFGs is very small. Since a CFG grammar entry is limited to being of depth one, any constraint which spans more than one level in the tree cannot be expressed in a single grammar entry. Systems of node labels can be devised to encode such constraints, but they suffer from the above limitation on node label specification and lose the perspicuity of the encoded constraint. Also, because a CFG grammar entry is interpreted as a complete description of the parent-child relationships for its root, all possible combinations of children for the parent must each be specified in a different grammar entry, rather than being able to modularize this specification according to co-occurrence restrictions. These factors prevent the perspicuous representation of many grammatical constraints.

PATR-II (Shieber, 1986) is an extension of CFGs which allows node labels to be specified using feature structures, including the ability to specify coreference between feature values in node labels. This greatly increases the power of the formalism, but it still suffers from most of the problems

discussed above for CFGs. In fact, the only problem it solves is CFGs' inability to partially specify node labels. Through the use of feature passing techniques, this ability in turn helps in the encoding of constraints which span more than one level in the tree, but such feature systems still lose the perspicuity of the encoded constraint. PATR-II still can't underspecify ordering constraints and can't modularize the specification of parent-child relationships according to co-occurrence restrictions.

A formalism which is formally similar to PATR-II, but which has much of the flexibility argued for above, is Segment Grammar (de Smedt and Kempen, 1991). A grammar entry in SG is a connected collection of node-arc-node triples, called segments. The nodes are labeled with feature structures, and the arcs correspond to immediate dominance relationships, plus a label specifying the role of the lower constituent in the upper one. Grammar entries are combined by equating nodes, which allows the children of a given node to come from more than one grammar entry. This makes SG very similar to SUG, which is not too surprising since the formalisms were developed (independently) to address similar issues, such as the incremental specification of syntactic information. SG differs from SUG in that it does not provide for the specification of dominance constraints, and linear precedence constraints are specified separately from grammar entries. As was argued above, dominance constraints are necessary for a formalism's domain of locality to be sufficient to directly express long distance dependencies.

The formalism described in (Rounds and Manaster-Ramer, 1987), *A Logical Version of Functional Grammar*, adds to feature structures the ability to specify dominance and linear precedence relations. The resulting logic is used in fixed point formulas to specify grammars. The process of instantiating the type variables in a fixed point formula provides the same kind of structure as CFG derivations, but this structure is not used to enforce ordering constraints. Ordering constraints are enforced using the dominance and linear precedence constraints, which may be unrelated to the variable instantiation structure. This system is less expressive than SUG because it does not have immediate dominance relations or instance unique terminals, but the most important problem arises from the use of fixed point formulas to specify grammars. Just as the instantiation of type variables has the same structure as CFG derivations, the specification of type variables in fixed point formulas has the same restricted domain for specifying grammatical constraints as CFG rules. Any constraint which spans more than one level in the instantiation structure can only be stated with the use of feature passing techniques. The other problems discussed above are avoided in this system because of the extensive use of partial information, including the unrestricted use of disjunction.

Tree Adjoining Grammars (Joshi, 1987) and its variants do have a sufficiently large domain of locality for expressing grammatical constraints. In these formalisms, the adjoining operation provides a mechanism by which two pieces of a grammar entry can be stretched arbitrarily far apart, thereby allowing long distance dependencies to be expressed within grammar entries. In (Henderson, 1990), the adjoining operation for Feature Structure Based Tree Adjoining Grammars (FTAG, Vijay-Shanker, 1987) is simulated in SUG using dominance constraints and a set of feature restrictions. The relationship between adjoining in tree rewriting systems and dominance constraints in description accumulating systems was discussed in detail in (Vijay-Shanker, 1992), where FTAG is cast in terms of accumulating partial descriptions of a tree. In that article Vijay-Shanker also discusses how this description based view of FTAG naturally generalizes to Multi-Component Tree Adjoining Grammar, which allows grammar entries and adjoinings to involve sets of trees. The result of this generalization is extremely similar to SUG. The only differences appear to be that FTAG has no instance unique feature values, and that the basic combination operation in FTAG is tree substitution, not node equation. Vijay-Shanker defines tree substitution in a way which is

equivalent to nonterminal node equation under the condition that one of the nodes have no immediate parent and the other have no immediate children. The restriction against equating two nodes which have immediate parents is not part of the definition of SUG, but it must be imposed in order to allow nodes to be forgotten, and it appears to be linguistically justified. This requirement will be discussed in the next section. The restriction against equating two nodes which have immediate children provides a mechanism for preventing equations even when the feature structures are compatible. For example, without this restriction a grammar entry could attach to the root of another instance of itself an arbitrary number of times. Thus, this restriction plays the same role as instance unique feature values in SUG. However, conflating a restriction against iteration with a structural property doesn't allow sufficient flexibility in the specification of these restrictions. While this conflation doesn't effect the formal power of the formalism, it requires multiple nodes to be used in cases where SUG can use a single node.[28] A clear example of this is TAG's need for Chomsky adjunction in those cases where arbitrary iteration is allowed. This forces two nodes to be represented at any site where a modifier could attach, and it prevents the formalism itself from distinguishing between Chomsky adjoining structures and what in the linguistic work on TAG are called complement auxiliary trees. Another example is the need for multiple nodes in the structure of noun phrases and clauses. In the structure of noun phrases there needs to be a node whose children prevent the iteration of the determiner and another node whose children prevent the iteration of the noun head. In the structure of clauses there need to be such nodes for the complementizer, the inflection, and the verb. Because of the extremely limited memory capacity of the S&A computational architecture, the need to use multiple nodes for these two purposes makes such a formalism inappropriate for use in the parser. In contrast, SUG allows constituent structure to be represented using only one node per N, V, A, or P head. The ability to not specify a word as the value of the *head* feature avoids the need for Chomsky adjunction, and the ability to use multiple features with terminal values (e.g. *head*, *determiner*, *complementizer*, *inflection*) allows the iteration of multiple things to be controlled at a single node. This compact representation will be discussed in depth in section 6.1. Sections 6.1 and 6.2 directly address how to adapt the linguistic work which has been done in the TAG formalisms to SUG.

3.4 Forgetting Grammatical Information in SUG

One of the constraints on symbolic computation in the Shastri and Ajjanagadde architecture is that at most ten variables can be used at any one time. Since the parser proposed in the next chapter uses variables to refer to nonterminal nodes in the phrase structure of the sentence, the number of nonterminals which the parser needs information about at any one time needs to be bounded. Since arbitrarily many nonterminals are needed to represent complete phrase structure trees for natural language sentences, the parser needs the ability to abstract away from the existence of some nodes during the course of a parse. The use of partial descriptions in SUG provides for abstracting away from information. In SUG this abstraction ability is used to keep from having to specify information which has not yet been determined. This abstraction ability can also be used to keep from having to represent information which is no longer needed. The forgetting operation described in this section is used to abstract away from the existence of nodes which are no longer needed to complete the parse. This provides a mechanism which the parser can use to stay within the bounds on the number of nonterminal nodes which can be represented at any one time.

[28] Note that Context Free Grammars and all the formalisms based on them have this same property, so it is not surprising that the techniques for working around it are common practice in linguistics.

One class of grammar formalisms which can be used to parse arbitrarily long sentences with bounded resources are those based on Categorial Grammar. As discussed in section 3.1, this is because these formalisms represent grammatical information in categories which can be viewed as abstract types for tree fragments. For example, the category NP is the type for tree fragments which have an NP node as their root and have no missing constituents. The categories of the form X/Y are types for tree fragments which would be tree fragments of type X if they were to combine with a tree fragment of type Y. Because these categories only represent the root node and the missing constituents, they abstract away from the completed constituents which the tree fragments contain. Combinatory Categorial Grammar (CCG, Steedman, 1987) even provides a mechanism for abstracting away from nodes before their constituent is completed, as long as the node itself is complete. This mechanism is the composition operation, which is illustrated in figure 3.10. For concreteness, consider the example given in the second line of that figure. Under the tree building interpretation of this operation, it equates the IP argument of "said" with the IP node for "like" to fulfill the expectations of these nodes. In the resulting structure the constituent for "like" is still incomplete, since it still needs a tree fragment with root N. But since the local expectations of "like" 's IP node have been fulfilled through the equation, this node can be removed from the CCG syntactic category of the sentence fragment. In this way, sentences with an arbitrary number of nodes in their phrase structure can be derived using categories with only a bounded number of basic categories in them, as long as at any given point in the sentence the number of nodes which have local expectations is bounded.

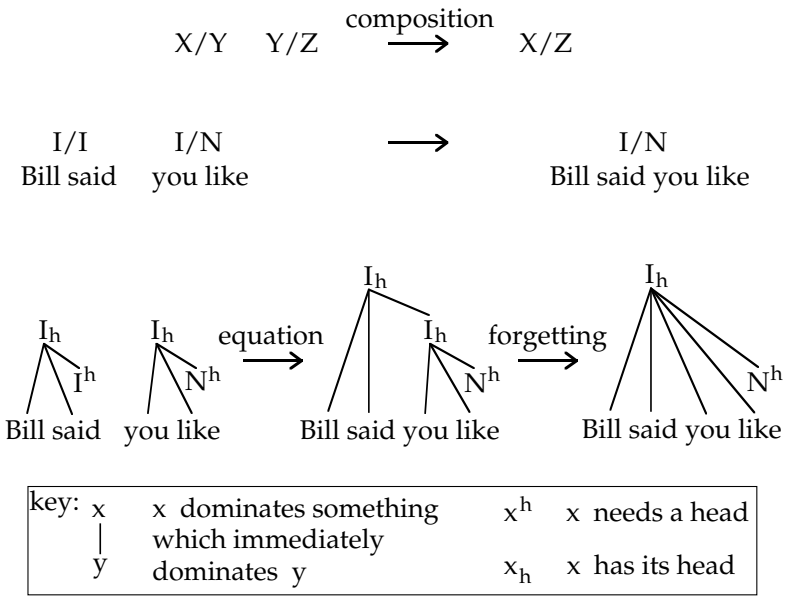


Figure 3.10: Abstracting away from the existence of certain nodes in CCG and in SUG.

Unfortunately, the representation of grammatical information used in CCG is not adequate for parsing in the Shastri and Ajjanagadde architecture. Not only *can* you forget nodes whose local expectations are fulfilled, you *must* forget such nodes. Because natural languages have optional modifiers which can come after all the expectations of the node they modify have been fulfilled, this greedy forgetting strategy cannot be used. In CCG this problem can be avoided by either delaying the equations which fulfill a node's expectations, or by introducing an expectation for the optional modifier. The first case would not provide the required incremental output, since the relationship between the nodes would not be output until well after that relationship could be

determined. The second case could not be done deterministically, since the decision of whether to introduce the expectation for the modifier would have to be done well before the presence or absence of such a modifier could be determined. Thus we need to separate the process of equating nodes from the process of forgetting nodes whose local expectations have been fulfilled.

SUG descriptions can also be viewed as types for tree fragments. Like CCG categories, an SUG description is a type for all those tree fragments which have all the characteristics specified in the description, and have exactly the expectations specified in the description. SUG uses the opposite strategy from CCG for removing nodes from the syntactic type of a sentence fragment. SUG never forgets nodes, even when it is impossible for those nodes to be involved in any more equations. Parsing in the S&A architecture requires that it be possible to forget nodes which will not be involved in any more equations. To have such forgetting, the language SUG uses to describe phrase structure trees needs to be slightly altered. As was determined in the discussion of CCG in section 3.1, rather than specifying immediate dominance relationships, the descriptions need to specify that a node has an immediate parent without explicitly specifying what node that parent is. Dominance relationships can still be used to specify the structure of a tree, so the immediate dominance relationships can still be recovered from a complete description of a tree. The only effect this change has on the formalism is that now two nonterminal nodes which both have immediate parents can never be equated. Such an equation would require that the immediate parents of the nodes also be equated, but since the immediate parents can't necessarily be identified, this equation can't be done.[29] This difference has no impact on the generative capacity of SUG, and none of the SUG examples given above require the equation of two nodes with immediate parents. With this change to SUG's descriptive language, an SUG description which has the same information as a given CCG category is a type for the same tree fragments as the CCG category is a type for. Thus these descriptions have the same ability to represent arbitrarily large trees in a bounded representation as does CCG.

The advantage of these descriptions over CCG categories is that nodes don't have to be forgotten when their expectations are fulfilled. Both the minimally informative CCG representation and the maximally informative SUG representation of a given tree fragment can be used. To change to a representation which has less information, the *forgetting* operation is used. Once a node has been forgotten it can't be equated with, so only nodes which do not need to be equated with in order for the description to be complete can be forgotten. Forgetting a node simply abstracts away from the existence of that node and all the information about it. This information can safely be forgotten without allowing future operations which are incompatible with this information, because all the forgotten information is specific to the forgotten node, and no future equations can be done with that node. This operation is illustrated in figure 3.10, and is defined in section 4.3.3.

It should be stressed that the forgetting operation only applies to the representation used by the syntactic constituent structure parser. Forgetting a node in this representation does not necessarily mean that the associated node(s) in higher level representations are also forgotten. Because the parser's output is incremental, all the information which the constituent structure parser has discovered about that node has already been output to other modules before the node is forgotten. Thus forgetting will not interfere with the interpretation of the output of the parser. Forgetting nodes will also not interfere with determining whether the parser has produced a complete description at the end of a parse, since only locally complete nodes can be forgotten, and a description is complete if and only if each node is locally complete.

[29]Such equations need to be ruled out by the parse anyway, because in general zipping up a structure like this can require an arbitrary number of equations to be checked and done for a single combination of two structures.

The addition of the forgetting operation to Structure Unification Grammar results in a grammatical framework which has the properties needed for parsing in the S&A architecture. SUG is sufficiently partial and sufficiently expressive to allow the parser to store all and only what it knows about the phrase structure of the sentence, specifying expectations and iteration restrictions separately from structural constraints allows a compact phrase structure representation, and the forgetting operations allows the parser to store only what it needs to know about that phrase structure. These properties are possible because of the extensive use of partial descriptions.

Chapter 4

The Parsing Model

The argument being made in this dissertation is that the Shastri and Ajjanagadde connectionist computational architecture is computationally adequate and linguistically significant for recovering the constituent structure of natural language sentences. Chapters 2 and 3 developed a level of representation which is appropriate for investigating parsing in the S&A architecture. This chapter uses that framework to present a specific parser, and to argue that it has the necessary properties to be implemented in the S&A architecture. Chapter 5 then shows how this implementation has been done. In chapter 6, this parser will be tested in various ways to argue for the adequacy of the architecture, and the mechanisms the parser uses to compensate for the architecture's limitations will be used to argue for the linguistic significance of the architecture.

The model of parsing presented in this chapter computes Structure Unification Grammar derivations. Descriptions of phrase structure trees are taken from the grammar and combined using node equations so as to produce a complete description of a phrase structure tree for the input sentence. Because the primary job of the parser is to find and perform node equations, it is called the Neural-network Node Equating Parser, or NNEP. NNEP can only compute one derivation at a time, but SUG is designed for deterministic parsing. Also, only one SUG description can be stored in NNEP's memory at a time, but since SUG descriptions can have multiple unconnected tree fragments, this isn't really a constraint. NNEP's operations compute SUG derivation steps, plus forgetting. Most of these operations combine the description in the parser state with a description from the grammar. Some of these operations equate nodes which are already in the parser state. To successfully parse a sentence, grammar entries for each word in the sentence need to be combined with the parser state in the order in which the words appear in the sentence, and the final description must be a complete description. NNEP's output is the sequence of derivation steps which it computes. This output conveys all the information which would be conveyed by the total final description. In addition, this output is maximally incremental, and forgetting information does not interfere with it. During the course of a parse, NNEP may need to make choices, and these choices may require input from other language modules. A plausible disambiguation mechanism has been designed and implemented for this parsing model, but it has not yet been adequately tested.

As discussed in chapter 2, parsing in the Shastri and Ajjanagadde connectionist computational architecture must be done under some constraints. These constraints have a number of implications for the parsing model. Some of these implications are due to the constraint that only one variable can be in both the antecedent and consequent of each of the rules which implement the parser. This restricts the set of operations which the parser can use to calculate SUG derivation steps. Some operations the parser needs require more than one nonterminal node to be involved, but they

can be implemented by requiring all but one of these nodes to be uniquely identifiable. Rules can refer to a uniquely identifiable node using a constant, and thus these operations' rules only need to propagate information about one variable. In order to make all but one nonterminal involved in an operation uniquely identifiable, a stack needs to be introduced, and some operations need to be constrained to only apply to the top node on this stack. The rules which calculate possible long distance dependencies also need to use this stack. This mechanism imposes some of the linguistic constraints on long distance dependencies, as discussed in chapter 6. The constraint that the S&A architecture can only use a fixed set of predicates means that the depths of this stack is bounded. For the same reason, the length of the list of tree fragments in the parser state is also bounded. These bounds have implications for NNEP's ability to parse center embedded sentences, which will be discussed in chapter 6. The resource bounds of the architecture also mean that at most ten nonterminals can be stored in the parser state at any one time. Because the grammatical framework discussed in the last chapter allows certain nodes to be forgotten during a parse, NNEP has the mechanism necessary to stay within this bound. The linguistic implications of this constraint will also be discussed in chapter 6.

This chapter describes the parsing model in detail. It begins with a discussion of what properties the parsing model should have, and what existing approaches to syntactic parsing have some of these properties. The second section characterizes the grammars which NNEP supports. Not all SUG grammars can be used by NNEP, not all SUG derivations for a given grammar can be computed by NNEP, and some additional assumptions are made for the purposes of this investigation. The third section defines NNEP's operations. There are four operations for combining descriptions from the grammar with the current description, two operations for equating nodes within the current description, and one operation for forgetting nodes. The fourth section then presents how the grammatical information needed by the operations is represented in the parser state, and how that information is kept up to date as the description changes. In the fifth section, the mechanism NNEP is currently designed to use to make disambiguation choices is discussed. The last section defines NNEP's output, which is simply the derivation structure that was followed in the parse.

4.1 Related Approaches to Syntactic Parsing

In chapter 2, computation in the Shastri and Ajjanagadde architecture was characterized as symbolic computation under a set of constraints. This allows the investigation of parsing using the S&A architecture to be done at the level of symbolic computation. Chapter 3 presented a grammatical framework for parsing within the constraints, but because these constraints are computational in nature, that representation does not fully reflect the constraints' significance. This section characterizes what additional requirements these computational constraints impose on a syntactic parser implemented in the S&A architecture, and discusses previous work on syntactic parsing that has addressed some of these requirements. The parsing model discussed in the rest of this chapter embodies these requirements.

4.1.1 Requirements for the Parsing Model

Chapter 2 identified the following set of constraints on parsing in the S&A architecture.

1. information about at most ten nodes stored at a time
2. no explicit representation of disjunction between predications
3. at most three tuples of unary predicates for storing relations
4. at most one variable can appear in both a rule's antecedent and consequent
5. input is incremental
6. parse in quasi-real time
7. produce maximally incremental output
8. produce monotonic output

In addition, some effort should be made to minimize the use of rules that involve multiple phrase structure nodes. As was discussed in chapter 2, constraints 2, 7, and 8 mean the parser must be deterministic, in the sense of (Marcus, 1980).

Chapter 3 identified two requirements for the parser's grammatical framework that influence the design of the parsing model. First, the grammatical representation must allow information which the parser does know at a given time to be specified independently of the information which the parser does not know. This allows compliance with constraints 2, 7, and 8 (i.e. determinism). Second, the representation should allow as much information as possible to be local to individual phrase structure nodes, and as little information as possible to be expressed as relationships between nodes. This allows compliance with constraints 1, 3, and 4, and with the desire to minimize the use of rules that involve multiple nodes.

The determinism constraint (constraints 2, 7, and 8) also places requirements on the process of constructing a representation of the sentence's phrase structure tree. The parser must figure out what can be inferred about the phrase structure of the sentence, and store it. To figure out what can be inferred, the parser needs access not only to its own state information, but also to information from other modules of the language processing system. Altmann and Steedman (1988) showed that the expectations of the anaphora resolution process can affect decisions about syntactic structure. The extent to which other kinds of nonsyntactic information is necessary for making decisions about syntactic structure is not yet clear, but it is clear that there needs to be some mechanism for allowing input from other modules to influence the parser's decisions. Also, to simplify the addition of information to the parser state, the grammar needs to be organized according to the inferences that the parser can make. Because new information comes to the parser in the form of words, the parser's grammar entry for a word should include all the information which is implied by the presence of the word. Thus the parser should have a predominantly lexicalized grammar. The only cases where information should be placed in grammar entries which do not include words are those where no words reliably imply the information, such as relative clauses that have no overt complementizers. Given such a grammar, the parser can simply add the grammar entries that correspond to the chunks of information that it can infer. Because the construction of the phrase structure description has to be information driven in this way, the parser cannot expand phrase structure nodes in any fixed order. Many parsing algorithms traverse the tree in a fixed order (bottom up, top down, left corner, etc.), considering what information to add to each node one at a time. A deterministic parser needs to add information according to information dependencies, not according to structural configuration. The nature of these information dependencies is an empirical issue, but previous investigations of deterministic parsing and lexicalized grammars give an indication.

The need to localize computation to individual phrase structure nodes (constraints 1, 3, and 4) has a number of implications for the design of the parser. The two tasks of the parser are deciding what grammar entries to use and deciding what pairs of nodes in these grammar entries need to

be equated. The later task is highly constrained by the need to localize computation to individual nodes. One way to equate nodes without violating this constraint is to use rules that are specific to grammar entries. This allows information about nodes in the grammar entry to be compiled into the rules, so this information can be accessed without using either variables, or predications about the situation as a whole. For example, the rule that calculates whether a given grammar entry can attach to a node in the parser state only needs to test predications about the node in the parser state, because the information about the root of the grammar entry is compiled into the rule. However, even with this technique, the number of ways that a grammar entry can be combined with the phrase structure description in the parser state is limited. In general only a single node in the parser state can be equated with a node in the grammar entry, but because of a mechanism introduced for calculating long distance dependencies, subjects can sometimes be attached to their sentences at the same time as the sentence is attached. This limitation may help explain some differences between head final/case marked languages and head initial/non-case marked languages, but since the scope of this dissertation is being limited to English, no literature on this (rather broad) subject will be discussed.

When two nodes that are both already in the parser state need to be equated, then one of the two nodes needs to be uniquely identifiable at the time. Given this property, only the identity of one of the nodes needs to be propagated from the pattern to the action of the rule that equates the nodes, and thus constraint 4 does not need to be violated. The two cases when nodes in the parser state need to be equated are when an unattached tree fragment needs to be attached, and when a gap for a long distance dependency needs to be equated with its trace node. For English, the way to restrict these equations to make one of the nodes uniquely identifiable is clear. Because English dependencies are nested, attachment equations can be restricted to only involve the rightmost tree fragment root. Following Pesetsky's path containment condition (Pesetsky, 1982), gap filling equations can be restricted to only involve the most recently introduced trace node. The calculation of the candidate equations obeys these same constraints.

The only other calculation that the parser needs to do that in general would involve multiple nodes is the enforcement of ordering constraints. Most ordering constraints can be expressed with respect to a word (e.g. prehead, post-determiner), so they don't involve more than one nonterminal node. The one case where this cannot be done is the ordering constraint between the objects of a ditransitive verb. In terms of competence grammar, there are situations where more than one such relationship would have to be inherited down the tree at the same time. In general, this would require rules that violate constraint 4. However, constraints on the nesting of ditransitive verbs prevent exactly these situations from occurring, so these ordering constraints can be enforced with rules that don't involve multiple nodes. I know of no previous investigations that address this issue.

The S&A architecture's bounded memory capacity also affects the design of the parser. Information about only a bounded number of phrase structure nodes can be stored, the parser has a fixed set of predicates, and only a bounded number of bindings for representing binary relations can be stored. The first constraint requires the parser to close off nodes from further consideration, so that they can be removed form the memory (through the use of the forgetting operation, discussed in section 3.4). This investigation does not propose any specific node closure strategy, but previous investigations of this issue indicate what such a strategy should be like. The fact that the parser's predicates are fixed means that all data structures are of bounded size. The two data structures that would naturally be of unbounded size are the list of unattached tree fragments and the stack of trace (and sometimes subject) nodes. The list of tree fragments is needed to keep track of word order information, and the stack of trace nodes is needed to keep track of which one was introduced most recently. Bounding the size of these data structures places constraints on center embedding,

which has been investigated by several researchers. The bounds on these data structures turn out to subsume the bound on the number of bindings that can be stored for representing relations between nodes, so this later constraint doesn't have any additional effect on the design of the parser. However, some recent work on modeling center embedding phenomena proposes a very similar general constraint that covers some of the data that is not ruled out by this account.

4.1.2 Inferring New Information

Deterministic parsing requires new information to be added to the parser state based on the information dependencies in the language. As discussed above, this requires the use of a predominantly lexicalized grammar, since words are the primary source of new information to the parser. Schabes (1990) investigated lexicalized grammars, particularly Lexicalized Tree Adjoining Grammars (LTAG). LTAG has two combination operations, substitution and adjunction. Substitution allows the arguments of a phrase to be specified independently of the phrase itself. Adjunction allows adjuncts and long distance dependencies to be specified independently of the phrases they modify or span. These mechanisms allow all the information in a natural language grammar to be associated with individual words (i.e. grammars can be lexicalized). Schabes argues that a parser can be more efficient using a lexicalized grammar, because it can restrict its attention to that portion of the grammar that is relevant to the sentence at hand. This division of grammatical information into groupings that are convenient for the parser is the same argument that was given above for lexicalized grammars, and the relevance of information is determined by the information dependencies discussed above. Thus the grammar entries Schabes and his colleagues use in their lexicalized grammar of English (Abeillé *et al.*, 1990) are an indication of how grammatical information should be grouped in NNEP's grammar. This grammar obeys the generalizations from linguistic work in TAG, namely that long distance dependencies and semantic dependencies should be local to grammar entries. For example, a predicate and all its argument phrases can be specified in a single grammar entry, as can a modifier and its modified phrase. Unfortunately some properties of adjunction in TAG prevent the information about a long distance dependency from being associated with its wh- word, but these dependencies can still be localized within the grammar entry for the wh- word's sentence's verb.

The problem just mentioned with associating information about long distance dependencies with their wh- word is an example of a general problem LTAG has with expressing information dependencies involving words other than the semantic head of a phrase. There is a general tendency in linguistic work to assume that semantic dependencies are the core dependencies (e.g. "Theta Criterion" versus "Case Filter" in GB), but since this investigation is concerned with syntactic constituent structure, and not predicate-argument structure, case and other primarily syntactic dependencies are the central concern here. The importance of information dependencies other than those arising from semantic heads was recognized in work on Description Theory (Marcus *et al.*, 1983). D-Theory has two mechanisms for specifying grammatical information, a form of context free grammar and a set of templates that trigger parser actions. The later mechanism is used for such things as starting the construction of an NP when a determiner is found. This mechanism allows the parser to take advantage of the fact that the leading edge of a phrase is often very informative, even when it carries relatively little semantic information. For example, the semantically vacuous complementizer "that" is often used to start an embedded clause, especially if the subcategorizing verb can also take a noun phrase object that the subject of the sentence could be confused with (Trueswell, personal communication). The importance of information dependencies other than those arising from semantic heads was also recognized by Kimball in his New Nodes

principle: "The construction of a new node is signalled by the occurrence of a grammatical function word" (Kimball, 1973). From these other investigations we see that it is important for the parser to use grammar entries that express the information available from words other than semantic heads.

The nature of the output produced by a D-Theory parser is also informative for determining what kinds of information dependencies are needed to infer new phrase structure information. A D-Theory parser does not necessarily output a complete phrase structure tree. As in work done earlier on parsing spontaneous speech (Fidditch, (Hindle, 1983)), the output is only a partial specification of the sentence's phrase structure tree. In Fidditch, the output can be a sequence of subphrases of the sentence, and in D-Theory later stages of processing are allowed to add statements to the description that change the standard interpretation of the sentence. In the domain of spontaneous speech, this partial output is necessary because there may not be a complete analysis, since people don't always speak in complete sentences. In more controlled domains or for more abstract investigations, the use of partial output is justified because of the need for semantic information to make some disambiguation decisions. Because these investigations assume that no semantic information is available to the syntactic parser, they must take this approach. In this investigation I am assuming that semantic information is available to the syntactic parser, through input from other language processing modules. Thus the syntactic parser is in a position to make these disambiguation decisions before it is finished its output. Thus since only complete sentences are being considered here, NNEP is required to output a complete description of the phrase structure of the sentence. This issue was also discussed in section 3.1. As mentioned there, it is possible to loosen the requirement for a complete analysis through the more extensive use of soft constraints, thus allowing NNEP to handle less idealized domains, such as spontaneous speech, without losing the formal basis for expressing expectations.

4.1.3 Modeling Memory Bounds

Most of the work on modeling memory bounds has been concerned with sentences where it is clear what information needs to be kept in the memory, but the human parser is still not able to process the sentence. This phenomena only occurs in center embedded sentences, which are also the only sentence structures that cannot be parsed using only a bounded amount of memory (Chomsky, 1959). The question that research on this subject has tried to address is the nature of these bounds. Kimball (1973) proposed the simplest theory of these bounds in his Two Sentences principle: "The constituents of no more than two sentences can be parsed at the same time". Unfortunately, much of the data now available is incompatible with this simple statement (Gibson, 1991). The greatest empirical coverage has been achieved with a rather more complicated theory proposed by Gibson (1991). He associates load with three kinds of locally unsatisfied linguistic requirements, and bounds the sum of this load. Given the differences in parser design, grammatical representation, and form of resource bound, it is difficult to compare this theory with the constraints on NNEP's resources. However, the types of nodes that require the use of NNEP's bounded resources (subjects and trace nodes) are also associated with load in Gibson's theory, so it is possible that a thorough comparison between the two would reveal NNEP's bounds to be a subset of the resource bounds that underlie Gibson's success. Since this investigation is concerned with the implications of a particular set of constraints, and not with the coverage of a particular set of phenomena, the fact that NNEP's constraints don't rule out all the unacceptable sentences identified by Gibson is not of great concern here. It is also interesting that Gibson associates load with locally unsatisfied linguistic requirements that all involve relationships between nodes. This property is even more apparent in Lewis' work on center embedding. Lewis (1993) proposes that unacceptable center

embedded sentences are due to interference between multiple instances of the same unsatisfied grammatical relation. If more than two of the same kind of node are looking to assign or to receive the same relation, then the parser cannot store them all. While this and other work using the S&A architecture assume that up to three instances of relations can be stored, the similarity between these two constraints is very interesting. There seems to be a general convergence, from both computational and linguistic perspectives, on modeling center embedding through bounds on the resources necessary to identify relationships between nodes, although the precise nature of all these resources and bounds is still unclear.

Memory bounds can also affect the behavior of the parser in cases where it isn't clear what information needs to be kept in the memory. In particular, a long right branching sentence could conceivably have a modifier at the end that modified any one of the nodes on the right frontier. If such modification were possible in practice, the parser would have to store all these nodes just in case such a modifier occurred. This would require more than the ten nodes that NNEP has space to store. Fortunately, such posthead modification is highly restricted. Church (1980) proposes the A-over-A early closure principle as a theory of these restrictions. It closes a node when there is another node of the same category below it and both nodes have their parents, have all their obligatory children, and don't immediately dominate the next phrase to be parsed. Once a node is closed, it isn't available for further syntactic processing, so it can be removed from the parser's memory. Assuming only the four major categories, this principle results in a maximum of six unclosed nodes for right branching structures, well within the bound of ten. This principle essentially closes a node whenever there is a more recent node that will compete for the same posthead modifiers, and the next word doesn't attach to either node. One exception is that the matrix root of the sentence is never closed, since it never has a parent. The need for a preference for both high and low attachments is also shown in recent work by Gibson, Pearlmutter, Canseco-Gonzalez, and Hickok (1993). They argue for two interacting attachment preferences, one for more recent nodes of a given type, and one for the arguments of verbs. For the sentences they investigate (subject NP, followed by two PPs and a relative clause), this has the effect of preferring high and low attachments over middle attachments. Assuming the parser's strategy for forgetting nodes is approximately optimal, it would always forget the nodes which are least likely to be attached to. Thus the results of (Gibson *et al.*, 1993) support the same type of node closure strategy as proposed by Church (1980). Given the compact representation of phrase structure used by NNEP, these results indicate that the bound on the number of nodes that NNEP can store is not likely to be a problem for maintaining possible attachment sites.

4.2 NNEP's Grammars

The parsing model proposed in this chapter does not support the full range of grammars provided for in Structure Unification Grammar, but it does support a substantial subset. All the examples discussed in this document are either within this subset, or could be translated into an appropriate form. In addition, not all derivations allowed in SUG can be calculated by NNEP. There may be some sentences which are in the language of an SUG grammar, but which can not be parsed by NNEP using that grammar. This difference between the interpretation given to a grammar by the grammar formalism and the interpretation given to the same grammar by the parser is the usual competence performance distinction, but there are some differences between the way the distinction here divides phenomena and the division traditionally attributed to that distinction. This section discusses in what ways the grammars and derivations are restricted by the parser.

4.2.1 Restrictions on the Grammars

The constraints on the parser impose some definite constraints on the grammars which NNEP can support, but mostly they just make some grammatical specifications easier for a parser to deal with than others. Thus most of the restrictions on possible grammars are the result of a compromise between what is easy for a parser and what is needed for specifying natural language grammars. The first restriction on grammars is required in order to represent feature structures with a fixed set of predicates. No grammars are allowed for which some node's feature structure could grow arbitrarily large during a derivation. This restriction is commonly used to prevent a formalism which mixes feature structures with structural information from being undecidable. Because with this restriction there is a maximum to the size of a feature structure and there are only a finite number of labels and atoms[1], there are only a finite number of possible feature structures producible by a derivation. This allows feature structures to be represented with a finite set of predicates over nodes, thus eliminating the need to represent each embedded feature structure as an entity.

Another restriction is that no grammars are allowed for which one equation may cause as a side effect the equation of another pair of nonterminal nodes. This constraint is motivated by the locality constraint on rules. In order to perform an equation which has another equation as a side effect, both these equations need to be checked and done at the same time, which would in general require rules which propagate information about multiple variables. There are two ways one equation could cause another equation, through the constraint that only one node can immediately dominate a given node, and through the unification of the feature structures of the equated nodes. The first case is already ruled out because two nodes which both have immediate parents can't be equated, as was discussed in section 3.4. The prohibition against the second case in general excludes the use of features with nonterminal nodes as values. Features may still have terminals as values, since terminals are not represented with variables. Most cases which seem to require features with nonterminals as values can be handled by representing both nonterminals as a single nonterminal with a more complicated feature structure. This technique is used extensively in the grammars which are used in testing NNEP, in part for this reason and in part because it reduces the number of nonterminals which need to be stored in the parser's memory.

The remaining restrictions on grammars are less well motivated, but do make the parsing model simpler. A grammar entry can contain at most one phonetically realized terminal. Otherwise decisions about when to use grammar entries in the course of parsing would be complicated. Also, no node can have two nodes which both dominate it but neither of which dominates the other. Such a configuration implies that one of the dominating nodes dominates the other, and the implications of this disjunction are hard for NNEP to reason about. In addition, any node which does not have an immediate parent can not have nonredundant linear precedence constraints between it and another nonterminal. In other words, any linear precedence constraints between a nonterminal and an unparented node must be inherited from nodes which dominate the unparented node. The exclusion of such linear precedence constraints simplifies the calculation of long distance dependencies, since these constraints don't need to be checked when finding possible equations for an unparented node. This restriction also means the parser does not have to calculate the inheritance of the constraints, and thus does not have to represent some dominance information which would otherwise

[1] Because instance unique strings are used as atoms, there are actually an infinite number of atoms. However, because instance unique atoms are unique for each time they are introduced into a derivation, the identity of these atoms is not needed, only their existence. Replacing the instance unique string 'Barbie342' with 'Barbie343' in a set of feature structures will not in any way change the functionality of that set of feature structures to the parser. Thus there are only a finite number of inequivalent feature structures, which is all that is necessary.

be needed. The last restriction on grammars is that only nodes which dominate a word can have an immediate child. Thus the parent for any immediate dominance relationship in a grammar entry must dominate a word. This eliminates the need for the parser to inherit the property of dominating a word, which again frees the parser from having to represent some dominance information.[2] This set of constraints is sufficient to ensure that a given grammar can be effectively used by NNEP.

4.2.2 Restrictions on the Derivations

Because of the flexibility of SUG derivations, the parsing process described below can follow SUG derivations directly. Each change to NNEP's partial structure description corresponds to a derivation step, and each parser state corresponds to a description in a derivation. However, not all derivations can be followed by NNEP. Because NNEP only stores one description in its memory, the derivation must consist of a single sequence of descriptions, each of which is combined with at most one grammar entry to produce the next description.[3] The order in which grammar entries are combined with this sequence must respect the incrementality of the parser. Thus the grammar entries which contain words must be combined in the same order as these words appear in the sentence. Grammar entries which do not contain words can be combined at any time. These restriction on derivation structures by themselves do not actually limit the possible derivations in any significant sense. In SUG two descriptions can be combined without equating any nodes and equations can be done in a description without combining it with any other descriptions, so the actual structure of a derivation may have little connection to the phrase structure being built. These restrictions by themselves do not restrict the trees which can be built from any given grammar.

Although grammar entries which do not include words can be combined at any time in a derivation, the need to parse in real time means that only a bounded number of such nonlexical grammar entries can be combined between any two lexical grammar entries. This means that the number of nonlexical grammar entries used in a derivation can not exceed some linear function of the length of the sentence. This restriction does limit the power of the system. Any analysis which requires the size of a resulting structure to grow more than linearly with the length of the sentence can not be handled. However if there are no such constructions in a grammar, then the actual bound on the number of nonlexical grammar entries which can be combined between any two words does not effect the structures which can be built. This is because an equivalent grammar can be produced by precombining any finite number of nonlexical grammar entries with lexical grammar entries. In the extreme case where the bound is set to zero, this process results in a completely lexicalized grammar. Whether to make a cluster of information a nonlexical grammar entry or part of some lexical grammar entries is part of the larger question of how generalizations in the grammar should be represented and used in parsing. In this investigation generalizations are generally compiled out, and therefore the grammar entries are generally lexicalized. Compiling generalizations makes parsing more straightforward, but it does not have to be done in all cases. Compiling generalizations fails to explain a number of phenomena, for example our ability to handle words we have never seen before. Since these issues are outside the scope of this work, I will not impose any particular

[2]This restriction technically prevents the parser from using nonlexical grammar entries with more than one node in them, but such grammar entries can still be used if the immediately dominating nodes are specified as dominating a word, despite the fact that there is no word in the grammar entry. NNEP will interpret this to mean that the empty "word" is in the current position in the sentence, but it will still try to find a grammar entry for the next word in the sentence.

[3]The information about grammar entries are compiled into the rules which implement NNEP's operations. Thus grammar entries do not have to be stored in the memory to be used.

bound on the number of nonlexical grammar entries which can be combined between two words. For the data addressed in chapter 6, this bound could be one.

The limited set of operations used by NNEP to equate nodes restricts the derivations which can be computed. These operations never equate more than two pairs of nodes at a time. The implications of this restriction for the set of phrase structure trees which can be built will be discussed after the specific operations have been given.

The derivations which can be followed by NNEP are also restricted by the bound on NNEP's memory. The exact nature of this restriction depends on the strategy the parser uses to decide what nodes to forget and when to forget them. Regardless of the strategy, in any description in a derivation tree, there can be only a bounded number of nonterminals which are involved in equations subsequently in the derivation. This bound is the bound on the number of entities which the memory can store information about. Any particular memory conservation strategy will restrict derivations more than this, but no such strategy is presented in this document, as this issue is orthogonal to other concerns. For the data addressed in chapter 6, it is sufficient to forget nodes which have no chance of being equated in the future.

For the current version of the parsing model it is assumed that it will only be parsing sentences. To express the parser's expectation for a sentence, it initializes its parser state with a description containing only an IP (inflection phrase) node which expects a head. Thus, for this version, all derivations must start with a description containing an IP node with an underspecified *head* terminal, and that node must be the root of the final description. Eliminating this restriction would allow NNEP to handle input other than sentences.[4]

4.3 NNEP's Operations

NNEP's operations calculate steps in SUG derivations. Some of these operations combine the current description with a description from the grammar, some simply state equations between pairs of nodes already in the current description, and one removes nodes from the parser state so as to conserve memory. The sequence of parser states, grammar entries, and operations in a successful parse specify a complete SUG derivation.

The set of operations which can be used to calculate derivation steps is limited by the constraint that these operations be implemented with rules which only have one variable in both their antecedent and consequent. Three operations can be implemented simply within this constraint. One equates the root of a grammar entry with a node in the parser state, one equates the root of a tree fragment in the parser state with a node in a grammar entry, and one simply adds a grammar entry to the parser state without doing any equations. A fourth operation can be implemented using the list of tree fragments which are stored in the parser state. A list, rather than a set, is needed to represent the ordering between the terminals of the tree fragments. Given this list, the root of the rightmost tree fragment can be uniquely identified, so it can be referred to as a constant in a rule which attaches the rightmost tree fragment to a node in the adjacent tree fragment. Unfortunately, this set of operations isn't adequate. In order to recover long distance dependencies, NNEP needs a mechanism which identifies a unique trace node for which relationships with other nodes must be calculated. This mechanism is a stack containing nodes which have no immediate parent. All

[4]However, the matrix root has special linguistic properties, so perhaps this requirement should really be incorporated into the specification of SUG grammars, rather than being eliminated from the parser for SUG. Such a modification would not change the power of the formalism.

trace nodes are on this stack, and sometimes tree fragment roots are also put on it. Only the top node on the stack can be used in the rules which calculate long distance dependencies. Two new operations can be implemented with this stack. One simply equates the node on top of the stack with another node in the parser state. It is used for equating trace nodes with posthead gap sites. The other operation simultaneously equates the unparented node on top of the stack with a node in a grammar entry, and equates the root of that grammar entry with another node in the parser state. This operation is used both for equating trace nodes with subject gaps, and for attaching subjects to their clauses while at the same time attaching the IP of the clause. The rules which calculate what nodes are eligible to be equated with a trace node also use this stack, as will be discussed in the next section. This set of six operations is adequate for parsing natural language sentences.[5]

The stack discussed in the last paragraph is called the public node stack, and the top node on that stack is called the public node. Because of the locality constraint on rules, a rule which is testing information about one node does not generally have access to information about any other node. In this sense the information about most nodes is private, and not available with the information about other nodes. The public node is distinct in that virtually all the information about that node is available along with the information about each other node, in the form predications about the tree as a whole (i.e. constant predications). In this sense the information about the public node is public. Other uniquely identifiable nodes have some of their information available in constant predications, but most of their information is still private. Representing information as public requires a distinct constant predicate for each uniquely identifiable node and each predicate whose value for that node is part of the public information. Given the large number of predicates which are needed to represent the information about a node, it is very costly to have all a node's information be public. The node on the top of the public node stack needs its information to be public because rules need to be able to test this node's equatability with other nodes based solely on the public information about it. No other nodes need to be fully public. Provided no other nodes are made fully public, and given the constraint that all operations be implemented with rules which only have one variable in their antecedent and consequent, the only other operations which are possible are ones which attach tree fragments other than the rightmost one. These don't appear to be necessary, so NNEP's six operations form a complete set under the imposed constraints.

4.3.1 Combination Operations

There are four operations NNEP can use to combine grammar entries with the parser state, shown in figure 4.1. These operations can be implemented within the locality constraint on rules because the rules are specific to grammar entries. There is a distinct rule for each way each grammar entry can participate in each combination operation.[6] The patterns of these rules look for nodes in the parser state where the rule's combination operation could combine the rule's grammar entry by

[5] In the implementation there is actually a seventh operation for dealing with appositives and parentheticals. I have not included this operation in the general discussion because appositives and parentheticals are generally not considered part of the constituent structure of the sentence. These constructions are handled in the implementation by building them as tree fragments in which all nodes are purported to have immediate parents. The seventh operation simply removes from the parser state, tree fragments which have no remaining needs and no unparented nodes. These tree fragments are always completed appositives or parentheticals.

[6] In the actual implementation some parts of these rules are shared across operations, but they can be thought of as completely distinct. Also, because these operations are compiled out with the grammar entries, and because no other combination operations are possible given the constraints and the available data structures, the operations can be thought of as derived schema, rather than primitive operations.

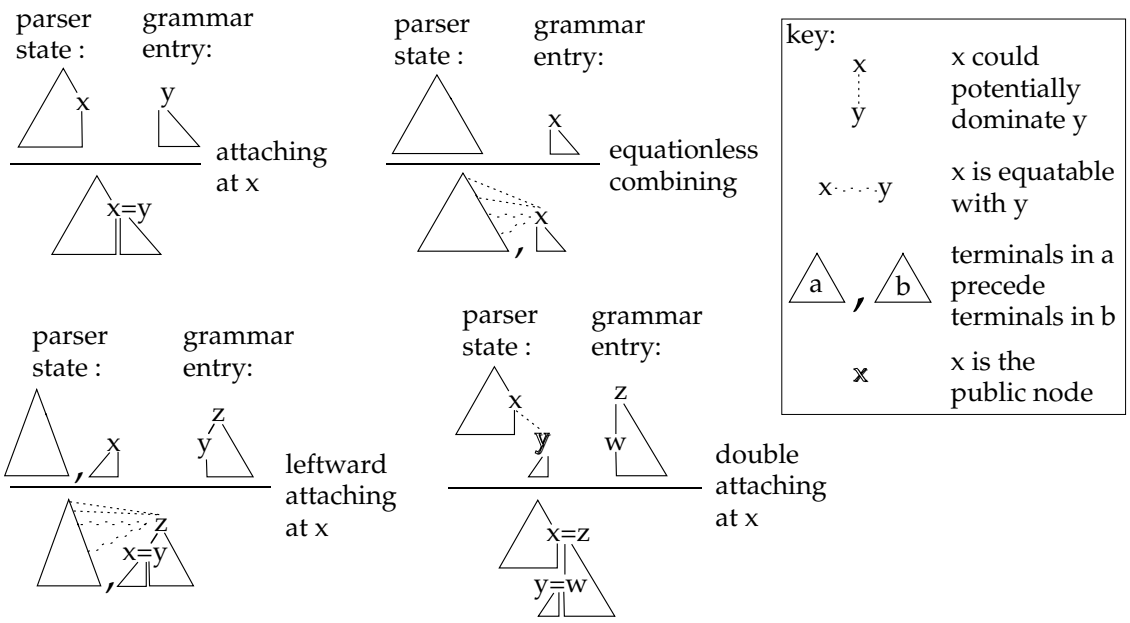


Figure 4.1: The operations which combine grammar entries with the parser state.

equating the rule's node(s). The actions of these rules add the information to the parser state which results from the chosen combination. By making rules specific to grammar entries, all the information about a rule's grammar entry can be compiled out into the rule, and thus all nodes in the grammar entry can be referred to with constants. Therefore the only variables which need to be used in these rules are the ones which range over nodes in the parser state. The only combination operation which involves more than one node in the parser state is double attaching. As was mentioned above and will be discussed further below, this rule can be implemented by restricting its application to only the cases where the unparented node is the public node. This restriction allows the implementation rules to access information about this node via constant predicates. Thus all these combination operations can be implemented within the constraints on the parser by compiling them out with the grammar into a set of massively parallel pattern-action rules. Given only this one public node, no other combination operations can be implemented within the locality constraint on rules.

The first operation shown in figure 4.1 is called attaching. It adds a grammar entry to the parser state and equates the root of the grammar entry with some node in the parser state. This operation is used to attach modifiers to the nodes they modify, and to attach posthead arguments to their subcategorized argument positions. If y doesn't have a head then this is modifier attachment, and if y is headed then this is an argument attachment. As with all equations, the resulting structure description must be satisfiable. Thus the feature structures of the two equated nodes must unify, and the x node must be on the right frontier of the rightmost tree fragment in the parser state. The x node must be on the right frontier because any words in the parser state's description must precede any words in the grammar entry, and thus x cannot precede words in the parser state. This combination operation also needs to be restricted so that it does not produce a structure which NNEP cannot complete. Particularly, nodes which have unfulfilled expectations should not be removed from the right frontier due to the combination. Thus x can not be preceded by any node which needs a parent or has unfulfilled expectations, and the grammar entry can not have any nodes which precede any words in the grammar entry and need parents or have unfulfilled

expectations. Also, unfilled terminals in the grammar entry can not precede any words, unless that terminal will have its word specified by the equation. This last restriction only applies to posthead modifiers. A posthead modifier has an unheaded root because it is a modifier, and the unfilled head terminal of the root precedes the word of the grammar entry because it is posthead. Thus this restriction simply requires that posthead modifiers can not attach to a node before that node has a head, as desired. Prehead modifiers are handled similarly. When a prehead modifier attaches to a node, the node must not have a head.[7]

Prehead arguments must be added to the parser state before they can be attached to anything. This is also true for grammar entries with several possible attachment sites which NNEP cannot immediately choose between. In both these situations the equationless combining operation is used. It adds the grammar entry to the end of the list of tree fragments, and stores in the parser state all the potential dominance relationships (which represent the possible prehead argument positions) and all the potential equations (which represent the possible attachment sites) with the root of the grammar entry. These relationships are added because calculating them within the parser state would be much more complicated than simply storing the values of pattern matches from the grammar entry's rules.[8] In order for the resulting parser state to be completable, the root of the grammar entry must have at least one potential dominance or equation relationships with a node in the adjacent tree fragment.[9] Since subjects are prehead arguments, a subject's first grammar entry is combined with the parser state using equationless combining. The IP node for the subject is specified as potentially dominating the subject node. Two examples of when an ambiguity between possible attachment sites might cause equationless combining to be used are prepositions and the first NP after a verb like "give". In the former case, the preposition may have several phrases which it could modify, and NNEP may need to wait for information about the preposition's object before it can make a decision. In the later case, the first NP after "give" could attach to either argument position, and information about subsequent constituents may be necessary to resolve this ambiguity. It is also possible that an ambiguity may exist between an attachment site and being a prehead argument. This is the case for the first word in most posthead NP arguments. That word could be starting the argument NP, or it could be starting an NP for the possessor of the argument NP. In all these cases, the equationless combining operation can be used to delay the decision until disambiguating information can be found. An operation to be discussed in the next subsection is used to commit to an attachment when there is enough evidence.

When a prehead argument phrase is added to the parser state using equationless combining, one way it can be attached to its argument position is with the leftward attaching operation. This operation is the symmetric case of attaching. It adds a grammar entry to the parser state and equates the root of the rightmost tree fragment in the parser state with a node in the grammar

[7]This mechanism isn't quite adequate for expressing prehead constraints, since the linear precedence constraint with the head terminal is not added to NNEP's representation of the current structure. In the few cases when prehead modifiers can have subsequent modifiers, the grammar can take advantage of the way precedence constraints are enforced by the parser. In these cases the modifier node can be specified as preceding the modified node. This is technically an inconsistent parser state, but since these constraints are only checked when operations are performed, it will have the effect of requiring any constituents which attach to the modifier to precede all subsequent terminals which are attached to the modified node. In the cases where this constraint is needed (namely wh- words), this has the right effect.

[8]This method also has the advantage that the relative values of the predications can be specific to the word, and not just dependent on the features assigned to the root node. These values are used for later disambiguation decisions.

[9]Checking for the existence of potentially dominating or equatable nodes does not violate the locality constraint on rules because the existence of such a node can be represented as a property of the structure as a whole, and testing for such nodes can be done independently for each candidate node.

entry. It also stores in the parser state all the potential dominance relationships and all the potential equations with the root of the grammar entry, for the same reason as this was done in equationless combining. Also as in equationless combining, the root of the grammar entry must have at least one potential dominance or equation relationships with a node in the adjacent tree fragment. Figure 4.1 shows more than one tree fragment in the parser state, because the x node can't be the root of the leftmost tree fragment. The root of the leftmost tree fragment is always the matrix root, and the matrix root must be the root of the final description produced by the parser. As for attaching, the feature structures of the two equated nodes must unify, and the y node must be on the left frontier of the grammar entry. To make sure the resulting structure can lead to a successful parse, the grammar entry can't have any nodes which precede a word and either need a parent or have unfulfilled expectations. Also, all unfilled terminals in the grammar entry which precede a word must be filled by the equation. Usually the y node precedes a word in the grammar entry. In this case, there can't be any nodes which need parents or have unfulfilled expectations in the rightmost tree fragment, unless they aren't preceded by any words.

The last combination operation shown in figure 4.1 is double attaching. This operation equates an unparented node in the parser state with a node in a grammar entry, and equates the root of the grammar entry with a node which could potentially dominate the unparented node. In order to implement this operation within the locality constraint on rules, the unparented node in the parser state (y) is restricted to be the public node. If the public node is the root of the rightmost tree fragment, then this operation attaches a prehead constituent of a phrase at the same time as it attaches the phrase. If the public node is a trace, then this operation equates the trace with its gap site at the same time as it attaches the phrase which contains the gap site. This operation is often used to attach subjects, regardless of whether the subject is a phonetically realized constituent or a trace. Because the public node is the top node on a stack, the restriction that y be the public node has some interesting linguistic implications, which will be discussed in section 6.2. The other constraints on the applicability of this operation are simply a combination of those on the attaching operation and the leftward attaching operation. Nodes x and z are constrained as are x and y in the attaching operation, and nodes y and w are constrained as are x and y in the leftward attaching operation. Constraints on the existence of other types of nodes in the parser state and the grammar entry map analogously from the attaching and leftward attaching operations to this operation.

4.3.2 State Internal Operations

There are two operations which NNEP can use to equate nodes already in the parser state, shown in figure 4.2. Since the nodes involved in these state internal operations are all in the parser state, one node in each operation needs to be referred to with a constant. For the internal trace equating operation this can be done in the same way as was done for double attaching. This operation is restricted to only apply when y is the public node. Thus the y node is always unique, and it can be referred to with a constant. For the internal attaching operation, the list of tree fragments needs to be used. This operations is restricted to only apply when y is the root of the rightmost tree fragment. Since this node is always unique, it can be referred to with a constant. Thus both of these state internal operations can be implemented within the constraints on the parser, using only one rule each. No other state internal operations can be implemented within the locality constraint on rules given just these two uniquely identifiable nodes. Other versions of internal attaching could be defined which apply to other roots in the list of tree fragments, but they don't seem to be necessary.

The first operation shown in figure 4.2, called internal attaching, is just like attaching, except the

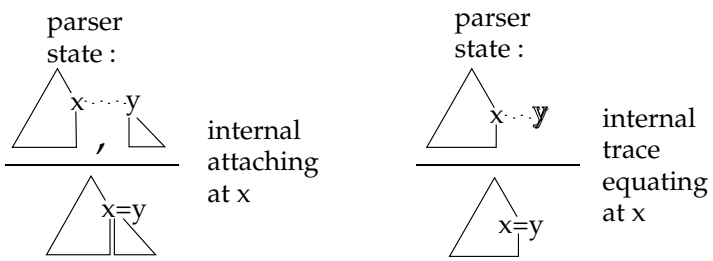


Figure 4.2: The operations which equate nodes already in the parser state.

attached subtree is a tree fragment in the parser state rather than a grammar entry. The internal attaching operation is used to commit to attachments which have been delayed using equationless combining, and to attach phrases which have had a prehead constituent attached to them using leftward attaching. The requirement that the resulting description be satisfiable and completable place the same restrictions on this operation as for the attaching operation. This operation requires that there be an *equatable* relation specified between the two equated nodes, but the mechanisms which calculate these relationships for this case are general enough that this requirement does not limit the applicability of this operation. Because the equatability of the two equated nodes can be checked simply by checking the *equatable* relation, not all the information about the rightmost tree fragment root needs to be represented in constant predicates. The requirement that the y node be the root of the rightmost tree fragment does limit this operations applicability, since otherwise it could be used to combine any two adjacent tree fragments. This does not have any significant implications for the ability to construct descriptions, but it does limit the strategies NNEP can use to delay and commit to disambiguation decisions. This limitation does not seem to be significant, but this question has not been adequately investigated.

The internal trace equating operation equates the public node with a node with which it is specified as equatable. The public node can't dominate any phonetically realized terminals, since then it would be the root of a tree fragment, and it would be attached using the internal attaching operation. The internal trace equating operation is used to equate trace nodes with posthead gap sites. The only constraints on this operation are that the trace node be the public node and that the equatable relationship be stated. Unlike for internal attaching, this later requirement does have implications. As will be discussed in section 4.4.2, the rules which calculate possible long distance dependencies are not completely general. Thus in some cases, not all the nodes which the SUG grammar allows to be equated with a given trace node will be specified as *equatable* with it by NNEP. This prevents the internal trace equating operation from doing some equations which the grammar says are possible. Along with the constraint that this operation only apply with the top node on the public node stack, the inability to calculate all possible equations for a trace node has interesting linguistic implications, which will be discussed in section 6.2.

4.3.3 The Forgetting Operation

As discussed in chapter 2, the memory of a computing module in the Shastri and Ajjanagadde architecture can only store information about at most ten variables at a time. In the implementation of NNEP, variables refer to nonterminal nodes in the current SUG description. Thus the number of nonterminals represented in the parser state needs to be kept below ten. As discussed in section 3.4, SUG descriptions are well suited for satisfying this requirement, because their partiality allows the parser to abstract away from unneeded information. The abstraction operation which NNEP uses

is called forgetting. The forgetting operation removes nonterminal nodes from the parser state, thereby abstracting away from the existence of that node in the SUG description being built. Since NNEP's output is incremental, forgetting does not interfere with the ability of other modules to interpret the results of a parse. Forgetting a node does prevent NNEP from equating it with any other nodes during the remainder of the parse, so only nodes which don't need to be equated in order for the parse to produce a complete description can be forgotten. Thus there are circumstances where even with the forgetting operation, NNEP cannot stay within the bounds on its memory. The linguistic implications of this fact will be discussed in section 6.4.

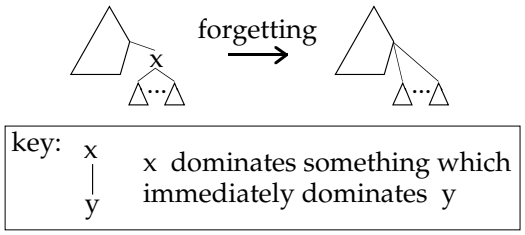


Figure 4.3: The operation which abstracts away from the existence of a node in the parser state.

The forgetting operation is shown in figure 4.3. It simply removes a nonterminal node and all the information about that node from the parser state. No information needs to be added when a node is forgotten. Because immediate dominance is represented with dominance information (if it is needed) plus a predicate representing the existence of an immediate parent, after forgetting a node the parser state is as if the children of the forgotten node were immediately dominated by the parent of the forgotten node, as depicted in figure 4.3. The only constraint on the applicability of this operation is that the node must not have any unsatisfied requirements. Thus the node must have an immediate parent, and can't have any unfulfilled expectations. If such a node were forgotten, then it could never be equated with a node which would satisfy its requirements, and thus the parser could never produce a complete description. Any nodes which are locally complete can be forgotten, but doing so may prevent the parser from doing an equation which would satisfy a requirement of a different node. For example, a modifier might be encountered later in the sentence which needs to be attached to the forgotten node. In some cases NNEP can determine from the structure that a node can not be involved in any more equations, so the node can be forgotten without losing any possible parses. However, in other cases a shortage of memory space forces nodes to be forgotten which could otherwise be involved in the remainder of the parse. The forgetting operation provides a mechanism for removing nodes from the parser state, but it is not a theory of when nodes can be removed. This question is one of disambiguation, and was discussed in section 4.1.3.

4.4 NNEP's State Information

NNEP's parser state stores information about the intermediate partial descriptions in SUG derivations. Since NNEP's output is the derivation structure of the parse, and not the resulting structure description, the only information which needs to be stored is that which the operations presented in the last section need to test. This information is represented in a way which makes testing it and setting it as fast as possible for the operations, given the constraints imposed by the architecture. The entities in this representation are nonterminal nodes. Binary predicates are used to represent potential dominance relationships and potential equations. All other information which NNEP

needs to store is represented with unary predicates. Predications are added to the parser state by operation actions and by a set of rules that calculate the indirect implications of information added by actions. The first subsection below presents the information stored in the parser state and its representation. The second subsection discusses what rules are necessary to calculate indirect implications of added information.

4.4.1 The Representation of State Information

NNEP's representation of its state information is designed to provide the information NNEP's operations need without violating the constraints on the parser's memory. The parser's memory can only hold a small number of entities, but can store an arbitrary (but bounded) number of predications over those entities. Thus the representation of NNEP's state information should use as few entities as possible. The entities in SUG descriptions are the feature structures, but luckily not all feature structures need to be represented as entities. For example, the feature structure $x \approx$*[cat:[V:+, A:−]]* has four feature structures in it, x, *cat(x)*, the +, and the −. However only x needs to be represented as an entity. The rest of the information can be represented with the predications *cat-V+(x)* and *cat-A−(x)*, assuming suitable interpretations of these predicates. Such techniques can be used to eliminate the need for feature structures other than the nodes themselves. In addition, the need to represent terminals as entities can be eliminated with the addition of the predicates *anchored* and *nonfrontier*. A node is *anchored* if it dominates a terminal with its word specified. A node is *nonfrontier* if it precedes a terminal with its word specified. The use of terminals in feature structures can be represented with special purpose predicates. For example, a node with a *head* feature specifying an underspecified terminal would have *needs_head* predicated of it, and if the *head* terminal was a word it would have *headed* predicated of it. Such information is all that NNEP needs to know about terminals in the structure description, and thus the terminals themselves do not have to be stored. Thus feature structures other than nonterminals do not have to be represented as entities.

In NNEP all nonterminals are represented as entities. This allows parser operations which are independent of the structural position at which they apply. This means that generalizations in the grammar are stated in terms of the same abstract entities as generalizations in the parser. However, this is not a claim about the relationship between the study of competence linguistics and the study of parsing. The grammars being discussed here are the grammars used by the parser, and may be quite different from the grammars which are appropriate for the study of competence linguistics. For example, the grammar used to test the adequacy of NNEP represent as a single nonterminal what in Government Binding Theory would be an entire lexical projection and all its associated functional projections. All the information about these projections is compiled into the feature structure of the node. The question of what the appropriate abstract entities are for parsing is dependent on several factors, including the generalizations which can be stated in terms of those entities, the ability to parse without violating the bound on the number of entities which can be stored, and the restriction's on the architecture's ability to implement rules involving multiple entities. In any case, this issue can be investigated in terms of the grammars used by the parser, and does not need to directly involve the design of the parser.

The structural information in SUG descriptions is represented with two binary predicates and a set of unary predicates. The binary predicates are potential dominance (*could_dom*) and potential equality (*equatable*). Potential dominance keeps track of what nodes may dominate what other nodes through a chain of immediate dominance links which have yet to be introduced. This relationship was used above in the double attaching operation. Potential equality keeps track of what

nodes may be equated with what other nodes through a sequence of equations which have yet to be done. This relationship was used above in the internal attaching and internal trace equating operations. The information which is needed about dominance relationships is represented in three ways: by not specifying a potential dominance relationship, with the representation of tree fragments, and with predicates that specify what nodes were added by the last grammar entry.[10] The root of a tree fragment dominates all the nodes in the tree fragment, and the root of the last grammar entry dominates all the nodes added by that grammar entry. The information about immediate dominance relationships which NNEP needs is represented with the *parented* unary predicate. A node is *parented* if some other node immediately dominates it. NNEP does not need to know which node is the immediate parent of a node, only that it has one. This alternative representation was introduced in the last chapter in order to allow for the forgetting operation. The nodes which dominate words are distinguished with the predicate *anchored*. This predicate is necessary in order to calculate which nodes are on the right frontier of the structure and thus are eligible for equations. If a node precedes an *anchored* node, then it precedes a word in the structure and thus is not on the right frontier. Such nodes are specified with the predicate *nonfrontier*. Ordering constraints between nonterminals and words are specified with the *nonfrontier* predicate, with a predicate such as *head_leftward*, or by limiting how the grammar entry can be combined with the parser state. Ordering constraints between nonterminals and unfilled terminals can also limit how the grammar entry can be combined with the parser state, and can also be expressed with a predicate that expresses the direction in which the terminal is needed, such as *needs_head_rightward*. Because of the compact representation of phrase structure in NNEP's current grammars, the vast majority of ordering constraints involve terminals. The only exceptions are the ordering constraints between objects of ditransitive verbs. As will be discussed in section 6.4, performance constraints mean that NNEP only needs to store such ordering constraints for one second object at a time. Because of this, such ordering constraints can be stored with two unary predicates, *preceded* and *preceding*, rather than a binary predicate.

The only other structural information which NNEP needs to represent in its memory is ordering constraints between terminals. Grammar entries don't have these constraints, but they do come as part of the input. The input words come ordered, and when the parser state includes more than one unconnected tree fragment, these constraints have to be stored, not just checked for consistency with other constraints. To represent this ordering, NNEP has three predicates, *matrix_fragment*, *adjacent_fragment*, and *right_fragment*. The matrix tree fragment is the one rooted by the matrix root of the sentence, the right tree fragment is the rightmost tree fragment, and the adjacent tree fragment is the one just to the left of the rightmost tree fragment. This set of predicates allows for four tree fragments to be stored at one time, where the fourth consists of all those nodes which are not in any of the other three. As will be discussed in section 6.4, performance constraints make this length sufficient. Any subtree which contains phonetically realized material and is rooted by a node which has no immediate parent is in this list. Nodes which represent traces for long distance dependencies are never roots of these tree fragments, because these trace nodes never dominate any phonetically realized material.

[10] NNEP does not always enforce the constraint that links in a phrase structure tree should not cross. In order to implement this constraint, all nodes would need to know what other nodes they dominate. The no-crossing constraint is usually assumed for English, although it is not language universal, and even in English it can be violated without much trouble. For example, "I saw a man yesterday who was wearing stockings". The aspects of this constraint which are enforced by NNEP can be thought of in terms of an attention shifting mechanism, which limits attention to the rightmost tree fragment whenever that tree fragment has unsatisfied requirements. This mechanism is discussed in section 4.5. Heaviness considerations are probably also involved in the predominance of noncrossing structures in English, but these have not been modeled here.

The only nonstructural information in SUG descriptions is contained in the feature structures. As was illustrated above, feature structure information is represented with predicates which are specific to the feature structures used in the grammar. For a simple feature structure with no coreference of values either within itself or between itself and other nodes, the information can be specified using a unary predicate for each feature label path to an atom. For example, *[cat:[V:+,A:−], tense:3sg]* can be represented with three predicates, which might be called *cat-V+*, *cat-A−*, and *tense-3sg*. Because of the constraints on allowable grammars, the set of such predicates which NNEP needs can always be determined from the grammar, and this set will always be finite. NNEP checks these predicates to determine the unifiability of feature structures. Coreference between nonterminal feature structures can be represented with a binary predicate for each possible equality between distinct feature label paths. For example if we need to represent *head(x)≈head(y)*, then we need a predication like *same_head(x,y)*. Because of the transitivity of equality it may be necessary to define more such predicates than are needed to specify the coreferences in the grammar, but because of constraints on the grammar the set of necessary predicates will always be finite and determinable from the grammar. For the grammars used in chapter 6, no such binary predicates are needed. Features which refer to terminals can be represented with two kinds of unary predicates, one kind if the terminal has its word specified and one kind if the terminal is unfilled. Predications of the later kind express the need for a predication of the former kind. Predications of the former kind prevent any equations where both nodes have the same such predicate predicated of them. As in the example mentioned above, a node with an underspecified *head* terminal could be specified as *needs_head* and a node with a word as its *head* could be specified as *headed*. When the parse is done all nodes with *needs_head* predicated of them must also have *headed* predicated of them. During the parse two nodes which are both *headed* can not be equated[11]. The set of such predicates which are needed to parse can be determined from the grammar.

As will be discussed in section 6.1, most of the linguistic constraints on long distance dependencies can be expressed as a system of coreferential feature values in SUG grammar entries. However, rather than use the techniques just discussed, it is easier for NNEP to represent these constraints with two predicates, *extractable* and *foot*. The rules which calculate long distance dependencies only look for possible gap sites at nodes which are specified as *extractable*. They only check to see if a node potentially dominates a trace if that node is specified as the *foot* of its grammar entry. The name of this predicate is taken from Tree Adjoining Grammar, as will be explained in chapter 6.

In addition to the above information about the SUG description being accumulated, NNEP also needs to store the state of the public node stack. The information which NNEP's operations need about this stack is what node is on top of the stack. This is represented with the *public* predicate. As will be discussed in section 6.4, the *equatable* and *could_dom* predicates mentioned above are actually implemented in two versions, one for nodes on the public node stack and one for other tree fragment roots. The first arguments of the first version of these predicates can be used to determine what nodes are on this stack. Using only this source of information and the *public* predicate, this stack can only be two deep. As will also be discussed in section 6.4, performance constraints and a strategy for conserving space on this stack make this depth sufficient. All trace nodes are on this stack, and sometimes the root of a tree fragment is on this stack.

[11] Two nodes which are both *headed* can be equated if they have the same *head* terminal. Thus a grammar might also need a predicate like *same_head* to distinguish this case. This isn't an issue in the grammars used here.

4.4.2 Maintaining State Information

NNEP's operations test and predicate the information just discussed, but sometimes the information added by an operation has implications for NNEP's current description which were not specified in the definition of the operations. If this implied information is not explicitly represented, then it would not be immediately available for later operations to test, and it might be lost altogether if a node is forgotten. For this reason, NNEP has a set of rules which calculate information indirectly implied by the effects of operations. These rules calculate the inheritance of ordering constraints, the positions of tree fragments on the tree fragment list, possible long distance dependencies, and the effect of equations on *equatable* relationships.

Linear precedence relationships inherit downward across dominance relationships. Linear precedence is defined so that if x precedes y, then everything which x dominates precedes everything that y dominates. This inheritance applies whenever a grammar entry or tree fragment attaches to a node with an ordering constraint on it. Because of constraints on the grammars, the grammar entry or tree fragment will always dominate a terminal. Thus if the attachment takes place at a node which is preceded by other nodes, then the inheritance results in the preceding nodes no longer being on the right frontier (*nonfrontier*). Since nonfrontier nodes cannot be involved in further equations, no other precedence information needs to be calculated in this case. If the attachment takes place at a node which precedes another nonterminal node, then all the nodes in the attached subtree also precede that node. Since NNEP is constrained to only have one linear precedence relationship, this amounts to inheriting the *preceding* predicate to either the nodes in the tree fragment or the nodes which have been introduced by the grammar entry. The only other case is when the attachment takes place at a node which precedes a word. Since such a node is not on the right frontier, this can only happen if the node is in a grammar entry and the attachment takes place with either the leftward attaching or double attaching operations. For double attaching, no inheritance needs to take place if the attached constituent is a trace node. Otherwise the attached constituent is the rightmost tree fragment, the *nonfrontier* predicate needs to be inherited to all its nodes. The inheritance of ordering constraints can therefore be calculated with the following rules. The variables in each of these and the subsequent rules are implicitly universally quantified.

$$preceded(next_comb_upper_site) \wedge preceding(x) \Rightarrow nonfrontier(x)$$

$$preceded(next_inter_attach_site) \wedge preceding(x) \Rightarrow nonfrontier(x)$$

$$preceding(next_comb_upper_site) \wedge new_node(x) \Rightarrow preceding(x)$$

$$preceding(next_comb_upper_site) \wedge anchored(next_comb_lower_site) \wedge right_fragment(x) \Rightarrow preceding(x)$$

$$preceding(next_inter_attach_site) \wedge right_fragment(x) \Rightarrow preceding(x)$$

$$nonfrontier(next_comb_lower_grammar_node) \wedge anchored(next_comb_lower_site) \wedge right_fragment(x) \Rightarrow nonfrontier(x)$$

Maintaining the tree fragment list means updating the predicates *matrix_fragment*, *adjacent_fragment*, and *right_fragment*. All new nodes added by combination operations are in the rightmost tree fragment, so the actions of the operations specify these nodes as *right_fragment*. When the equationless combining operation is used, a tree fragment needs to be added to the end of the list. This means that nodes which were *adjacent_fragment* are no longer *adjacent_fragment*, and nodes which were

right_fragment are now *adjacent_fragment* but not *right_fragment*. The rightmost and adjacent tree fragments need to be combined when either the internal attaching operation is used, or when the double attaching operation attaches the rightmost tree fragment. In these cases, nodes which were *adjacent_fragment* need to be changed to *right_fragment*, nodes which are *right_fragment* may need to become *matrix_fragment*, and which tree fragment is the adjacent one needs to be figured out. These last two issue depend on how full the list was before the operation, so their rules are sensitive to this information. The following rules are sufficient for maintaining the tree fragment list.

$$equationless_combining \wedge adjacent_fragment(x) \Rightarrow \neg adjacent_fragment(x)$$

$$equationless_combining \wedge right_fragment(x) \Rightarrow \neg right_fragment(x) \wedge adjacent_fragment(x)$$

$$combine_fragments \wedge adjacent_fragment(x) \Rightarrow \neg adjacent_fragment(x) \wedge right_fragment(x)$$

$$matrix_fragment(x) \wedge adjacent_fragment(x) \Rightarrow halffull_list$$

$$combine_fragments \wedge halffull_list \wedge right_fragment(x) \Rightarrow matrix_fragment(x)$$

$$\neg right_fragment(x) \wedge \neg adjacent_fragment(x) \wedge \neg matrix_fragment(x) \Rightarrow full_list$$

$$combine_fragments \wedge \neg adjacent_fragment(x) \wedge \neg right_fragment(x) \wedge \neg (full_list \wedge matrix(x)) \Rightarrow adjacent_fragment(x)$$

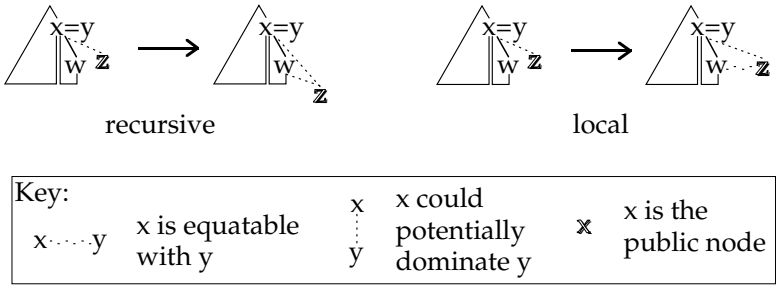


Figure 4.4: The updating rules which help calculate long distance dependencies.

In addition to the double attaching and internal trace equating operations, two updating rules are needed to calculate long distance dependencies. As illustrated in figure 4.4, one of these rules calculates new potential dominance relationships for a trace, and the other calculates new potential equation sites for a trace. The first rule is used to iteratively calculate the recursive component of a long distance dependency. As the right frontier of the tree is built, the trace node gets "passed" from root to leaf of each attached subtree. The second rule calculates the *equatable* relationships which are used by the internal trace equating operation to fill object gaps. Both these rules apply when the root of a tree fragment or grammar entry is equated with a node which could potentially dominate a trace node. In order to implement these rules using only one variable, the trace node must be restricted to be the public node. With this restriction, the x's and z's are all unique, so the only variables are the w's. Both these rules are further constrained in accordance with linguistic constraints and the appropriateness of the relation. The first rule requires that w be a *foot* node. This predicate specifies the linguistic constraints on what nodes can be extracted out of. The second rule requires that w be *extractable*. This predicate specifies the linguistic constraints on what nodes can be the sites of extractions. Both these predicates will be discussed in depth in chapter 6. In addition, the second rule needs to test whether the feature structure labels of w and z can unify.

Because z is the public node, enough of the information about z is available in constant predicates for this test to be done. The restriction that in both these rules z must be the top node on the public node stack imposes a number of important constraints on long distance dependencies, some of which cannot be specified within SUG (or TAG). These will be discussed in chapter 6. These two movement rules, then, can be implemented with the following rules.

$could_dom(next_comb_upper_site, public_node) \wedge \neg anchored(public_node) \wedge$
$\quad new_node(w) \wedge foot(w)$
$\quad \Rightarrow could_dom(w, public_node)$

$could_dom(next_inter_attach_site, public_node) \wedge \neg anchored(public_node) \wedge$
$\quad right_fragment(w) \wedge foot(w)$
$\quad \Rightarrow could_dom(w, public_node)$

$could_dom(next_comb_upper_site, public_node) \wedge \neg anchored(public_node) \wedge$
$\quad new_node(w) \wedge extractable(w) \wedge unify(w, public_node)$
$\quad \Rightarrow equatable(w, public_node)$

$could_dom(next_inter_attach_site, public_node) \wedge \neg anchored(public_node) \wedge$
$\quad right_fragment(w) \wedge extractable(w) \wedge unify(w, public_node)$
$\quad \Rightarrow equatable(w, public_node)$

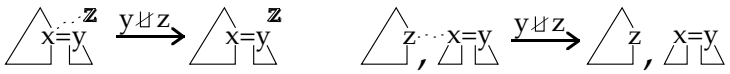


Figure 4.5: The rules which keep *equatable* relationships up to date after equations.

If two nodes are equated, any *equatable* relationships with those nodes need to be updated. As shown in figure 4.5, there are two cases when this applies, one for trace nodes and one for tree fragments. Since nothing is ever attached to a trace node, the only case when its *equatable* relationships need to be updated is when there is an equation with one of its possible equation sites. If the trace node is the public node, then this updating can be done by simply recalculating the equatability of the equated node with the public node after the equation. Because, as discussed in chapter 6, NNEP never puts a tree fragment on the public node stack if there is already a node there, the only time a trace node will not be the public node is when there is more than one trace node. In these circumstances, the equation must occur within the domain of the more recent trace, and because of this is extremely constrained. As discussed in section 6.2, such an equation site is either not a possible extraction site for the earlier trace, or the last equation at that site is done with double attaching, which also equates the more recent trace. In the later case the earlier trace becomes the public node as part of the effects of the operation, at which point the equatability of the equation site with the trace is recalculated. For the case of tree fragment roots, their *equatable* relationships need to be updated either when something is attached to them, or when trace equating is done in a lower tree fragment. In the later case, the equation itself combines the equatability of the two equated nodes to produces the equatability of the resulting node with the tree fragment root. The only other case is when something is attached to a tree fragment root, because no operations (other than trace equating) allow equations to be done to the left of the rightmost tree fragment without also attaching the rightmost tree fragment somewhere. If a grammar entry is attached to a tree fragment root, then pattern matches from the combination arbitrator are used to update the *equatable* relationship with the tree fragment root. If the rightmost tree fragment root is attached

to the adjacent one, then the *equatable* relationships with these two roots are combined to produce those for the resulting node. In these later two rules, the node which is not being equated is the single variable.

$$redo_equatable_public(x) \wedge equatable(public_node,x) \wedge \neg unify(public_node,x) \Rightarrow \neg equatable(public_node,x)$$

$$equatable(next_comb_upper_site,x) \wedge \neg unify(grammar_entry_root,x) \Rightarrow \neg equatable(next_comb_upper_site,x)$$

4.5 NNEP's Disambiguation Mechanism

The parser operations are distinguished from the rules for maintaining the parser state by the fact that the parser must choose whether and how to apply them, rather than having their application follow directly from the state of the parser. After each step in the parse, NNEP must determine how operations can be applied and choose among these possibilities. The current parsing model can calculate how operations can be applied, and it is designed to be able to choose among these possibilities on the basis of the available evidence. However, the later component of this process has not been tested, so no claims about the adequacy of this particular disambiguation mechanism can be made. As a matter of completeness, this section discusses the disambiguation mechanism which NNEP is currently designed to use. Doubtless there are other possible mechanisms, but determining the best one would require a substantial investigation of connectionist learning and evidential reasoning methods, which is outside the scope of the current investigation. The success of connectionist networks in other disambiguation problems indicates that this issue is not of particular concern for determining the computational adequacy of the S&A architecture. Indeed, one of the most promising aspects of the work done in this dissertation is that it makes it possible to apply connectionist methods to syntactic disambiguation at the level of syntactic representation.

4.5.1 Coordinating Disambiguation Decisions

NNEP has three types of decisions it needs to make, when and where to forget nodes, when and where to do an internal operation, and how to combine a grammar entry for the next word into the parser state. Forgetting generally does not interact with the other decisions, since it only involves nodes with little or no chance of being equated. If there is a conflict between forgetting a node and picking it for an operation, the forgetting operation is simply blocked from applying while the other decision is being made. The other operations interact in that only one of them can be performed at a time. Whenever one of the internal operations decides to apply, NNEP blocks the triggering of any further actions until that operation has finished modifying the parser state.[12] On the other hand, the combination arbitrator blocks the triggering of internal operations right after a word is input. Thus no internal operations can be triggered while NNEP is figuring out how a word could be combined, even if no actual decision has been made. This preference for the incorporation of input words over doing internal equations is motivated by the time pressures associated with the input of words.[13]

[12] If the internal operations match at the same time, and therefore on the same node, the internal attaching operation is done, since a trace node is more likely to find another possible equation site.

[13] The right mechanism shouldn't have this kind of discrete preference, but this strategy is easier to implement and control by hand. Some very recent work on a Bayesian approach to NNEP's disambiguation problem characterizes this

Another interaction occurs when an internal operation is necessary in order for a combination to take place. The space of possibilities that the combination arbitrator uses when making decisions includes both combinations which could be immediately applied and combinations which might be able to be applied after some sequence of internal operations. If it can't choose an operation which can be immediately applied, then the internal operation arbitrators are asked to find an operation to apply. This is done by removing from consideration the possibility that future input will provide an equation site for the nodes the internal operations are trying to attach. Thus the internal operations are asked to choose between the equation sites they currently have available. If this is successful, then control reverts back to the combination arbitrator, which tries again to pick an operation it can immediately apply. Currently there is no strategy for what to do if no internal operation can be picked, but progressively lowering the decision making threshold would be one plausible approach.

4.5.2 Choosing an Action

Of the three types of decisions, when and where to forget nodes is the simplest one. Currently, a node is forgotten when its chance of being equated is lower than a fixed threshold. A more sophisticated strategy would vary this threshold depending on the number of nodes stored in the parser state, but this is not necessary for the data considered in this document. The chance that a node will be equated is the chance that it will either be attached to another node or have another node attached to it. A node will be attached to another node if and only if it has no immediate parent. As will be discussed in the next section, the chance that a node will have no further nodes attach to it is stored as the value of the *finished* predicate. Thus the forgetting decision can be made by choosing those nodes which are *parented* and have a *finished* value which is greater than some threshold.

The internal operations each have their own arbitrators. These arbitrators make decisions using a summation network and a threshold unit. The summation network sums the probability estimates for all the ways the unparented node of concern could end up being attached. This includes the possibility that subsequent input will provide the site of this attachment. The threshold unit receives weighted inhibitory input from the summation network, and excitatory input from the estimates of the probabilities for the ways the attachment could currently be done. The weight of the summation input must be between 0.5 and 1. If the probability estimate for the current attachment site is greater than the weighted sum of the probability estimates for all the possible attachments, then that attachment action is triggered. This calculation is formalized and explained in the following two paragraphs.

For the internal trace equating operation, the test the arbitrator wants to perform is:

$$\theta \ < P(x \mid s)$$

where θ is the decision threshold, x is the site of the candidate equation, and s is the parser state. In order to guarantee that this test will only be passed by a single alternative, θ must be at least 0.5. The probability estimates which are used to perform this test are stored in the *equatable* and *could_dom* predicates. The *equatable* predicate stores estimates of $P(x \mid s)$ for each potential equation site x, and the *could_dom* predicate stores estimates of the probability that future input

choice in terms of the probability that resource limitations will cause the parse to fail, but this work is too preliminary to report here. Using this approach would allow a single arbitration mechanism to be used for all decisions, rather than having to coordinate multiple arbitrators, as is done here.

will introduce potential equation sites for the trace node through an attachment at the potentially dominating node. I'll write this later probability as $P(future_node_under(x) \mid s)$. The trace node has to eventually equate with some other node, and each of these estimates represent probabilities of mutually exclusive alternative equations, so these alternatives form a partition of the possible equations for the trace node. Thus they should sum to 1. However, a constant factor may be introduced in the representation or calculation of these estimates, so they will actually sum to that constant factor. Thus, in order to perform the above test, each estimate needs to be divided by the sum of all the estimates. This gives us the following test.

$$\theta \ < \frac{P(x \mid s)}{\sum_i (P(x_i \mid s) + P(future_node_under(x_i) \mid s))}$$

This test is equivalent to:

$$0 < P(x \mid s) - \theta \sum_i (P(x_i \mid s) + P(future_node_under(x_i) \mid s))$$

This is precisely the calculation described in the previous paragraph.

For the internal attaching operation, the same test needs to be performed, but the probability estimates which are involved are a little more complicated. Again, these probability estimates are stored in the *equatable* and *could_dom* predicates, but in this case the values of these predicates were set using probability estimates from the combination arbitrator's calculations. These probability estimates are of the form $P(x, \ldots \mid s)$ and $P(future_node_under(x), \ldots \mid s)$, where the ellipsis represents additional information which was available at the time that the tree fragment root was added to the parser state, as will be discussed below. Given that this additional information is available, the test the arbitrator wants to perform is:

$$\theta \ < P(x \mid s, \ldots)$$

Using Bayes' rule, we get the following relationship.

$$P(x \mid s, \ldots) = \frac{P(x, \ldots \mid s)}{P(\ldots \mid s)}$$

As will be argued below, the probability estimates in the *equatable* and *could_dom* values are for mutually exclusive alternatives which exhaust the possible attachments which are compatible with the information in "..." and s. Thus:

$$P(\ldots \mid s) = \sum_i (P(x_i, \ldots \mid s) + P(future_node_under(x_i), \ldots \mid s))$$

Combining this with the previous two formulas, we get the following test.

$$0 < P(x, \ldots \mid s) - \theta \sum_i (P(x_i, \ldots \mid s) + P(future_node_under(x_i), \ldots \mid s))$$

This is exactly the test for internal trace equating, only the estimates are based on more information. The threshold may be different than that for internal trace equating, but it still must be at least 0.5. Note that this calculation will also take care of any constant factor in the estimates.

The combination operation arbitrator differs from the others in that it needs to choose a grammar entry and operation as well as a site. This makes the space of alternatives more complicated than just the currently available sites plus any future alternatives. In particular, the set of phrase

structures which are compatible with two different choices may not be disjoint. In other words, the information which would be added by one choice may not be incompatible with the information which would be added by another choice. Thus NNEP must not only choose what information to add, but also how much information to add. For example, if the next word is a preposition, there may be more than one place where the prepositional phrase could attach in the current phrase structure description. NNEP can either specify one attachment site, the other attachment site, or just the grammar entry's information without specifying any attachment site (using equationless combining). The first two options constitute the basic choice to be made, but NNEP can pick the third option, which is more general than each of the first two. Currently NNEP will always pick the most specific option which it is sufficiently confident is correct. It estimates the probability of each option not adding any incorrect information, then picks the most specific option whose probability of being correct is greater than a threshold. As above, this threshold must be at least 0.5. The best value would have to be determined empirically, and may need to vary depending on time and resource pressures.

The space of ways that NNEP can combine a grammar entry with its current description (i.e. the space of combination choices) has some properties which make the calculation of the above probabilities easier. At a given point in a parse, there is a set of phrase structure trees which are compatible with the parser's current description. Each of the combination choices restricts that set further. If one choice is more general than another, then all the trees which are compatible with the more specific choice are compatible with the more general choice. I will call the choices which are not more general than any other possible choice the basic choices. The grammar can be designed so that any basic choice adds information to the description which is incompatible with the information added by each other basic choice. In other words, no tree is compatible with more than one basic choice. I will assume that NNEP's grammar is in the appropriate form to have this property.[14] The grammars used in this document are. These basic choices can be divided into two groups, those that are immediately applicable, and those which require internal operations to be performed before they might be applicable.[15] The basic choices which are immediately applicable play the same role as the currently available equation sites for the internal operations.

To completely characterize the space of alternatives available to the combination arbitrator, we also need to consider those equation sites for the root of a grammar entry which might be introduced by future grammar entries. For example, when the beginning of a noun phrase is encountered, that noun phrase may be the possessor for a subsequent genitive marker. There is no combination choice which would necessarily restrict the possible trees to only those compatible with this alternative, so it is not a basic choice, but this alternative still has to be considered. These alternatives play the same role as those whose probability estimates are stored in the *could_dom* predicate for the internal operations. To determine what these alternatives are, grammar entries which can participate in the double attaching operation need to be classified according to what types of nodes can participate in the upper and lower equations. The transitive closure of these pairs of node types determine what nodes in the current description could dominate (*could_dom*) the roots of what grammar entries via future grammar entries. The classes of alternatives which are found using these relationships and the next word's grammar entries are the future choices we need. These future choices are equivalent to the type raised grammar entries used in formalisms like Combinatory Categorial

[14] I believe any SUG grammar can be transformed into an equivalent grammar which has this property.

[15] There is a glitch here. Since the combination arbitrator can't predict what the internal arbitrators are going to do, it can't predict with certainty what additional choices will be available after the internal operation. This doesn't seem to be a problem. Perhaps a more uniform disambiguation mechanism (such as that mentioned in footnote 13) which doesn't split the decision making process like this would not have this limitation.

Grammar (Steedman, 1987). As with basic choices, I will assume that these future choices are defined so that they are compatible with mutually exclusive sets of phrase structure trees. This can always be done.

Together, the basic choices and the future choices characterize all the ways that a grammar entry for the next word can become part of the sentence's phrase structure tree. They are also mutually exclusive, since within each class they are assumed to be mutually exclusive, and the basic choices equate the root of a grammar with nodes which are already in the parser state, while the future choices equate it with a node which has not yet been introduced. Thus this set of choices partition the set of phrase structure trees which could be built given the current description and the next word. This characteristic will be important in the calculations of the combination arbitrator.

The basic and future choices characterize the issue of what information to add to the current description, but we still need to characterize the issue of how much information to add. NNEP can add less information than that for a basic choice by either using a grammar entry which is strictly more general than another grammar entry, or by using an operation which is strictly more general than another operation. An example of the former case would be if a grammar entry was split into a lexical and a nonlexical component, but the original grammar entry was still left in the grammar. For simplicity, and because it makes the parser faster, I will assume that no such grammar entries exist. In conjunction with the constraint that basic choices be mutually exclusive, this means that all grammar entries for a given word must impose mutually exclusive sets of constraints. This is the case for all the grammar entries used in this dissertation.[16] NNEP does, however, have operations which result in strictly more general parser states than would other operations. One is equationless combining, which results in a parser state which is more general than that produced by any use of the attaching operation. In cases where the double attaching operation can apply, the leftward attaching operation would produce a more general parser state. In addition to basic choices, these more general combinations may also be compatible with future choices. For example, using equationless combining to add a noun phrase after a transitive verb is not only compatible with attaching the noun phrase as the object of the verb, it is also compatible with attaching the noun phrase as the possessor for a subsequent genitive marker. Under the above assumptions about the form of the grammar, these more general combinations are the only nonbasic choices which the combination arbitrator has available to it. In addition, the only difference between the basic choices and the nonbasic choices is whether an equation with the root of the grammar entry is specified. This means that the set of phrase structure trees which are compatible with a nonbasic choice is simply the union of the sets of trees which are compatible with the basic and future choices that the nonbasic choice is more general than. Because of this, the probability of a nonbasic choice not adding any incorrect information is simply the sum of the probabilities for its basic and future choices.[17]

As mentioned above, NNEP's combination arbitrator chooses the most specific action whose probability of not being incorrect is greater than a threshold. Since there are only two degrees of specificity, this amounts to trying to pick a basic choice which can currently be applied, and if none is sufficiently probable, trying to pick a nonbasic choice. As will be discussed in the next subsection, these probabilities are of the form $P(g, o, x, w, t' \mid a, a', s)$, where g is the grammar entry, o is the combination operation, x is the site of the combination (if any), w is the next word,

[16] As with the previous constraints on the form of the grammar, I think that all SUG grammars can be transformed into equivalent grammars with this property.

[17] This sum should be adjusted for the probability that the additional resource usage would cause the parse to fail. Making such adjustments would allow basic and more general choices to be compared directly, rather than only choosing the more general option when no basic option can be chosen.

t' is the category (tag) of the subsequent word, a and a' are the input information about the next and subsequent words (acoustics), and s is the parser state. Given a particular threshold θ, the test we want to perform is:

$$\theta \ < P(g, o, x, w, t' \mid a, a', s)$$

This test is done in an analogous way to the internal arbitrators' calculations. Bayes' rule gives us:

$$P(g, o, x, w, t' \mid a, a', s) = \frac{P(g, o, x, w, t', a, a' \mid s)}{P(a, a' \mid s)}$$

Since, as argued above, the basic and future choices partition the set of trees which are compatible with the parser state and input information, $P(a, a' \mid s)$ can be calculated by summing over the specific cases for the basic and future choices. For notation, I will use c to represent a basic or future choice type, in that g, c, x represents a specific choice. Although these choices include the basic choices represented with g, o, x, g, o, x represents an immediately applicable combination, whereas g, c, x represents either a combination which may require internal operations to apply before it can take place, or a class of combinations which require further input in order to take place. Using the sum over the specific basic and future choice cases to calculate $P(a, a' \mid s)$ in the above formula, we get:

$$P(g, o, x, w, t' \mid a, a', s) = \frac{P(g, o, x, w, t', a, a' \mid s)}{\sum_{i,j,k,l,m} P(g_i, c_j, x_k, w_l, t'_m, a, a' \mid s)}$$

This gives us the following test.

$$0 < P(g, o, x, w, t', a, a' \mid s) - \theta \sum_{i,j,k,l,m} P(g_i, c_j, x_k, w_l, t'_m, a, a' \mid s)$$

From this discussion we see that the combination arbitrator makes its decision by summing all the probability estimates for basic and future choices, then trying to find an immediately applicable basic choice which passes the above test, and if this fails trying to find a nonbasic choice which passes the test. As with the internal operations, this test can be performed with a summation network and a threshold unit. The summation network sums the probability estimates for all basic and future choices. The threshold unit gets input from the probability estimate for an immediately applicable combination, and inhibitory input from the summation network, weighted by θ. Unlike with the internal operations, there is a separate threshold unit for each grammar entry, and in some cases multiple threshold units per grammar entry to test the different operations that can be used. In addition, for each immediately applicable nonbasic choice, there is another summation network which sums basic and future choice estimates to calculate the nonbasic choice's probability estimate.[18]

4.5.3 Estimating Choice Probabilities

To perform the above tests, we need to be able to estimate each choice's probability of being correct. The evidence which NNEP is designed to use to estimate these probabilities comes from three sources, the input words, the parser state, and other modules in the language system. While the nature of the evidence from other modules is outside the scope of this investigation, it is important

[18] These nonbasic choices are all the same as basic choices where the root of the grammar entry does not get equated. Thus they share the same threshold units as the basic choices, but the value of the summed estimate is input so that the upper equation is to a node which does not currently exist.

that NNEP's disambiguation mechanism be able to incorporate this evidence. For example, when deciding between two places to attach a preposition, the module which processes the discourse model may want one phrase to be further specified by this preposition's phrase, but not the other. The existence of this type of interaction was demonstrated by Altmann and Steedman (1988). They also argued that interaction between higher level modules and the syntactic parser must occur in parallel. This parsing model's disambiguation mechanism allows the preferences of other modules to influence its decisions, and these preferences can be incorporated in parallel. In the following discussion, these preferences would be treated in the same way as information in the parser state.

The two sources of evidence which are within the scope of this investigation are the input words and the parser state. The parser state information was discussed in section 4.4, and one more property of nonterminals will be added below. The information about input words which NNEP uses in a given decision is the next word in the sentence, plus the grammatical category[19] of the subsequent word. The information about the subsequent word is rarely needed, but there are circumstances where it has to be used. Chapter 6 argues that the information represented in the parser state and this information about the next two words is sufficient for making disambiguation decisions which do not involve higher level preferences. That discussion does not predict specific choices, but it does argue that the necessary information for making a choice is available when the choice has to be made.

As discussed above, NNEP's disambiguation mechanism requires that probabilities be estimated for each of the basic and future choices. These choices designate alternative ways that the next word could be incorporated into the parser state, and consist of a grammar entry g for the word, a choice type c, and a node x in the current phrase structure description. For the sake of completeness, I will assume that the input information about the words of the sentence is probabilistic, so the next word w and the part of speech of the subsequent word t' need to be determined on the basis of this probabilistic information a and a' ("a" for acoustics, "t" for tag). The information about the parser state s also includes probabilities. From the last subsection, the probability we are trying to estimate is $P(g, c, x, w, t', a, a' \mid s)$.

There are two problems in estimating $P(g, c, x, w, t', a, a' \mid s)$. This probability needs to be broken up into pieces which can individually be estimated on the basis of a reasonable amount of data, and the recombination of these pieces needs to be computable within the S&A architecture. The following discussion proposes a computation which can be done by the architecture, and I believe the individual components can be estimated on the basis of a reasonable amount of data. However, some independence assumptions are made, and I have no evidence (other than intuition) that they are valid. Hopefully the reader will find them plausible.

With two applications of Bayes' rule we get:

$$\begin{aligned} P(g, c, x, w, t', a, a' \mid s) &= P(a, a' \mid g, c, x, w, t', s) P(g, c, x, w, t' \mid s) \\ &= P(a, a' \mid g, c, x, w, t', s) P(g, c, w, t' \mid x, s) P(x \mid s) \end{aligned}$$

Since a and a' represent the input information about the next two words, and words are pronounced or spelled the same way regardless of their grammatical use (or they would not be the same word), their probability is only dependent on w and t'. Since the category t' reflects all the information about the subsequent word that is needed for determining compatibility with g, c, x, and s, g, c, x, and s do not provide any additional information about a', despite the fact that t' does not

[19] There are only a few categories which need to be distinguished. These are categories such as words which can start NP's, or finite verbs. The part of speech tag of a word would be sufficient information for this categorization.

specify the actual word.

$$P(a, a' \mid g, c, x, w, t', s) \approx \ P(a, a' \mid w, t')$$

This probability can be the input to the estimation process from the word recognition module.

The probability of combining at a given node, $P(x \mid s)$, is independent of the input. For the combinations where the variable (x) is the site where the root of the grammar entry is equated, $P(x \mid s)$ is the probability of ever attaching something at the node, times the probability that no other node will be attached to first (except with internal operations). The first component is represented with the *finished* predicate. The value of the *finished* predicate is the probability that no root nodes will be equated to the node. The second component is a combination of the requirements of other nodes, linear precedence relationships, and the dominance relationships represented by tree fragments. If another node needs a grammar entry attached to it and precedes x, then x can't have a grammar entry attached to it until the other node does. This is calculated before the arbitration process and is represented with a signal called *blocking*. Dominance relationships play a similar role, to the extent that English does not allow branches in phrase structure trees to cross. This amounts to limiting the attention of the parser to the lowest constituent which hasn't finished being built. While this constraint is violated in English, there are some cases where it does apply. The current parser manifests this constraint in that, whenever there is a node in the rightmost tree fragment which must have a grammar entry attached to it, only choices where x is in the rightmost tree fragment are considered. This is also precalculated, and is represented with the signal *nonattended*. For the leftward attaching operation, the variable is the node which is being attached, not the attachment site. Thus, instead of the *finished* predicate, the *parented* predicate is used. Linear precedence constraints come into play if the site of the attachment is not on the right frontier of the grammar entry. In this case, if there are any requirements in the rightmost tree fragment which aren't satisfied by the attachment, then the operation can't apply. This is also precalculated, and represented with the *unfull* predicate. For the dominance instantiating operation, this later information is part of the public information about the public node. From this discussion we see that $P(x \mid s)$ can be calculated by multiplying probability estimates which are either part of the parser state or are precalculated.[20]

The only remaining calculation is for $P(g, c, w, t' \mid x, s)$. It seems plausible that the only way that information about one node in the parser state might influence the chances of combining at another node is by allowing combinations which might otherwise be performed at the other node. Since in this probability the site of the combination is given, information which is not local to that site should be irrelevant. Another way of thinking of this assumption is from the point of view of the speaker. If a speaker knows they are going to add to the specification of a constituent, then what they are going to add would not be dependent on what other nodes are in the structure. It would, on the other hand, be dependent on the information about the constituent itself. Making this locality assumption, we have:

$$P(g, c, w, t' \mid x, s) \approx \ P(g, c, w, t' \mid ld(x, s))$$

where $ld(x, s)$ is the information in the local domain of x in s. The more information in $ld(x, s)$, the more accurate this locality assumption is.

At this point it is necessary to take into consideration the fact that the information in s may be probabilistic. Thus there may be many sets of categorical parser states which are compatible with

[20]These predicate and signal values all represent the multiplied estimate subtracted from 1. This is because this multiplication is done with inhibitory links, which multiply the primary link's value by one minus the inhibitory value.

$ld(x,s)$. Let the set of $ld_i(x,s)$ be the set of all categorical local domains which are compatible with $ld(x,s)$. Since the sets of trees compatible with each $g,c,w,t',ld_i(x,s)$ partition the trees compatible with $g,c,w,t',ld(x,s)$, we have:

$$\begin{aligned} P(g,c,w,t' \mid ld(x,s)) &= \sum_i P(g,c,w,t',ld_i(x,s) \mid ld(x,s)) \\ &= \sum_i P(g,c,w,t' \mid ld_i(x,s),ld(x,s))P(ld_i(x,s) \mid ld(x,s)) \end{aligned}$$

Since the probabilities in s mean nothing other than the chances that the categorical feature values are true,

$$P(g,c,w,t' \mid ld(x,s)) = \sum_i P(g,c,w,t' \mid ld_i(x,s))P(ld_i(x,s) \mid ld(x,s))$$

Those $ld_i(x,s)$ which are incompatible with g,c,w,t' won't matter because $P(g,c,w,t' \mid ld_i(x,s))$ will be 0.

The last assumption which needs to be made is that the probabilistic information in s is specific to individual features. This allows this information to be stored with independent continuous valued predicates, and it means the following assumption is true, where $ld_i(x,s) = f_{1i}(x) \wedge \cdots \wedge f_{ni}(x)$:

$$P(f_{1i}(x),\ldots,f_{ni}(x) \mid ld(x,s))) \approx \; P(f_{1i}(x) \mid ld(x,s)) \ldots P(f_{ni}(x) \mid ld(x,s))$$

With this assumption we now have:

$$\begin{aligned} &P(g,c,w,t' \mid ld(x,s)) \;\approx \\ &\sum_i P(g,c,w,t' \mid f_{1i}(x),\ldots,f_{ni}(x))P(f_{1i}(x) \mid ld(x,s)) \ldots P(f_{ni}(x) \mid ld(x,s)) \end{aligned}$$

Note that if $P(g,c,w,t')$ is independent of a feature, then that feature does not have to be considered in this calculation.

We've now broken down the estimation of $P(g,c,x,w,t',a,a' \mid s)$ into presumably estimable components. The resulting formula is:

$$\begin{aligned} &P(g,c,x,w,t',a,a' \mid s) \;\approx \\ &P(a,a' \mid w,t')P(x \mid s)\sum_i P(g,c,w,t' \mid f_{1i}(x),\ldots,f_{ni}(x)) \\ &\qquad\qquad P(f_{1i}(x) \mid ld(x,s)) \ldots P(f_{ni}(x) \mid ld(x,s)) \end{aligned}$$

This estimate can be calculated within the S&A architecture by calculating each case in the sum independently, and using links which inhibit other links to perform the multiplications. The estimates for each case go to the same units, thereby getting summed. A reasonable abstract model of one link inhibiting another is that given the primary link's weighted activation p, and the inhibitory link's weighted activation i, the resulting activation is $p(1-i)$. Thus each case of the sum can be calculated with a single primary link inhibited by $n+k$ inhibitory links. The input activation to the primary link is $P(a,a' \mid w,t')$, since this is presumably the output of the word recognition module. Because $P(g,c,w,t' \mid f_{1i}(x),\ldots,f_{ni}(x))$ does not depend on the actual parser state, it can be the weight of the primary link.[21] k of the inhibitory links input one minus each of the values which are multiplied together to calculate $P(x \mid s)$. The other n inhibitory links likewise do the remaining multiplications by inputting $(1 - P(f_{ji}(x) \mid ld(x,s)))$ from the parser state. Preferences from higher level language modules would be multiplied in using other inhibitory links.

[21] The "x" in this formula refers to a generic site where a combination takes place, and does not actually refer to the x in the parser state currently being considered.

4.6 NNEP's Output

Once NNEP has chosen an action to perform, other language modules need to know about it. This is necessary in order to comply with the requirement that the output of the parser be incremental, and therefore express as much information as possible as soon as possible. The output of this parsing model is simply the sequence of actions performed. This includes the operations used, the grammar entries used (if any), and the nodes where the operations were performed (if any). This output is the same as the derivation structure of the SUG derivation which the parser is computing. It conveys all the information about the sentence's phrase structure which is included in the parser state, since another module could simply follow the derivation itself to construct that parser state. This form of output is also maximally incremental, and does not require other modules to know anything about the internal representations of the parser. Other modules simply need to know the significance for them of the choice of a particular grammar entry, and the significance for them of the equation of two nodes in the syntactic constituent structure. Because the domain of locality of SUG grammar entries is sufficient to express predicate-argument relations over them, the significance of a grammar entry for the module that constructs predicate-argument structure is simply the predicate-argument structure associated with that grammar entry. The significance of the equations which the parser does may be less clear, since there may not be a deterministic mapping from nodes in the constituent structure to nodes in the predicate-argument structure. For example, the different thematic roles of the objects in "load the truck with hay" and "load the hay on the truck" would not be represented in the constituent structure, but would have to be distinguished in the predicate-argument structure. In order for the equations to mean anything to other modules, the interface between the modules needs to keep track of the relationship between nonterminal nodes and the entities in other modules. Such an interface has been proposed by Aaronson (1991). In addition to the sequence of actions performed, NNEP continuously signals whether the current parser state represents a complete description. A parse is only successful if it ends with this output being true.

Chapter 5

The Connectionist Implementation

The previous chapters have given a complete description of a model of syntactic parsing in the Shastri and Ajjanagadde connectionist architecture with virtually no mention of units or links. This has been possible because the S&A architecture provides a clear mapping from the level of units and links to the level of symbolic computation. Because of this relationship, the discussion of the parser could be done in terms of constrained symbolic computation. This allows the findings of other work on parsing and on natural language in general to be used in the investigation, and it prevents the aspects of the architecture which are irrelevant to syntactic parsing from complicating the investigation. However, it is still important to show how this abstract model can be implemented at the level of units and links. This demonstration is necessary in part to check that no errors have been made in the argument that the parsing model is implementable in the S&A architecture. A connectionist implementation is also necessary because there are characteristics of NNEP which are best characterized in terms of units and links. These characteristics are not emphasized in this document because the primary concern is demonstrating the adequacy of the architecture for doing traditional natural language parsing. Also, the specifics of the current connectionist implementation are not intended to be a model of the real biological computations which the brain uses to parse, so any predictions made by these specifics should not be interpreted as characteristics of the parsing model. However, it is worth discussing those characteristics which are likely to be true of any implementation of a parser like NNEP in the S&A architecture. Perhaps future research will be able to develop a model at this less abstract level which will be able to make predictions about the nature of parsing in the brain.

Chapter 2 discussed the basic characteristics of the S&A connectionist computational architecture. Sets of units are used to represent predicates, and firing phases are used to represent variables. Thus the firing of a unit in a phase represents a predication over a variable. Predications about the situation as a whole are represented with sets of units that produce constant output across all phases. The strength of an output pulse will be used here to represent the probability of a predication being true. Links are used to implement pattern-action rules. These links are weighted, so soft rules can be implemented, and links can inhibit other links, which allows activations to be gated or multiplied. The first section below gives a more specific specification of these primitives.

As touched on throughout chapter 4, these connectionist primitives are used to implement NNEP. The organization of this implementation is shown in figure 5.1. Variables in the memory refer to nonterminal nodes in the parser state, so whenever a new node is introduced it is assigned an unused phase in the pattern of activation. Specific sets of units are included in the implementation for each predicate which is needed to represent the information in the parser state. The collective

output of these sets of units is high in the phases in which their predicate is true. There are also units which transform stored information into a form which can be more easily used. This can be combining information within a phase, transferring information from one phase to other phases, or a combination of both. Because of the locality constraint on rules, transferring information between phases is always done with predications about the situation as a whole, so only value information can be transferred, not phase information. The grammar, operations, and rules of NNEP are implemented in pattern-action rules. The grammar and operations are implemented together in the pattern-action rules which calculate parser actions. These rules are split into links for finding nonterminals which match their patterns, and links for exciting the units for the information in their actions, with arbitrators in between. The arbitrators implement the disambiguation mechanism discussed in section 4.5. The patterns use output from both predicate storage units and from the units which transform stored information. Likewise, the actions set stored predications directly, and make use of units which transform signals from the actions into the form which is stored. The action of the forgetting operation is implemented with links which suppress all predications stored for the node to be forgotten, and does not require any information to be transferred to other phrases. The parser rules which keep the parser state up to date are compiled into the grammar-operation rule actions. Thus they are implemented as part of the set of units which transform signal from actions into the form which is stored. Because some predicates are implemented with fewer units than would be needed in the general case, there are also links which remove such predications when they are no longer needed.

The rest of this chapter starts with a discussion of the structure of the implementation, including some comparisons with other network structures. The second section then provides a short presentation of the primitives of the S&A connectionist architecture and the ways they are used here. For a detailed and extensive presentation of this architecture see (Shastri and Ajjanagadde, 1993). Then the implementation of NNEP's operations and grammar is discussed, followed by the implementation of the rules which keep the parser state up to date. These components are then brought together in a discussion of the time course of parsing. In the sixth section, some characteristics of this implementation of NNEP are discussed. In the last section the computer simulation of the implementation is briefly described, and an example of its operation is presented.

5.1 Network Structure

Most of this chapter presents the details of the connectionist implementation, but first this section will discuss its basic structure. The network has three basic parts, input units, grammar units, and predicate units. The input units are externally set to reflect the words being input to the parser. The grammar units calculate what actions the parser should take, and output these actions. The predicate units calculate and store the information about the parser's previous actions that will be needed to determine future actions. As shown in figure 5.1, information flows from both the input units and the predicate units to the grammar units, and from the grammar units to both the output and the predicate units. There are two sources of recurrence in this structure. Information cycles from the grammar units through the predicate units and back, and it cycles within the predicate units to maintain their state over time.

The fact that the network is highly structured and specialized to this particular parser design makes it different from most connectionist networks, but some parallels can be drawn. In some respects, the input, predicate, and grammar units correspond to the input, hidden, and output

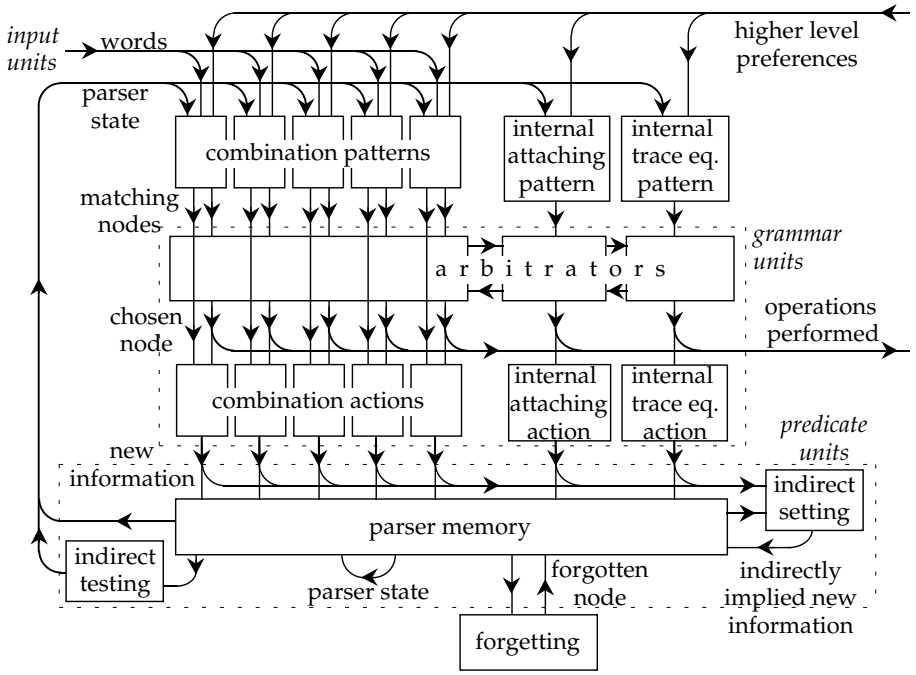


Figure 5.1: The boxalogical organization of the parser. Dotted boxes and italics labels show the the major unit groupings. Lines and arrows show flow of information. Solid boxes show computations and the parser memory.

units, respectively, of a Parallel Distributed Processing network (PDP, (Rumelhardt and McClelland, 1986)). Most obviously, the activation over the input units represents the input, the activation over the grammar units represents the output, and the activation over the predicate units is only interpreted within the module. Also, as in many PDP networks, the input and grammar/output units use a localist representation, and the predicate/hidden units use a distributed representation. The predicate unit's representation is distributed because types of phrase structure nodes map to patterns of activation across the units.[1] The reason the network is easily interpreted at the symbolic level is the addition of the temporal synchrony variable binding mechanism, not the abandonment of distributed representations. The input and grammar units use a localist representation simply for convenience. A more distributed representation of the input would be useful to represent morphological decompositions of words, and to represent word categories. A more distributed representation of the grammar would be useful to capture morphological and category generalizations across grammar entries.

Other than their structured nature, the three groups of units differ from the levels of a PDP network in the way information flows between them. First, information flows directly from the input units to the grammar units, rather than passing through the predicate units. This is possible because of the use of inhibitory links from the predicate units to the links between the input and grammar units. These link patterns are similar to the multiplicative connections of (McClelland, 1986). Also, no information flows directly from the input units to the predicate units. As in simple recurrent networks (Elman, 1990), the output of the predicate/hidden units is fed back to that layer, plus, as in (Pollack, 1990), the output of the grammar/output units is fed back to the predicate/hidden layer. This structure partitions the network into one part that decides what to do next, and

[1] In linguistic investigations, this form of representation is called a feature decomposition.

one part that decides what information about the history of the parse to keep around. In other recurrent network structures, at least one layer has to do computations for both these problems simultaneously.

Of course, the biggest difference between NNEP's connectionist implementation and other connectionist networks is the use of temporal synchrony variable binding. For units that represent properties of the situation as a whole, this has no effect, since such properties don't involve variables. Units in NNEP's network that represent properties of an entity map to multiple units in a conventional network, one for each entity that could have the property. Links map analogously; the information that NNEP's links can carry using temporal information (entity identity) has to be mapped to distinctions between multiple links.

5.2 The Connectionist Architecture

While its design is motivated by biological considerations, the Shastri and Ajjanagadde connectionist architecture is still an abstract computational model. Thus the primitives of the architecture are idealizations of the way neurons might compute. The objective of this idealization is to capture the primary characteristics of biological computation, without making the task of modeling in this architecture too complex. The level of idealization which is appropriate for the investigation done in this dissertation has a single kind of primitive unit. These units represent groups of neurons, so the system is robust against the failure of single neurons or links. Units perform simple computations and store simple state information. Links between units carry a scalar value. A link can also inhibit the activation passing across another link. For arguments for the biological plausibility of these primitives, see (Shastri and Ajjanagadde, 1990, 1993).

Section 2.2 presents the major characteristics of the S&A architecture. Its central property is the use of the temporal dimension to represent variable bindings. When active, a unit produces a pulse train. If two units are pulsing synchronously, then they are representing information about the same variable's referent, and if they are firing out of phase , then they are representing information about different variables' referents. To make synchronization simple, in this investigation I assume that the interval between pulses in a pulse train is fixed, and that two firing pulses are either synchronous or not. Thus time can be divided up into periods, and each period can be divided up into phases. A period can be thought of as a computation step which cycles through the set of variables.

The information stored in the memory is manipulated using pattern-action rules. A rule is implemented as a collection of units and links. The rule tests the temporal pattern of activation in the memory to see if it matches the rule's pattern. If a match is found then the rule modifies the contents of the memory to reflect the effects of the rule's action, given the entity or entities which were involved in the match. Rules can compute in parallel, thus allowing computation to be done at a speed which is independent of the number of rules.

5.2.1 Units and Links

This system only needs one kind of basic unit. Initially a unit sums its input and tests to see whether this sum exceeds its threshold. If it doesn't, the unit continues checking for sufficient input. If the total input at this time does exceed the unit's threshold, the unit ignores all its input for a fixed amount of time, then outputs one firing pulse. For the units used here the magnitude of

this pulse is either always 1, or is the value of the input. This output activation is always between 0 and 1. After starting to fire, the unit waits another fixed amount of time before once again checking its input to see if it exceeds its threshold. The units which store predications wait one period before firing, and do not wait at all before looking at their input.[2] Units with these timing constants are called ρ-btu units. If such a unit has a self-exciting link, then once excited, it will fire in its exciting phase every period until it receives sufficient inhibiting input in that phase. I will also use two other kinds of units, called τ-and and τ-or units, which can be constructed from sets of the above primitive units. A τ-and unit fires (with activation 1) after receiving one period of uninterrupted sufficient input, and stops firing after receiving insufficient input. Thus it computes the temporal *and* of its input over the last period. τ-or units are not used very often. They fire (with activation 1) after receiving sufficient input and stay firing until they have received an entire period of insufficient input. Thus they compute the temporal *or* of their input over the last period.

Links in this system are weighted and can either be inhibitory or excitatory. Links take the output of their source unit and multiply it by their weight to get their value. For excitatory links, their value is added to the input of their destination unit. For inhibitory links with unit destinations, their value is subtracted from their destination unit's total input. The destination of an inhibitory link can also be another link. In this case the value of the destination link is multiplied by one minus the value of the inhibitory link (*primary*(1-*inhibitory*)). Thus inhibitory activation of 1 will completely block any activation, and lesser values will let through a proportion of the activation. I will often talk about signals rather than links. A signal is a set of links, with the value of a signal being the sum of its links' values. While these signals can in general be thought of as just a conceptually convenient way to refer to sets of links, they seem to be necessary to control the complexity of link interconnections. If they are realized using units, the delay they introduce can be handled in the same way as other propagation delays.

5.2.2 Implementing Predicates

Predicates are implemented with collections of units. Some of these units fire in the phases of the entities of which the predicate is true, and some provide the circuitry used to make sure each new predication is stored by some unit. Constant predicates are represented with sets of units whose collective output has the same value regardless of the phase. Unary predicates require one set of ρ-btu units to store predications, and their collective output specifies the probability of the predicate being true of the entity in each phase. For predicates with greater arity than unary predicates, the relationships between phases are represented using fixed associations between units. For example, two associated units can store one predication of a binary predicate by having one unit storing the phase of the entity in the first role of the predication and the other unit storing the phase of the entity in the second role. In the general case the number of units required to implement a predicate in this way is proportional to n^k, where n is the maximum number of entities which can be stored by the memory and k is the arity of the predicate. Interestingly, this parser only needs to represent unary and binary predicates. Also, the use of binary predicates is limited in ways which allow their implementation to use many fewer units than would be necessary in the most general case. These predicate modules have signals which set predications to be true, set predications to be false, and output what predications are true. The value of a signal represents the probability of a predication.

[2]To be precise the delay between the start of firing and the detection of the input by another unit needs to be taken into account. Thus the units actually wait one period minus this delay before firing, and wait this delay before looking at their input.

The particular way binary predicates have been implemented reflects the use of these predicates in NNEP. The phases for the first argument of the predicate are stored in a list of ρ-btu units. For each of these first argument storage units there is a unary predicate module which stores all the phases in the second argument of a predication with the first argument. The associations between units which represent the bindings across argument positions are organized in this first-argument-centered way because the resource requirements of NNEP allow the number of nonterminals which can be in the first argument of each of the necessary binary relations to be significantly constrained. This reduces the length of the list of first argument storage units, and the number of second argument storage unary predicate modules. This optimization is the reason there are a set of rules which remove predications which are no longer needed.

The mechanisms which are used to query binary predicates are simplified by the locality constraint on rules. As discussed in section 2.2.4, this constraint means that all binary predicates are accessed through unary predicates. To do this without losing information about the relation, at least one of the nodes involved in the query must be uniquely identifiable at the time. Because of the way each binary predicate is represented, this unique node will always be in the first argument of the predicate. Thus all tests can be calculated using a signal of the form *pred_constant*, where *pred* is the predicate and *constant* is the name of the unique node. This signal specifies all those nodes which are in the *pred* relation with *constant*. For example, one such signal which is needed is *could_dom_public*, which specifies those nodes which the public node could possibly be dominated by. These signals are implemented using one τ-and node per first argument position in the binary predicate. Each of these τ-and nodes' outputs is high unless the *constant* node is the same as the node in their first argument storage unit. If one of these τ-and nodes goes low, then it gates links from the nodes in the second argument unary predicate module into the signal.

In the same way, the mechanisms which are used to set binary predicates are also simplified by the locality constraint on rules. Instead of a *pred_constant* signal, setting a binary predicate involves signals of the form *set_pred_constant*. These signals are implemented in the same way as the *pred_constant* signals, except the *set_pred_constant* signal is gated *into* the second argument storage module.

5.2.3 Implementing Pattern-Action Rules

Pattern-action rules can be implemented using links to do computations within a phase, and τ-and or τ-or units to coordinate computation across phases. Negation of predicates in rule patterns can be implemented with inhibitory links. Disjunction can be implemented with multiple links with the same destination. Conjunction can often be implemented with multiple links and a unit threshold greater than 1. In the case of $p(x) \wedge \neg q(x)$, conjunction can be implemented with links which inhibit other links. A link with value $P(p(x))$ which is inhibited by a link with value $P(q(x))$ will output the value $P(p(x))(1 - P(q(x)))$, which is $P(p(x) \wedge \neg q(x))$ (assuming independence). These mechanisms can be used to implement any pattern involving one variable, but some formula have a simpler implementation than others. The implications of the relative complexity of different patterns has not been investigated, but NNEP's pattern implementations are fairly simple.[3]

Since links can't store information, units are needed to communicate information between phases. ρ-btu units can't help, since they always fire in the same phase as their input, but τ-and and τ-or units can be used for this purpose. Since τ-and units fire after receiving one period of uninterrupted

[3] In particular, no patterns need more than two levels of inhibition. In a few cases the implementation needs a link which inhibits a link which inhibits another link, but never more. This keeps link propagation delays small.

sufficient input, they can be used to implement universal quantification across the entities in the memory. Usually the universal quantification is of the form $\forall x, \neg p(x)$, which is implemented with a τ-and unit with threshold zero and an inhibitory input from $p(x)$. This unit's output is high unless it receives input from $p(x)$, after which it stays low for one period. Such a unit can be used as a gate by having its output inhibit another link, thereby implementing $q(y) \wedge \neg\forall x, \neg p(x)$, or $q(y) \wedge \exists x, p(x)$. Activation can only flow across the inhibited link when the gate is low. τ-or units can be used similarly to implement existential quantification over the entities in the memory.

5.3 Implementing NNEP's Operations and Grammar

NNEP's grammar is compiled into a set of rules which combine a given grammar entry with the current phrase structure description. The ways these combinations can occur are characterized by the set of combination operations. In addition, there is a rule for each internal operation, and a rule for the forgetting operation. These rules share the property that their application needs to be coordinated by the arbitrators which implement NNEP's disambiguation mechanism. This coordination is done by splitting these rules into a pattern component and an action component. The patterns provide the input to an arbitrator, and the output of the arbitrator triggers the actions. The arbitrators choose among matching patterns, and pass the phase of the chosen pattern match to the pattern's action. The implementation of these arbitrators was discussed in section 4.5. In this section the patterns and actions for these rules will be presented. The patterns for these rules calculate the choice probability estimates discussed there.

The operations which do not involve grammar entries are each implemented with a single rule. The forgetting operation's rule matches on locally complete nodes and, if chosen, suppresses all the predications in the phase of the matching node and reclaims that phase for future use.[4] The strength of a match is determined by predicates which represents the probability that the node will be equated with in the future (*finished(x)*$\wedge$*parented(x)*). The state internal operations' rules look for nodes which can be equated with their respective unparented nodes, and, if chosen, transfer all the predications about the matching node to the phase of their unparented node. The matching node's phase is then made available for future use. The strengths of these matches are determined by the values of the *equatable* relationship. For these choices the arbitrator also takes into consideration the values of the *could_dom* relationship, since this relationship represents the possibility of future equation sites.

The grammar and combination operations are implemented with a large number of rules. Conceptually, there is a distinct rule for each way each grammar entry could be used by a combination operation. The patterns of these rules look for nodes in the parser state where their combination can apply. The strengths of pattern matches are determined by whatever information is available in the phase of the matching node, the next word in the input, and sometimes the subsequent word in the input. For the equationless combining operation and the case of leftward attaching where the attached node is the public node, there is no specific node where the combination occurs. The value of these matches is determined by summing over matches for nodes that could equal or dominate the root of the grammar entry. This value is passed to the arbitrator in a phase which is currently unoccupied. All these patterns are specific to words, so lexically specific constraints can be used in determining the strength of a match. The arbitrator chooses between matching patterns based

[4] For simplicity, I assume there are a fixed set of phases, some of which are occupied by nodes and some of which aren't.

on their relative strengths, plus the strengths of similar patterns that specify what combinations might be possible if internal operations are performed first.

The actions for combination rules add to the parser state the information which is implied by their combination taking place at the node chosen by the arbitrator. The signal from the arbitrator which triggers the action is used to set new information about that node. This signal also drops a gate, which causes new nodes, and sometimes information about the public node, to be added to the parser state. In the case of equationless combining and leftward attaching, information is also added about the *equatable* and *could_dom* relationships between the root of the grammar entry and other nodes. This information is taken from the pattern matches which were summed in the pattern of the rule. This allows the values of these relationships to be lexically specific.

Although it is simpler to think of there being a distinct rule for each way each grammar entry could be used by a combination operation, the actual implementation takes advantage of the commonalities between rules. The most important of these is that rules which have the same effect on the parser state share the same action component. Since patterns are implemented with links and the arbitrator grows with the number of output choices, this means that the number of units in the parser grows (linearly) with the number of different structures in the grammar, not the number of words. The number of different structures is much smaller than the number of words. There is also sharing between different combination operations for the same grammar entry. For example, equationless combining is implemented in part as attaching at a phase which is currently unoccupied. Similarly, the case of leftward attaching where the attached node is the public node is implemented in part as double attaching at a phase which is currently unoccupied. More sharing is possible, but this question involves issues far beyond the adequacy of the S&A architecture, so it has not been investigated.[5]

5.3.1 Combination Operation Rule Patterns

As mentioned in section 4.5, a combination operation rule pattern consists of a single primary link and a set of inhibitory links. The input activation to the primary link is high only when the word for the pattern is the next word in the input. The inhibitory links filter out those phases which either don't have nodes or have nodes which are incompatible with the pattern's operation or grammar entry. For example, figure 5.2 shows a pattern which has inhibitory links from the $-V$, $+A$ and $-$*finite* predicates to filter out those nodes which are not finite I's. The *has_head* and *has_verb* filter out those nodes which already have filled terminals as the values of their *head* or *verb* features. The *available* predicate specifies what phases do not represent nodes, so it is used to filter out unused phases. All patterns also have inhibitory links from the *nonfrontier* predicate, since they can all only apply to nodes on the right frontier. The *finished* predicate represents the probability that more structure will be attached below the node, so it filters out those nodes which are for independent reasons unlikely to be attached to.

The above mechanism for testing patterns checks for matches within the phase of each node. Thus any information which the pattern needs to check needs to be represented within the phase of that node. This is a manifestation of the limitations of the S&A architecture which was characterized as

[5]In particular, sharing a component of the implementation between combination rules implies that any learning which occurs for that component will be generalized to all the combinations which share the component. This issue is central to the debate over how much of the nature of language is inherent to the language mechanism, and how much is extracted from regularities in the linguistic environment. This issue cannot be resolved without an extensive empirical investigation of language learning, which is outside the scope of this work.

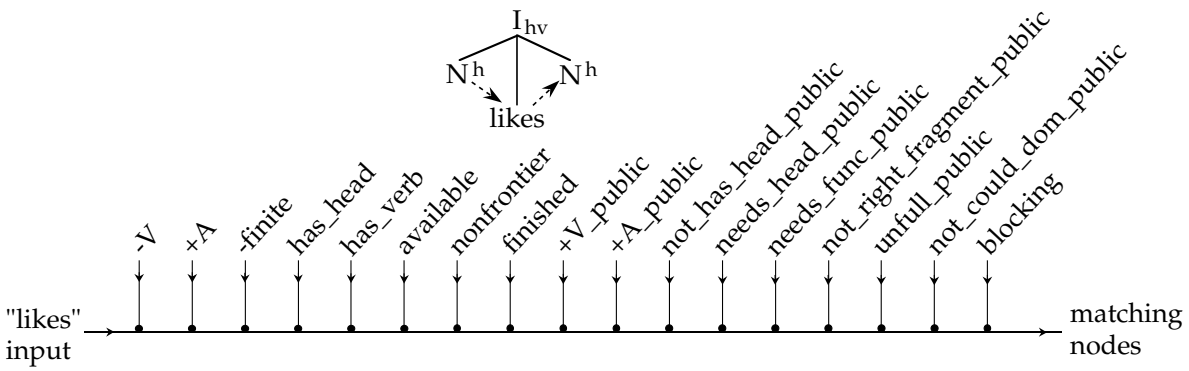


Figure 5.2: The rule pattern for double attaching with the grammar entry shown.

the constraint that all rules in the implementation have only one variable in both their antecedent and consequent. As discussed in chapter 4, the combination operations are specifically designed to comply with this constraint. However, even though the combination rules only need to propagate information about one variable, there is still information about other nodes which needs to be tested. One case of this is constraints on the public node, which can be referred to with a constant rather than a variable. Because the public node is uniquely identifiable, information about this node can be represented with constant predicates, which are implemented with signals which are constant across all phases. Practically all the information about the public node needs to be represented in this way. These signals are generated with rules of the following form.

$$predicate(x) \wedge public{=}x \Rightarrow predicate_public$$

As illustrated in figure 5.2, these signals are used in the double attaching operation to test whether the public node can unify with the lower attachment site. The *unfull_public* signal is the same as these signals, except *unfull* is itself a signal, rather than a stored predicate. This signal will be discussed below.

In addition to constant signals about the public node, operations need unary signals about nodes which are in a binary relation with the public node. For the double attaching operation, the pattern needs to test whether the matching node could dominate the public node. This signal is generated with the following rule.

$$\neg could_dom_by(x,y) \wedge public{=}x \Rightarrow not_could_dom_public(y)$$

This is an instance of the mechanism described in section 5.2.2 for querying binary predicates. Although there are two variables shown in this rule, only one is in both the antecedent and consequent.

In addition to information about the public node, combination patterns need to know information about nodes which precede the matching node. Combination patterns need to check whether combining at the matching node will block future equations which need to occur for the phrase structure description to become complete. As mentioned in chapter 4, precedence constraints between nonterminals is represented with two unary predicates, *preceded* and *preceding*. In order to check that the combination will not produce an uncompletable result, information about the completion requirements of the *preceding* nodes needs to be transferred to the phase of the *preceded* node. This could be done with the following rules.

$\neg parented(x) \wedge preceding(x) \Rightarrow gate_blocking$

$needs_head(x) \wedge preceding(x) \Rightarrow gate_blocking$

$needs_functional(x) \wedge preceding(x) \Rightarrow gate_blocking$

$needs_verb(x) \wedge preceding(x) \Rightarrow gate_blocking$

$gate_blocking \wedge preceded(x) \Rightarrow blocking(x)$

The implementation does not actually use rules like this, since this constraint is not a categorical one. These predicates (except *parented* and *preceded*) are continuous valued, and the implementation needs to calculate an estimate of the probability that the combination will block an equation which needs to be done. This calculation is done by assuming each of the predicates is independent, which is an assumption made in all calculations. The values of the predicates in the antecedents are multiplied, and the resulting values for the first four rules are summed appropriately ($x+y-xy$). This produces the value of the *gate_blocking* signal, which is then filtered so that it is only present in the phase of the *preceded* node.

Another way that a combination can result in an uncompletable description is when the lower attachment site for double attaching or leftward attaching is not on the right frontier of the grammar entry. If there are incomplete nodes in the rightmost tree fragment, then after the combination these nodes will be blocked from satisfying those needs. This information about the nodes in the rightmost tree fragment is represented in the signals *unfull* and *unfull_public*. As with the *blocking* signal, these signals are continuous valued. They are calculated in the same way, except with respect to the following rules.

$\neg parented(x) \wedge right_fragment(x) \wedge \neg anchored(x) \Rightarrow gate_unfull$

$needs_head(x) \wedge right_fragment(x) \wedge parented(x) \Rightarrow gate_unfull$

$needs_functional(x) \wedge right_fragment(x) \wedge parented(x) \Rightarrow gate_unfull$

$needs_verb(x) \wedge right_fragment(x) \wedge parented(x) \Rightarrow gate_unfull$

$gate_unfull \wedge right_fragment(x) \wedge \neg parented(x) \wedge anchored(x) \Rightarrow unfull(x)$

$unfull(x) \wedge public{=}x \Rightarrow unfull_public$

The first of these rules is for trace nodes, and it is categorical. The rest are calculated by assuming the predicates are independent evidence. The root of the rightmost tree fragment is not included in this calculation, since that information is already available in the matching phase, and the significance of such needs are dependent on the grammar entry. The root of a tree fragment is identified as the node which dominates terminals but does not have an immediate parent.

The equationless combining operation and the case of leftward attaching where the attached node is the public node are implemented by sharing components of the implementations of the attaching and double attaching operations, respectively. This can be done because combinations using the former operations are simply less specified versions of combinations using the later operations, where the equation site of the root of the grammar entry is the unspecified information. One of the components the less specified combinations use is the patterns of their more specific counterparts. These patterns specify the possible equation sites for the root of the grammar entry which are

already in the parser state. A second pattern is also needed. This pattern finds nodes where there may be attached, a grammar entry that introduces a node which will be attached to the root. As discussed in section 4.5, the value of the pattern for the less specific combination is the sum of these patterns over the last period. This value is passed to the arbitrator in a phase which is not currently occupied by a node.

From the above discussion we see that attaching and double attaching patterns are implemented as a single inhibited link, and that equationless combining and the public node case of leftward attaching are implemented by summing the values produced by a set of inhibited links over the last period. The one remaining class of combinations is those that use leftward attaching to attach a node which is not the public node. The patterns for these combinations need to do the same calculation done in the other case of leftward attaching, except the pattern match is done in the phase of the lower attachment site. Thus there need to be inhibited links which calculate what nodes could equate with the root of the grammar entry, what nodes could dominate that root via structure which has not yet been introduced, and whether the rightmost tree fragment root can equate with the lower attachment site in the grammar entry.[6] The values from the first two of these signals are summed over the last period, and the result is multiplied by the value of the third pattern, using an inhibitory link. This pattern is also inhibited by the *public* predicate, so this pattern will only match if the other case of leftward attaching does not. The result is the pattern for a leftward attaching combination which does not involve the public node.

5.3.2 Combination Operation Rule Actions

Once the arbitrator has picked a match for some combination rule's pattern, that rule's action must instantiate the effects of the rule's combination at the chosen node. This involves adding new information about the node(s) in the parser state which are equated with a node in the grammar entry, plus adding new nodes from the grammar entry. These additions are triggered by a unit in the arbitrator which fires in the phase of the chosen matching node. There is one of these action triggering units for each combination rule action. Figure 5.3 shows an example of how the signals to set the new information are generated for a double attaching combination whose pattern was shown in figure 5.2.

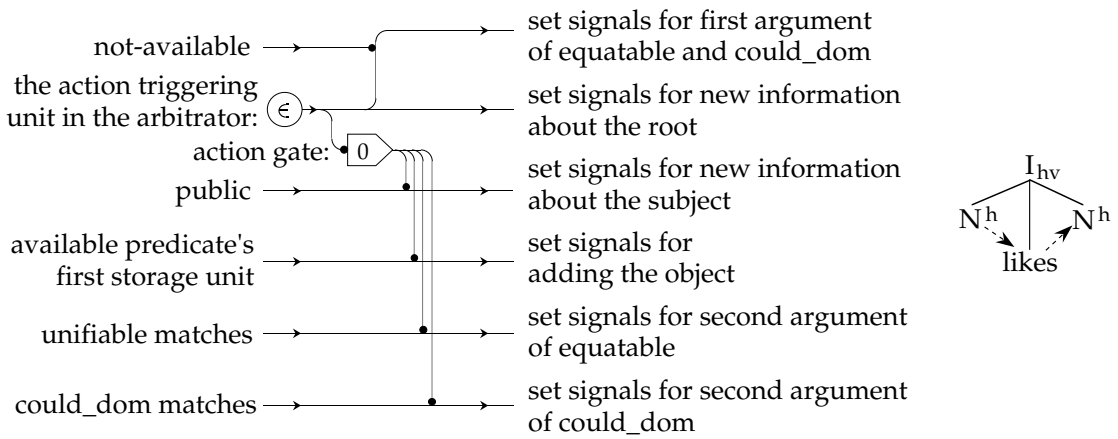


Figure 5.3: The rule action for double attaching with the grammar entry shown.

[6] Currently no portion of these calculations is shared with the double attaching combination, but it could be.

Since the information which needs to be added by an action involves multiple nodes, each action triggering node has an associated τ-and unit which only goes low for one period after its trigger node fires. This unit gates the links which set the information about nodes other than the one where the action's match occurred. For dominance instantiating, this unit gates links from the *public* predicate to the predicates for information which results from equating the public node with a node in the grammar entry. This unit also gates links which add new nodes and the information about them to the parser state. These links originate from the *available* predicate, which specifies those phases which are not being used. This predicate is implemented in a way that allows fixed links to pull out individual phases from this set. These gated links also unset the *available* predicate. Since the triggering unit fires in the phase of the matching node, new information about the matching node is simply added with links from the triggering unit to the appropriate predicates. This new information is the result of equating the matching node with a node in the grammar entry.

Combinations which use the equationless combining operation share their rule's action component with the attaching combinations for their same grammar entries, and the combinations for the public node case of leftward attaching share their rule's action component with their associated double attaching combinations. Since the patterns for the less specific combinations are signaled in a phase which is not currently occupied by a node, this has the effect of introducing a new node which is unconnected to the rest of the structure, then performing the more specific combination at that node. This results in a parser state which is almost exactly the result of the less specific combination. In addition, *equatable* and *could_dom* relations need to be specified for the new tree fragment root, the root may need to be pushed on the public node stack, and, in the case of equationless combining, the tree fragment maintenance rules need to be signaled that an addition has been made. Whether this additional information is needed is determined by testing whether the triggering signal is in a phase which is currently unoccupied. Occupied phases are filtered from the triggering signal, and the result is used to set the new information about the new root. This includes signals for setting the first argument of the *equatable* and *could_dom_by* predicates. The second arguments of these predicates are set using the values generated for the combinations patterns, gated by the action's gate. These second argument setting signals only have an effect if the first argument setting signals they are paired with go high. The setting of these binary predicates does not involve propagating pairs of variable bindings, since at any one time there is a unique node for the first argument of the predicates.

One case of equationless combining which has not yet been discussed is that for parenthetical phrases. Parenthetical phrases have not been discussed because they do not fit in the characterization of sentences as having a single connected phrase structure tree. A parenthetical phrase (as I am analyzing them) is one which has a phrase structure which is unconnected to the rest of the sentence's phrase structure. Sentence 337 is an example of this phenomena, taken from the Brown corpus. The "for his part" portion of this sentence would not be licensed to be in this position if it were part of the phrase structure of the sentence.

337 Mr. Nixon, for his part, would oppose intervention in Cuba.

To build a parenthetical phrase's structure, equationless combining is used to add a node for the root of the parenthetical phrase. This node is specified as having an immediate parent, so as to express the fact that it does not need to find a position in the phrase structure of the sentence. This means that tree fragments which are for parenthetical phrases are distinguishable in that they have no distinguishable tree fragment root.

The actions for leftward attaching combinations which do not involve the public node are implemented in the same way as the other actions. The triggering signal is used to set the new information about the root of the rightmost tree fragment, and the action's gate node is used to gate the instantiation of information about other nodes in the grammar entry. As with the public node case of leftward attaching, the patterns for the root of the grammar entry are used to set the *equatable* and *could_dom_by* predicates for the root of the grammar entry.

5.3.3 State Internal Operation Rules

The two state internal operations discussed in chapter 4 are each implemented in the same way. The rules for these operations each have two patterns, one for the possible equation sites and one for the possibility that future grammar entries will contain the site of the desired equation. The actions of these two rules cause all the information about the chosen node to be transferred to the phase of the unparented node which it is equated with. This includes the combination of information in binary predicates. There is a third state internal operation which was not discussed in chapter 4. This operation removes parenthetical phrases from the list of tree fragments, and it is called parenthetical popping.

The two patterns for the internal attaching and internal trace equating operations are generated using the *equatable* and *could_dom* relationships. The patterns which convey the probabilities for each equation that could currently be done are generated with the following rules.

$$right_fragment(x) \wedge \neg parented(x) \wedge anchored(x) \wedge equatable(x,y) \Rightarrow equatable_right_root(y)$$

$$public{=}x \wedge \neg anchored(x) \wedge equatable(x,y) \Rightarrow equatable_public_trace(y)$$

The patterns which convey the probability that future grammar entries will contain the site of the desired equation are generated using the *could_dom* relationship in the same way, using the following rules. The actual nodes which match this pattern are not used by the arbitrator, but the sum of the values is.

$$right_fragment(x) \wedge \neg parented(x) \wedge anchored(x) \wedge could_dom_by(x,y) \Rightarrow could_dom_right_root(y)$$

$$public{=}x \wedge \neg anchored(x) \wedge could_dom_by(x,y) \Rightarrow could_dom_public_trace(y)$$

These four rules are instances of the mechanism described in section 5.2.2 for querying binary predicates.

The actions for the internal attaching and internal trace equating operations are triggered by signals from the arbitrator. As with combination rule actions, these actions have τ-and units which only go low for one period after their triggering signal pulses. These units are used as gates for signals which involve the tree fragment root or trace node. One such signal is called *equate_to*. For internal trace equating, the gate unit gates the output of the *public* predicate to the *equate_to* signal. For internal attaching, the gate unit gates the output of the *anchored* predicate to the *equate_to* signal, inhibited by the *not_right_fragment* and *parented* predicates. As you might expect, the triggering signals for these actions are linked to a signal called *equate_from*. These signals are the interface to the mechanism which performs the chosen equation.

The *equate_from* and *equate_to* signals cause the information about one node to be transferred to the phase of another node. The phase of the first node is then forgotten, as described below. This process results in the effect of equating the two nodes. Because only one such equation will be done at a time, only one such pair of signals is needed. This equation mechanism is implemented with additional units for each predicate. All unary predicates have a unit which is used to store the value to be transferred. The value of the predicate in the phase of the *equate_from* signal is gated into this unit. After the unit gets input, it outputs that value continuously. This value is then gated to set the predicate in the phase of the *equate_to* signal, and the storing unit gets deactivating input. In general, continuous valued predicates treat two separate set signals as independent evidence for the predication, so if this equation combines two noncategorical values for a predicate, the two values are treated as independent. For second argument values in binary predicates, this same mechanism is used, since these are stored in unary predicate modules.[7] If only the *equate_from* node is a first argument, then that storage unit gets its phase changed to that of the *equate_to* signal. If both equated nodes are first arguments, the situation is more complicated. Such an equation involves transferring the values of the second arguments for the *equate_from* node to the second argument module of the *equate_to* node. This is done with the appropriate gates and signals.

The parenthetical popping operation removes the tree fragment for a parenthetical or appositive phrase from the phrase structure of the sentence. These fragments are distinguishable from other fragments because they have no distinguishable tree fragment root. As discussed above, the root of a parenthetical phrase is specified as *parented* when it is introduced. Since the root of a tree fragment is the one node which dominates terminals but does not have an immediate parent, for a parenthetical phrase's fragment there is no such root. The parenthetical popping rule can only apply to fragments which have no distinguishable root. Also, parenthetical popping requires that the parenthetical phrase be complete. Thus its pattern requires that there be no unfulfilled expectations or unparented nodes in the rightmost tree fragment. The action of the parenthetical popping operation simply removes all the nodes in the rightmost tree fragment from the parser state. This causes the parser to "pop" back to the state it was in before the construction of the parenthetical phrase was started.

5.3.4 The Forgetting Rule

The rule which implements the forgetting operation looks for nodes which are not likely to be involved in any future equations, and removes the chosen node from the parser state. Two predicate are used in determining what node to forget, *parented* and *finished*. Only nodes which have immediate parents can be forgotten, since otherwise this node would never get an immediate parent and the accumulated structure description would never become complete.[8] Thus the *parented* predicate represents the (categorical) chance that this node will be attached to another node. Since the *finished* predicate represents the probability that this node will have more structure attached below it, these two predicates together represent the chance that any equations will occur at a node. The value of the *finished* predicate is calculated using factors like whether the node has any unfulfilled expectations, and how long the node has been in the parser state (i.e. recency). The current implementation uses a fairly simpleminded theory of how to calculate this probability, but it is adequate for the data addresses in this dissertation.

[7] This means that the *equatable* relation needs to be represented in the form *not_equatable*, where the second argument values for a given first argument are the complement of those for the *equatable* relation. In this form the predicate grows monotonically, as do the other predicates in the system.

[8] The matrix root can also be forgotten, but the current implementation never does that.

The arbitrator for the forgetting operation simply picks nodes which are *parented* and have a *finished* value greater than a fairly high fixed threshold. It does not consider the current resource load, although I suspect that would be necessary for longer sentences. Any node which is picked has all its predications cleared and its phase is reclaimed for future use.

5.4 Maintaining the Parser State

As discussed in section 4.4, the parser's state information is represented in the connectionist implementation using unary and binary predicates. Most of these predicates are implemented in exactly the form described there, some are implemented in their complement form, and a few are implemented in both. These choices have some effect on the complexity of link interconnections, but since I have not found any situations where these issues effect the behavior of the parser, I will not discuss them here. Also, the binary relationship *could_dom* is represented in its reverse form (*could_dom_by*), so that the unparented node is in the first argument position. The remainder of this section will discuss the implementation of the rules which calculate information which is indirectly implied by the information added by operation rule actions. These rules' implementation has an affect on the time course of parsing. In order to make the parser faster, these rules are compiled into the actions of operation rules.

The rules presented in section 4.4.2 use some constants, unary predicates, and constant predicates which are implemented as signals from operation rule actions. The unary predicates are *new_node, new_foot, new_extractable, redo_equatable_public* and *unify(grammar_entry_root,x)*, the constants are *next_comb_upper_site, next_comb_lower_site,* and *next_inter_attach_site*, and the constant predicates are *equationless_combining, combine_fragments,* and *nonfrontier(next_comb_lower_grammar_node)*. The unary signal for x's such that *unify(grammar_entry_root,x)* is the result of one stage of the pattern matching process for the combination rule. The unary signal *redo_equatable_public* is generated for the site of internal attaching, attaching, and double attaching, but it is delayed until the predicates which need to be checked to determine unifiability with the public node are already set. In the case of double attaching, this signal is delayed longer to allow the the public node stack to pop the public node which is equated by the operation.[9]

The signals for constants are used to drop gates which cause a rule to fire. For example, the following rule is implemented with a τ-and unit that goes low when the *next_comb_upper_site* signal is simultaneous with the output of the *preceded* predicate. Thus this unit's output represents the constant predicate $\neg$*preceded(next_comb_upper_site)*. This unit gates the output of the *preceding* predicate to set the *nonfrontier* predicate.

$$preceded(next_comb_upper_site) \wedge preceding(x) \Rightarrow nonfrontier(x)$$

The unary signals are used as input to rule patterns. For example, the following rule is implemented with a link from the *new_node* signal to the *preceding* predicate, inhibited by the above $\neg$*preceded(next_comb_upper_site)* unit.

$$preceding(next_comb_upper_site) \wedge new_node(x) \Rightarrow preceding(x)$$

[9]The sentences in which this delay is necessary are only marginally acceptable. This fact will be discussed in section 6.2.

The constant predicates are implemented with signals from the trigger units of the operation rule actions to a τ-and unit for each constant predicate. These units then inhibit the links which implement the rules that use the constant predicates.

The rules which check the unifiability of the public node and some other node in the parser state make use of signals of the form *not_predicate_public*. These signals are the complements of those discussed in the last section, and they are constant across all phases. Ununifiable nodes are filtered out from rule patterns using inhibitory links from predicates, which are themselves inhibited by the above signals for the incompatible predicates. For example, the filter would include a link from the $+V$ predicate which is inhibited by the *not_−V_public* signal. This requires a whole lot of links.

5.5 The Time Course of Parsing

Now that the pieces of the parser's implementation have been presented, the total parsing process can be discussed. The parser's memory is initialized with a node for the matrix IP of the sentence. As each word is input, grammar entries are combined with the parser state in the memory and internal operations are performed until all the words have been processed and the parser state represents a complete description. The main loop of this computation starts by waiting for the input of a word. When a word is input, the combination rule patterns for the grammar entries for that word find what combinations are possible with the current parser state. The combination arbitrator sums these patterns and patterns for other possible ways a grammar entry for the word could be incorporated into the parser state. The combination arbitrator then compares this sum to each of the completely specified immediately applicable combinations (such as attaching to a node in the right tree fragment), and tries to pick one. If it can't, then it tries to pick one of the less specified immediately applicable combinations (such as equationless combining). If either of these kinds of combinations are picked, then after a period to absorb propagation delays, the chosen combination's action is triggered in the chosen phase. The grammar entry, operation, and phase of the chosen combination is also output to other language modules. The combination's action instantiates the new information introduced by the grammar entry and the equation(s) for the combination. The action also sends signals to the rules for maintaining the parser state, and these rules calculated and add information which is indirectly implied by the action's new information. After all the new information is added, it takes one period for the signals for indirect testing to be generated. Once this is done, as soon as the next word is input the above steps can start again.

If none of the immediately applicable combinations are sufficiently likely, then the current implementation of the arbitrator assumes that it was because there is a better option which is not immediately applicable. It then asks the internal operation arbitrators to find an immediately applicable internal operation. The internal operation arbitrators then sum only over patterns for immediately applicable options, and pick one. After a period to absorb propagations delays, the action for the internal operation is triggered in the phase of the chosen node, and the appropriate equation is done. The chosen operation and phase are output to other language modules. The chosen action also sends signals to the rules which maintain the parser state, and they add indirectly implied information. This new information may require that the signals for indirect testing be updated. Once this is done, the parser returns to trying to find an immediately applicable combination for the input word.

The method which is currently used to require that the parser produce a complete description is a grammar entry for periods. This grammar entry must attach to the matrix root of the sentence,

and that node can't be *unfull*. In order for the parser to combine this grammar entry with the parser state, the matrix root must be in the right tree fragment, and all the nodes in that fragment must be parented and have their expectations fulfilled. Such a parser state represents a complete description, so attaching the grammar entry for the period at the end of the sentence guarantees that the parser has finished the parse.

5.6 Characteristics of the Implementation

The most important characteristic of this implementation of NNEP in the S&A architecture is that it exists. This is all that is necessary for this portion of the argument for the computational adequacy and linguistic significance of the architecture. The other portions of the argument can all be discussed at the more abstract level of constrained symbolic computation. However, there are some characteristics of the implementation which are worth mentioning, and which must be discussed at the level of units and links. These characteristics pertain to speed and space requirements of NNEP.

Because NNEP's grammar is implemented in a set of pattern-action rules which compute in parallel, NNEP's speed is independent of the size of its grammar. Even the propagation delays which would be introduced by larger and larger grammars would not slow the parser down. The implementation uses one period to compensate for propagation delays, so as long as the propagation delays do not exceed one period in length they will not effect NNEP's speed. It is very important for the speed of a parsing model to be independent of the size of the grammar, since any real grammar for a natural language would be very large.

The implementation of NNEP's grammar is also interesting in that grammar entries which have the same effect on the parser state share the same units. Thus the number of units in the implementation is proportional to the number of different structures in the grammar, not the number of words. Adding a new noun, for example, only requires adding new links to the units which are used for all the other nouns. The number of different structures in a natural language grammar is much smaller than the number of words. Also, the number of units needed per structure is fixed (and fairly small), so the number of units in the implementation grows linearly with the number of structures.

One of the motivations for connectionism is the possibility that architectures of this kind could help bridge the gap between abstract theories of cognition and biological computation. This is one of the express objectives of Shastri and Ajjanagadde for their architecture. They present a plausible and fairly direct relationship between the primitives of the architecture and neurons. This relationship allows estimates to be made of the real time and space characteristics of a biological version of the implementation of the parsing model. No claims are being made here about the biological validity of the particular implementation that has been chosen, but it is still possible to see how fast and how small such a biological implementation could be using this particular implementation. Also, at our current level of knowledge the estimates that can be made are rather approximate, so even these upper bounds are only useful to determine whether the model's speed and space requirements are in the right general area. But, even this rough check would not be possible without the use of a biologically plausible connectionist architecture. The speed of a biological version of the implementation is fairly easy to calculate. Shastri and Ajjanagadde report that a typical value for the time between a neuron's activation spikes is about 20 msec. This is the biological correlate of a period in the architecture. Each completely specified combination operation takes at most six

periods, so if no state internal operations are needed, then a sentence could be parsed at a rate of about eight words per second. This assumes that neurons can propagate synchronous firing in a single period, but assuming this would take two periods (which (Shastri and Ajjanagadde, 1993) reports as plausible) would require only three more periods per operation, which would result in a rate of about five and a half words per second. The use of state internal operations would slow down the parser about the same amount as additional words. This puts the speed of the parser in the right general area for real time language processing. The space requirements of a biological version of the implementation is harder to estimated. The current simulation code uses about 3500 units, two thirds of which are for the grammar, which has 114 different structures. A unit corresponds to very roughly about a hundred neurons, and the number of structures for a complete grammar might be around fifty times more. That would give us an estimate of about 10^7 neurons, plus or minus a couple orders of magnitude. There are about 10^{12} neurons in the brain, so this estimate is well within the fraction of the brain which appears to be used in sentence processing. Once again, let me emphasize that such estimates are only useful to determine whether the implementation's time and space characteristics are in the right general area to comply with biological constraints. However, even such rough estimates point to the potential of future work using this model to investigate the relationship between abstract theories of language processing and the biological basis of that processing.

5.7 The Simulation of the Implementation

The above connectionist implementation of NNEP has itself been implemented using the Rochester Connectionist Simulator (RCS). This simulation is performed at the level of units and links discussed above. RCS provides a simple visual display, which is used to show the patterns of activations for important predicates, and signals. It also displays the important parts of the arbitrators, including all the grammar entries' combination rules. RCS also provides a command line interface, which is used to input words and to issue disambiguating commands when the implementation is not able to make the choice. The output of the parser is printed to another window. This output specifies what operation is being performed, what grammar entry (if any) is being used, and what nodes are involved. The results from the use of the simulation will be discussed in chapter 6.

An example of the output produced by the simulation is given below. This output was produced for the sentence "Who ate the pizza.". The first section was produced during parsing, and the second section was produced when the parser was done, to show the final state of the parser's memory. The words were input to the parser as fast as possible. Since the parser looks at two words at a time, combinations are always done with the next to most recent word. Figure 1.3 on page 19 depicts this example. After the parser's memory is initialized with the matrix root I node, "who" is attached to this I node, and the N for "who" and the N trace are added. Since a matrix CP wh-word means that the sentence is a question, the matrix inflection has to be inverted. However, if an inverted auxiliary is used, then the trace would not be governed (see section 6.2). Thus a special grammar entry is needed which allows inflection to be manifested on the verb in the case of matrix subject gaps. This grammar entry combines with the matrix root and the trace node using double attaching, and the N object is added. At this point the subject trace node is finished and has a parent, so it is forgotten. The information about that node is printed for our convenience. The printed predicate values can range from 1 to 1000, where 1000 represents a probability of 1. After that "the" is added using equationless combining, so as to allow for genitives. Since equationless combining is implemented as attaching to a previously nonexistent node, the action which printed

this message said it was attaching. "Pizza" is then truly attached the the N introduced by "the". At this point the next word to combine is ".", which wants to attach to the matrix root. The parser first tries to find an attachment site for ".", but realizes it needs to do an internal operation first. It then picks the one internal operation which can be immediately applied, attaching "the pizza" as the object of "ate". This involves forgetting the variable for the object node and transferring all its information to the node for "the pizza", as indicated in the output. When this is done, "." can be attached to the matrix root, which completes the parse. After that the information about the nodes which remain in the parser state were printed, along with a statement of the fact that the parse completed successfully.

```
Step 1, Node 0: I'm the matrix root.
Input: who
Input: ate
Step 3, Node 0: 'matrix_wh_N' is attaching to me.
Step 3, Node 1: Node 0 of 'matrix_wh_N' reporting for duty.
Step 3, Node 2: Node 1 of 'matrix_wh_N' reporting for duty.
Input: the
Step 9, Node 0: 'matrix_trace_trans_verb' is double attaching below me.
Step 9, Node 2: 'matrix_trace_trans_verb' is double attaching above me.
Step 9, Node 3: Node 0 of 'matrix_trace_trans_verb' reporting for duty.
Input: pizza
State: (Node:2, I-:1000, A-:1000, parented:1000, has_head:1000,
        has_func:1000, nonfrontier:1000, finished:1000,
        right_fragment:1000, matrix_fragment:1000, last_lower_eq:1000)
Step 11, Node 2: Forget my variable.
Step 16, Node 4: 'determiner' is attaching to me.
Input: .
Step 22, Node 4: 'complete_noun' is attaching to me.
Step 33, Node 3: Forget my variable.
Step 33, Node 3: Equating with equated node.
Step 33, Node 4: Equated with equating node.
Step 39, Node 0: 'period' is attaching to me.

Sentence description is complete.
State: (Node:0, I+:1000, A-:1000, finite+:1000, inverted+:1000,
        matrix_root:1000, has_head:1000, has_func:1000, has_verb:1000,
        anchored:1000, finished:782, right_fragment:1000,
        matrix_fragment:1000, last_upper_eq:1000)
State: (Node:1, I-:1000, A-:1000, parented:1000, has_head:1000,
        has_func:1000, anchored:1000, nonfrontier:1000, finished:862,
        right_fragment:1000, matrix_fragment:1000)
State: (Node:4, I-:1000, A-:1000, parented:1000, has_head:1000,
        has_func:1000, anchored:1000, finished:747, right_fragment:1000,
        matrix_fragment:1000, extractable:1000, last_foot:1000)
```

Chapter 6

Testing the Parsing Model

The previous three chapters of this dissertation presented a specific model of parsing in the Shastri and Ajjanagadde connectionist computational architecture. The next two chapters use this parser (NNEP) to argue that the S&A architecture is computational adequate and linguistically significant[1] for recovering the syntactic constituent structure of natural language sentences. This chapter presents the portion of this argument which relates directly to natural language data. The primary concern of this dissertation is the computational adequacy of the architecture, so this chapter mostly looks at the ability of NNEP to handle a variety of issues in parsing natural language sentences. By showing that a parser implemented in the architecture can be adequate, the architecture is shown to be computationally adequate. For the architecture to be linguistically significant, it not only has to be computationally adequate, it has to be constrained in ways which inform the study of natural language. Although this portion of the argument is less well developed than the first, there are some constraints on natural language which can be explained by the need to compensate for the limitations of the architecture.

Before discussing the specific tests which have been done, some discussion of the nature of these tests is necessary. Empirical investigations into the nature of natural language sentences usually start by classifying sentences into those which are "good" and those which are "bad". This is a distinction which is independent of the theories involved in the investigation (i.e. a pretheoretic distinction). The investigators must then choose which of these sentences are pertinent to the questions being addressed in the theories being investigated. As a first step, I limit the data of concern here to phenomena which are pertinent to sentence level processing, relying on intuition and tradition to define this class.[2] I will call the good sentences in this class "acceptable" and the bad sentences "unacceptable".

Since I am only investigating one module of the sentence processing system, the data needs to be limited further. By definition, a sentence is acceptable only if all the various stages of sentence processing complete successfully. Thus any adequate model of constituent structure parsing must be able to recover the constituent structure of any acceptable sentence (given the necessary inputs from other modules). On the other hand, if a sentence is unacceptable, then any one of the stages of processing may have failed. Thus a model of constituent structure parsing does not need to reject all unacceptable sentences. We do, however, eventually want a complete model of all the

[1] The term "linguistic" is being used to refer to the study of language in general, and not specifically to competence linguistics.

[2] This level of phenomena includes more phenomena than "grammatical" but still does not include things like trains coming through the wall and interrupting a sentence.

stages of sentence processing, and such a model will have to account for all unacceptable sentences. In investigating a single module of this complete model, it is worthwhile to ask what unacceptable sentences are ruled out by a given model of the module. By investigating these questions of unacceptability, we get a better understanding of how the module model might fit in a complete model of sentence processing, and thus of the linguistic significance of the model. However, only failure to explain the acceptability of a sentence would constitute a counter example to the model.

Although all acceptable sentences are relevant for the adequacy of a complete model of constituent structure parsing, NNEP only needs to be tested on sentences that pose particular problems for the S&A architecture. The objective of this testing is not to demonstrate the adequacy of NNEP in particular, but to use NNEP to demonstrate that an adequate parser could be implemented in the S&A architecture. Thus it isn't necessary for NNEP to address every kind of acceptable sentence, but only to address those phenomena which are of particular concern given the limitations of the S&A architecture.[3] This chapter demonstrates NNEP's ability to handle these phenomena. The argument that the set of phenomena addressed here is sufficient to demonstrate the computational adequacy of the S&A architecture is given in section 7.1.

As for adequacy, the investigation of linguistic significance is only concerned with NNEP to the extent that NNEP manifests the characteristics of the S&A architecture. However, it is often difficult to argue that no parser implemented in the architecture could parse a given unacceptable sentence, so some of these arguments involve auxiliary assumptions. These assumptions are all independently motivated by efficiency and simplicity considerations, so they are still explanatory to a significant degree. Due to time constraints and the interests of the parties involved, more effort has been put into providing these explanations in some areas than in others. The phenomena which are investigated are sufficient to demonstrate that the limitations of the S&A architecture do have linguistic significance, and they indicate that future investigations of this nature are justified.

The testing of whether NNEP is adequate for recovering the constituent structures of sentences has been done in the following areas. The evidence for the linguistic significance of the S&A architecture is mostly given in the second and fourth of these areas.

- phrase structure analyses
- long distance dependencies
- representing local ambiguities
- resource bounds
- diversity of language

In each of these areas the investigation proceeds by selecting a set of data which is representative of the phenomena in the area, and then NNEP is tested on the data. The first area tests whether NNEP's grammars can express those aspects of linguistic competence which are pertinent to constituent structure parsing. For this test the phrase structure analyses from (Kroch, 1989) are used. This paper is also used to test NNEP's ability to recover long distance dependencies. In both these first two areas, particular attention has been paid to characterizing the unacceptable example sentences. The problem of representing local ambiguities in sentences is investigated using the example sentences in (Gibson, 1991). This test addresses the question of how local ambiguities which people don't have trouble maintaining can be maintained in NNEP's representations. The

[3]Some of these phenomena also require the ability to rule out unacceptable sentences, because otherwise the parser would be mislead by false ambiguities when parsing acceptable sentences.

question of how NNEP makes disambiguation decisions has not been tested, and this issue will only be touched on in this chapter. This fact makes predicting unacceptable examples of local ambiguities difficult. The examples in another chapter of (Gibson, 1991) are used to test NNEP's ability to stay within the resource bounds specified in chapter 4. A specific strategy is proposed for handling the bounds on the depths of the public node stack and the tree fragment list, and with this strategy these bounds account for a number of unacceptable examples of center embedding. The data in (Gibson, 1991) is not sufficient to make specific arguments about strategies for handling the bound on the number of nonterminals, but all the acceptable sentences are handled without violating this bound.[4]

The first four areas test NNEP on specific phenomena which are of particular concern given the limitations of the architecture. The last area, diversity of language, includes all syntactic phenomena found in natural language, whether they be theoretically important, lexically or construction specific, or surface obvious. While there is no particular reason to believe that a parser implemented in the S&A architecture would have difficulty in handling the large quantity and wide variety of phenomena found in natural language, it is necessary to guard against the possibility that unforeseen problems will arise in handling one of these phenomena. Since any hand picked set of data will, by its very nature, include some phenomena and exclude others, the only way to avoid biases (intentional or otherwise) in checking for unforeseen problems is to test NNEP on sentences which have been randomly selected from the population of acceptable sentences. As an approximation to such a test, analyses and strategies have been given for parsing a set of fifty thirteen word sentences which were randomly selected from the Brown corpus. Of course there are some phenomena in language which NNEP is not currently equipped to handle. The diversity of language test requires that these excluded phenomena be made explicit, and that for each such phenomena, arguments be given as to why they should not pose any particular problem for extensions to NNEP within the S&A architecture.

In the rest of this chapter the specific tests just outlined will be discussed in detail. All five tests are concerned with demonstrating the adequacy of NNEP. The linguistic significance of the architecture is mostly argued for in the discussion of recovering long distance dependencies and handling resource bounds, plus some evidence will be given in the discussion of representing local ambiguities. The acceptable sentences from (Kroch, 1989) and 20 of the sentences form the Brown corpus were run on the computer simulation discussed in section 5.7. These sentences are listed in appendix A, along with the 15 manual intervention which were needed for disambiguation decisions that the simulation could not make. The remaining acceptable sentences from this set of tests (181 sentences) were simulated by hand. While undesirable, this was necessary due to the extremely time consuming nature of handcoding the grammar entries and running the examples. Because the simulation was done at an extremely low level, and simulation tools that are appropriate for this type of network were not yet available, coding and debugging was rather difficult. Hopefully as more people work in this connectionist architecture, appropriate tools will become available. Running examples was also difficult, due to the lack of a trained disambiguation mechanism. This meant manual intervention was necessary for many disambiguation decisions. This would be a particular problem for the sentences from the tests on representing local ambiguities and handling the parser's resource bounds, since the timing of the disambiguation decisions is crucial to the

[4]By its nature, any test of adequacy will involve a broad, and therefore shallow, test of the pertinent data. This is in contrast to the majority of dissertations, which involve a narrow and deep investigation. The reader should keep this distinction in mind when considering any particular portion of this test. The standards which are appropriate for a narrow and deep investigation are not necessarily appropriate for each of the areas in a broad and shallow investigation.

predictions of the model. Future work on automatically training the network should alleviate this problem, and therefore running these sentences on the computer simulation is being delayed until then.

6.1 Phrase Structure Analyses

Because the limitations of the S&A architecture place constraints on a parser's representation of grammatical information, NNEP needs to be tested on its ability to encode and use the grammatical information that is necessary to accurately characterize the constituent structures of the language. To illustrate NNEP's power and expressiveness, the phenomena discussed in (Kroch, 1989) have been investigated.[5] This section characterizes how the phrase structure analyses of English sentences used in (Kroch, 1989) can be specified in NNEP's grammars.[6] The following section compares the mechanism by which NNEP recovers long distance dependencies with that given in (Kroch, 1989). This paper was chosen because it surveys important issues in these areas, and because the grammatical framework used is Tree Adjoining Grammar (Joshi, 1987a), which expresses grammatical information in a rather similar way to Structure Unification Grammar.

The analyses presented in this section map the projection of a lexical (N, V, A, or P) head and all its associated functional projections (IP, CP, DP) into a single SUG node, and specify the distinguished positions associated with these projections using features in the SUG node. As was discussed in section 3.1.3, this mapping allows NNEP to make the best use of its limited memory capacity, and it reduces the need for rules which involve more than one nonterminal. The specific distinguished positions used in the mapping presented here are not intended to apply to languages other than English, and there are probably even cases in English for which they will prove inadequate. However, the same approach can be used to accommodate any schematized local collection of nodes in a competence analysis by putting as much information as necessary into the feature structure information in nodes.[7]

Because all the nodes used in the SUG representation are associated with a particular lexical head, there are only four categories of SUG nodes, N, I, A, and P. These will be represented with the feature decomposition [−V,−A] for N, [+V,−A] for I, [−V,+A] for A, and [+V,+A] for P. All these types of nodes have the distinguished position *head*, which specifies the terminal which the node's category is named after. For N nodes this is the head noun, for I nodes this is the word bearing inflection (tense and agreement), for A nodes this is the head adjective or adverb, and for P nodes this is the preposition. In addition, N and I (−A) nodes have the distinguished position *functional_head*. In N's this is the determiner, and in I's it is the either the complementizer or the wh- word which is adjoined to S′ (CP). The same feature can do double duty for complementizers and wh- words because in English the two positions are never both filled, and they can be distinguished on the basis of the lexical item (or a feature if necessary). I nodes also have the feature

[5] The current simulation of the connectionist implementation runs on all the acceptable data sentences from (Kroch, 1989), and produces the right constituent structure. There are 18 such sentences. Manual intervention was needed for 5 disambiguation decisions, in cases where the simulation could not decide. These sentences and interventions are given in appendix A.

[6] While it is important that a parser's grammars express the necessary information about linguistic competence, these grammars are not designed for use in the study of linguistic competence. They are designed for use by the parser. Thus many of the issues which are important in representations for theories of linguistic competence do not apply to the grammars described here.

[7] I should stress that the SUG grammars characterized here are for use by NNEP. They are not intended to be a competence theory of grammar, and they do not express all the information which is important in the characterization of natural language grammar in general.

verb, which specifies the verb. The position of a sentence's subject is also needed for representing ordering constraints, but this can be done by limiting the combination operations that a pre-subject or post-subject adverb can use to attach to the I node, and therefore no distinguished position is needed for the subject.

The above set of distinguished positions leaves one significant characteristic of a projection unrepresented. This is the distinction between arguments and adjuncts. In (Kroch, 1989) adjuncts are represented as Chomsky adjoined to the phrase they are a constituent of, while arguments are simply attached to the phrase. In most cases this distinction is represented in NNEP's grammar by specifying the attachment of arguments in the grammar entry for the head (or verb, etc) of the phrase, while specifying the attachment of adjuncts in the grammar entry for the head of the adjunct. However, PP's which are treated as arguments in (Kroch, 1989) are treated as adjuncts in this SUG analysis. These PP's can be distinguished on the basis of the representation of predicate-argument structure. In NNEP the phrase structure analyses are assumed to be augmented with a predicate-argument level of representation such as Lexical Functional Grammar's f-structure (Kaplan and Bresnan, 1982). For example, the I node for the verb "saw" might have [with-obj:[f-struct:x], f-struct:[instrument:x]] in its feature structure, where "x" is a variable used to represent coreference. Since this level of representation has not been implemented yet in NNEP, information about what PP's are arguments is represented with predications on nodes. In the example for "saw", the syntactic significance of the f-structure information is represented with the predicate *θ-to-with*, signifying that the head of the node assigns a theta role to the object of a PP headed by "with". Note that the number of prepositions for which such predicates must exist is small.

Most of (Kroch, 1989) is concerned with the analysis of long distance dependencies. The only difference between the structures produced by that analysis and those represented by NNEP's grammar is that when an adjunct has a long distance dependency, the modified node is the site of the dependency, not the modifier. The constraints used by NNEP to simulate the constraints on long distance dependencies given in (Kroch, 1989) are discussed more thoroughly in the next section, but here I outline how some of those constraints can be expressed in an SUG grammar. A few of the constraints don't appear to be expressible at this level, but they have a natural treatment in terms of the processing done by NNEP, as will be discussed in the next section.

In addition to the *θ-to-P* predicates introduced above, the next section uses two predicates to express the grammatical information about what long distance dependencies are possible. The *extractable* predicate specifies those nodes which can be the site of a dependency, and the *foot* predicate specifies those nodes which a dependency can cross. In other words, *extractable* nodes can be extracted and *foot* nodes can be extracted out of. This information can be specified in SUG using two features, *domain* and *path*, to constrain the domain in which a trace node can equate.[8] These features have terminals as their values. The *path* feature is used to define the long distance characteristics of an extraction domain. If a node is *foot*, then its *path* feature's value is the same as the *path* feature of the root of the grammar entry. Thus if this root is equated with another *foot* node, a chain of coreferential *path* features will be formed. If a node is not *foot*, then it has the word of the grammar entry as its *path* feature value. The matrix root also has an instance of a string as the value of its *path* feature, so all chains of coreferential *path* features will get a filled terminal as its value from the top node in the chain. The *domain* feature is used to define the local characteristics of an extraction domain. If a node is *extractable*, then its *domain* feature refers to

[8]While NNEP uses the *extractable/foot* representation of this information, it is interesting to show that this competence information can be expressed at the level of the grammatical framework. This is in contrast to some other constraints, which are characterized below in procedural terms.

the same node as the *path* feature of the root of the grammar entry. Thus all the *extractable* nodes in grammar entries whose roots are in the same chain of *foot* nodes will have the same word as the value of their *domain* feature. If a node is not *extractable*, then its *domain* feature refers to the word of the grammar entry. This puts such nodes in a domain by themselves. The domain in which a trace node can equate is constrained by making its *domain* feature coreferential with the *path* feature of the lowest node which dominates it. Thus the trace node can only equate with a node which is in its dominating node's domain, since otherwise the values of the *domain* features won't unify. This constrains long distance dependencies to only stretch across *foot* nodes, and only have sites at *extractable* nodes, as desired.

The local phrase structure analyses from (Kroch, 1989) are expressed in an SUG grammar in a very similar way to that used for Lexicalized Tree Adjoining Grammar (Schabes, 1990). The trees used in (Kroch, 1989) are completely ordered, but in SUG linear precedence constraints are used to allow the underspecification of this ordering, as is done in (Joshi, 1987b). A word has in its grammar entry the node which represents the word's preterminal's immediate parent, any −A arguments which the word subcategorizes for, optionally the node which it modifies (if it is the *head* of a +A node), any nodes for which the word fills a distinguished position (*head, functional_head,* or *verb*), any trace nodes coindexed with the above nodes, and all the known syntactic information about these nodes and their relationships to each other. This syntactic information includes information about the feature structure labels of the nodes, immediate dominance relationships, dominance relationships, and linear precedence relationships. Obligatory arguments and all modified nodes have *head* terminals without their word's specified, so as to express their expectation for a head. The same technique can be used to express the expectation for a determiner, complementizer, or verb.[9] Any local linguistic constraints on possible structures can be enforced with restrictions on possible grammar entries.

The above division of information among grammar entries in general complies with both the Tree Adjoining Grammar (TAG) generalization that a grammar entry should contain a predicate and all its arguments, and the SUG generalization that a grammar entry should express all the information known given the presence of a word. Because it complies with this TAG generalization, this grammar entry domain is large enough to express predicate-argument relationships over it, thus allowing the specification of a predicate-argument (f-structure) level of representation in the same grammar entries. In SUG this level of representation can be expressed within the feature structure labels of nodes. The equation of nodes will cause the unification of the entities in the predicate-argument structure.[10] This division of information among grammar entries is essentially the same as that used in Lexicalized TAG (LTAG), except information about long distance dependencies is

[9] In addition to these lexical grammar entries, there should be a few grammar entries which do not have words in them. These nonlexical grammar entries would express the possibility for constructions such as relative clauses which do not have overt wh- words. However, for simplicity, these structures are lexicalized by precombining them with lexical grammar entries. This gives grammar entries a slightly larger domain of locality than that just described. The only nonlexical grammar entries which are used in the current implementation are one node grammar entries that introduce parentheticals and appositives, and a structure for topicalization.

[10] Although the constructions addressed here allow the predicate-argument structure to be embedded in the SUG constituent structure, I believe a parser which handles coordination would have to have separate levels of representation for these two types of structure. Assuming the Combinatory Categorial Grammar analysis of coordination (Steedman, 1985) is essentially correct, the result of coordinating two constituent structure fragments is a single constituent structure fragment whose syntactic functionality is subsumed by each of the coordinated fragments. However, the predicate-argument structure for this resulting constituent structure includes two distinct sets of entities and predications. Thus a many-to-one mapping from predicate-argument structure entities to constituent structure entities is necessary. This requirement precludes embedding predicate-argument structure in the feature structures of the constituent structure nodes.

expressed with the wh- word, not with the word whose argument is extracted.

There are two conflicts between the TAG and the SUG generalizations about what information should be included in a grammar entry. The first is exceptional case marking verbs. Since an exceptional case marking verb provides the Case which allows the infinitival sentence to have an overt subject, the SUG generalization says that the subcategorization for both the subject and the infinitival verb phrase should be in the grammar entry for the verb. This results in a structure which expresses Case relationships. On the other hand, since the subject is an argument of the predicate in the infinitival verb phrase, the TAG generalization says that the subcategorization for the subject should be expressed in a grammar entry for the infinitival verb phrase. This results in a structure which expresses thematic relationships. Since the SUG generalization is important for NNEP, I will choose the former representation. The thematic relationship can still be expressed in the representation of predicate-argument structure, since the subject and the infinitival verb phrase have nodes in the same grammar entry. This analysis explains why heavy-NP shift can occur for the subjects of infinitival sentences, but not for the subjects of finite sentences. The examples which are ruled out in (Kroch, 1989) using their analysis (footnote 12) can be ruled out by stipulating that the infinitival verb phrase argument of an exceptional case marking verb is a *foot* node, and the embedded subject argument is not. The same type of constraint could be used to rule out sentences like (1.).[11]

(1.) *Who$_i$ did you give the parents of e$_i$ a present?

The second conflict between the above TAG and SUG generalizations is the specification of argument prepositional phrases. The TAG generalization says that the argument PP should be in the grammar entry for the subcategorizing word. The SUG generalization says that a prepositional phrase should have the same grammar entry regardless of whether it is an argument or an adjunct, since the preposition itself can't determine this information. As will be shown in section 6.3, the later property is necessary in order to delay resolving ambiguities between argument and adjunct PP attachments until disambiguating information is found. As proposed above, special features in the predicate-argument structure representation can be used to allow a word to refer to the semantics of the object of its argument PP without actually mentioning the argument PP in its syntactic structure. In this way the predicate-argument structure can obey the TAG generalization, while still allowing the syntactic constituent structure to obey the SUG generalization.

6.2 Recovering Long Distance Dependencies

This section compares the mechanism by which NNEP recovers long distance dependencies with the analysis given in (Kroch, 1989). Most of the constraints on long distance dependencies are characterized at the level of NNEP's grammars, as was specified in the last section. These grammatical constraints are compiled into features on nodes (i.e. *extractable* and *foot*), and enforced by the rules that calculate long distance dependencies. However, some of the constraints on long distance dependencies are predicted by the need to restrict these rules' access to trace nodes to the public node. Since the public node is the top node on the public node stack, these rules can only access the most recently introduced trace node. This constraint is used to explain the that-trace

[11] To help prevent confusion about what sentence numbers are from what sources, I will number my sentences as "(n.)", the sentences from (Kroch, 1989) and (Gibson, 1991) as "(n)", and the sentences from the Brown Corpus as "n".

effect, the cases of subject islands that precede inflection, and the limited possible extractions out of wh- islands. The later phenomena are particularly interesting, because accounting for this data required Kroch to go outside the power of TAG. Thus by accounting for this phenomena with a computational constraint, the competence theory of long distance dependencies can be simplified. This explanation for wh- island constraints is also interesting in that it subsumes Pesetsky's path containment condition (Pesetsky, 1982). Unless stated otherwise, the numbering and examples in this section are from (Kroch, 1989).

The SUG grammar entries described above use dominance relationships to express the constraint that a trace must be c-commanded by the node which binds it. They also use *domain* and *path* features to express what nodes can be extraction sites and what nodes can be extracted out of. NNEP's representation of this grammatical information uses the predicates *extractable* and *foot*, rather than the *domain* and *path* features. As described in chapter 4 and in the last section, these predicates directly represent what nodes can be extraction sites and what nodes can be extracted out of. Nodes which are specified as *extractable* are checked to see if they could equate with any trace nodes which the root of their grammar entry could potentially dominate. Nodes which are specified as *foot* are specified as potentially dominating any trace nodes which the root of their grammar entry could potentially dominate.[12] As with the coreference of *path* features, the iterative calculation of potential dominance (*could_dom*) relationships establishes an arbitrarily long chain of nodes. As with the the coreference of *domain* features with *path* features, trace nodes can only equate with allowable extraction sites which are within one grammar entry of this chain.

This domain of extraction roughly corresponds to the domain allowed by TAG, ignoring for the moment most linguistic constraints on this domain. In nonmulticomponent TAG, a gap and its filler are specified in a grammar entry with the predicate to which the gap is an argument or adjunct. This relationship between the filler and gap can then be stretched by adjoining auxiliary trees at nodes between them. Since adjunctions cannot change thematic role assignments, these adjunctions always take place above any phrase for which there is a chain of thematic roles from it to the gap. The constraints on the domain of a TAG grammar entry restrict the length and nature of this thematic chain. These restrictions define a local domain within which gaps can be specified. I will call this domain the local extraction domain of the gap. This local domain is analogous to the local domain around potentially dominating nodes in which a trace node can equate. After the adjunctions, the relationships between roots and feet of the adjoined auxiliary trees taken together form a path from the phrase of the filler down to the local extraction domain of the gap. This path is analogous to the chain of nodes which can potentially dominate a trace node. Like NNEP's potential dominance chain, the TAG "foot path" can be arbitrarily long and the gap needs to be within a local domain around the end of the path. Because of constraints on the domain of grammar entries, each TAG root-foot relationship in this path will correspond to one or two of NNEP's root-foot relationships. This correspondence extends to the cases in (Kroch, 1989) which use multicomponent TAG. In these cases the derivation includes one multicomponent adjunction into an elementary tree. This step involves an adjunction at an empty NP and an

[12] In some cases whether a node is a *foot* node can't be determined within the domain of a single grammar entry, given the way this domain has been defined. For example, the object of the preposition "of" is a *foot* node when it attaches to the N for "picture", but not in general. This can be handled with a predication on the N for "picture". The grammar entries for "of" are sensitive to this predication, and specify whether its object is a *foot* accordingly. As will be discussed below, the predicates introduced in the previous section for expressing the distinction between argument and adjunct PP's can also be used for this purpose. Since this argument/adjunct distinction is assumed to be the result of information in the predicate-argument structure, the fact that *foot* prepositional phrases are always arguments suggests that the *foot* predicate should also ultimately be represented at the level of predicate-argument structure.

adjunction at an S′ which dominates that NP. The relationship between this S′ and this NP thus becomes a segment in the path from the filler's phrase to the local extraction domain of the gap. This segment also corresponds to one or two of NNEP's root-foot relationships. As we will see, this segment is constrained in the same way as the other segments of this path.

The correspondence just discussed allows many of the constraints from (Kroch, 1989) to be expressed as constraints on possible foot nodes in NNEP's SUG grammar entries.[13] Constraint (17) from (Kroch, 1989) (copied below) can be enforced by requiring SUG's foot nodes to be lexically governed within their grammar entry, and requiring the maximal government domain of these nodes to be the root of their grammar entry.[14]

(17) The foot node of a complement [non-adjunct] auxiliary tree must be lexically governed, and its maximal government domain must be the root node of the tree.

As discussed above, in an elementary tree where a multicomponent adjunction takes place the relationship between the empty NP and the S′ is also treated in SUG as a chain of root-foot relationships. Thus the above SUG constraint also requires that such empty NP's must be lexically governed, and the **TG** relationship from (Kroch, 1989) (the transitive closure of lexical government) must hold between the S′ and the empty NP. If the S′ is the root of the elementary tree, then the above SUG constraint enforces constraint (19) in (Kroch, 1989) (copied below) for noncoindexed empty nodes.

(18) For any node X in an elementary tree α, initial or auxiliary, X is properly governed if and only if one of the following conditions is satisfied.

i) the maximal government domain of X is the root node of α;
ii) X is coindexed with a "local" c-commanding antecedent in α.

(19) For any node X in an elementary tree α, initial or auxiliary, if X is empty, then it must either be properly governed or be the head of an athematic auxiliary tree.

If the S′ is not the root of the elementary tree then the **TG** relationship may not extend to the root, and thus the empty category might not have the root as its maximal government domain, as is required by (19). This will happen when the elementary tree where the multicomponent adjunction takes place is for a relative clause. The Chomsky adjunction configuration at the top of the relative clause structure will prevent the root from being the maximal government domain of the noncoindexed empty NP. The grammaticality of sentence (2.) below (my example) indicates that this additional requirement is not desired, and thus I will not try to capture it in the SUG analysis. More recent work in TAG generates (2.) by substituting the S′ of the relative clause into the Chomsky adjunction structure, and thus the S′ is the root of its elementary tree even in the relative clause case (Kroch, personal communication). As long as any such S′ is the root of its elementary tree, the above SUG constraint fully enforces constraint (19) in (Kroch, 1989) for noncoindexed empty nodes.

[13] Taken literally, the SUG grammar entry doesn't specify "foot" nodes. However, the relationship specified above between the SUG grammar's *path* and *domain* features and NNEP's *foot* and *extractable* predicates makes the identification of foot nodes in the SUG grammar easy. Therefore, I will talk of the SUG grammar as specifying foot nodes.

[14] The transitive nature the **TG** relation in (Kroch, 1989) allows the maximal government domain portion of (17) to be specified individually on SUG foot nodes, even though there may be more than one SUG root-foot relationship in a single TAG root-foot relationship.

(2.) John only wanted to talk to the man who$_i$ he knew what$_j$ paper to give a copy of e$_j$ to e$_i$.

Constraint (19) also applies to empty nodes which are coindexed. Since the coindexation of the gap is expressed in what is being called TAG's local extraction domain, this is a constraint on this local extraction domain. The constraint has three cases, extraction of adjuncts, extraction of subjects, and extraction of other arguments. Adjuncts must be coindexed within their elementary tree. This eliminates the possibility of using multicomponent tree sets for the extraction of adjuncts, since in these sets the coindexation is between two trees within the set. Thus the local extraction domain of adjuncts is the domain of a single elementary tree which is rooted by a node of a V projection.[15] TAG grammar entries do not allow recursion on any type of node, except Chomsky adjunction. If the adjunct modifies a V projection then it must be the root V projection, since otherwise there would be recursion on S′. If the adjunct modifies an N projection then it must be a child of a node in the root V projection, since such adjuncts are PP's and thus no PP node can intervene between the N projection and the elementary tree root. In the SUG analysis the extraction of adjuncts is expressed with a long distance relationship between the modified node and the filler, not between the modifier and the filler. This is necessary because modifiers always provide their own immediate dominance relationship with their modified node, and thus if the modifying node was moved it would have no place to equate in the phrase of the modified node. Thus a trace node which represents an adjunct filler (i.e. those which do not have filled heads) must only be allowed to equate with an I node which potentially dominates it or with an N child of one of the nodes which potentially dominate it. Also, such a trace node cannot be potentially dominated by an N node, since this would be the equivalent of using a multicomponent TAG analysis. The later constraint must simply be stipulated: no trace nodes which do not have filled heads can be specified as potentially dominated by a node which is of type N. The former constraint is the same as the constraints on the local extraction domain of arguments, except that extraction out of complement PP's is not allowed. This can be enforced by applying the constraints about to be discussed for argument extraction to these unheaded trace nodes, plus stipulating that the object of a complement PP will not be considered for possible equation with an unheaded trace node.

Coindexed arguments other than subjects must have the grammar entry root as their maximal governing domain. Because recursion is not allowed in a grammar entry, the argument will either be a child of the grammar entry's root projection or a child of a PP which is a child of the root projection. Since in the SUG analysis PP's have both their parent and their children in their grammar entry, both these cases have the gap within one SUG grammar entry of the root projection for the TAG local extraction domain. Thus NNEP's local extraction domain is large enough to cover the TAG local extraction domain. All that remains is to constrain the SUG domain so that it exactly matches the TAG local extraction domain for these cases. For the case of children of the root projection, the extractability of an argument can be determined within the locality of the grammar entry, and thus their eligibility for equation with a trace node which is passed to their root projection can simply be stated in the grammar entry using the predicate *extractable*. For the case of children of PP's, the extractability of an argument is dependent on both the head of the PP and the head of the root projection. This is the problem of distinguishing argument PP's from Chomsky adjoined PP's. As discussed above, this problem is solved with predications on the root projection which specify what PP's it takes as arguments. For example, the V node for "saw" would have the predication *θ-to-with*, indicating that it assigns a theta role to the object of a PP headed by "with". The grammar entries for PP's are sensitive to their *θ-to-x* predications and specify their object as *extractable* accordingly. These mechanisms ensure that NNEP will only

[15] I'm using the term "V projection" to include VP, S, and S′.

consider for equation with a trace node those arguments which the TAG analysis treats as within its local extraction domain.

The last case of constraint (19) from (Kroch, 1989) to be considered is that of coindexed subjects. Since subjects are never lexically governed, they must be "locally" coindexed. The locality for subjects is stricter than that for adjuncts, namely they must be adjacent to their antecedent governor in their grammar entry. This means that all the material which intervenes between the filler and the subject gap in the resulting tree must come from adjoined auxiliary trees. In the SUG analysis given so far all the material from the filler down to the local extraction domain of the gap must come from the equivalent of auxiliary trees, but there may be material in the local extraction domain which precedes the subject gap. The possible types of such unwanted material which are discussed in (Kroch, 1989) are complementizers, wh- words adjoined at S′, "do" when used as in subject-aux inversion, and topics. The case of "do" can be ruled out by treating such post-inflection subjects in the same way as all other rightward subcategorized arguments. Since this subject argument is not lexically governed, it will not be specified as *extractable* by its grammar entry, and thus will not be considered for equation with any trace nodes when "do" is attached. This split between arguments which precede the terminal of their grammar entry and those which follow it is justified because NNEP handles these cases with two different mechanisms. Like most overt pre-inflection subjects, pre-inflection subject gaps are filled using the double attaching combination operation. On the other hand, post-inflection subject gaps are filled using the internal trace equating operation, which requires that the gap site first be found by the local movement rule, which only applies to *extractable* nodes.

The remaining unacceptable cases of coindexed subjects all have material other than inflection to the left of them in their local extraction domain. Given some assumptions about the grammar entries for complementizers, these cases are all ruled out by the requirement that NNEP's access to trace nodes obey a stack discipline. As specified in chapter 4, trace nodes and some tree fragment roots are put on the public node stack, and all the rules NNEP uses to calculate long distance dependencies are constrained to only apply to the top node on this stack. Thus only the most recently introduced trace node is available to be equated with a gap site. This clearly will rule out the case of a wh- word preceding the subject gap in the local extraction domain, because that word will add a trace node to the stack, and thus the previous trace node will not be available for the double attaching operation to equate it with the subsequent subject gap. Since topicalization is also handled using trace nodes, the same argument covers topicalized constituents. This leaves complementizers.

Although the grammar entry for a complementizer will not introduce a trace node, it can introduce a node which prevents subject extractions. Since complementizers are only on finite clauses, and in English all finite clauses subcategorize for subjects, the presence of a complementizer implies the presence of a subject. Thus the grammar entry for a complementizer can safely express this expectation for a subject.[16] Indeed, the grammar entry for a complementizer should express this

[16]There is a slight chicken-and-egg problem here, but only if evidence is viewed categorically. A complementizer must always be followed by an overt subject in order to justify expressing the expectation for a subject in its grammar entry. Complementizers are always followed by finite clauses, and finite clauses always have overt subjects, unless they have subject gaps. As long as subject gaps don't occur after complementizers, a complementizer can express its expectation for a subject, thereby ruling out subject gaps after complementizers. But, viewed categorically, if a person ever hears a that-trace violation, they will no longer expect subjects after complementizers, and the that-trace violating dialect will spread through the language community like the plague. Of course, the human language learning mechanism is much more robust than this. It would require repeated exposure to that-trace violations before a speaker would change their grammar entries. Opportunities for these violations are rare compared to the use of complementizers in general, and there is no motivation for using a complementizer in the case of subject extractions.

expectation, since grammar entries are supposed to express everything that is known about the phrase structure of the sentence given the presence of the word. This information makes the decision of how to incorporate the subject into the current description easier. A complementizer expresses this expectation for a subject by having a −A (N or I) node in its grammar entry, as shown in figure 6.1. As will be discussed in section 6.4, in this situation NNEP won't put the subject node on the public node stack, but whether the subject is on the stack or not, this node prevents the equation of the trace node with the subject node in the following finite verb's grammar entry. If the node is put on the stack, then it is the top node on the stack, so the trace node is not available for double attaching when the finite verb is encountered. If the node is not put on the stack, then it is the rightmost tree fragment. This means the trace node is not in the rightmost tree fragment, and thus is not available for double attaching. Thus in either case the equation of the trace node with the subject node in the following finite verb's grammar entry is blocked by the presence of a node which expresses the complementizer's expectation for a subject.[17]

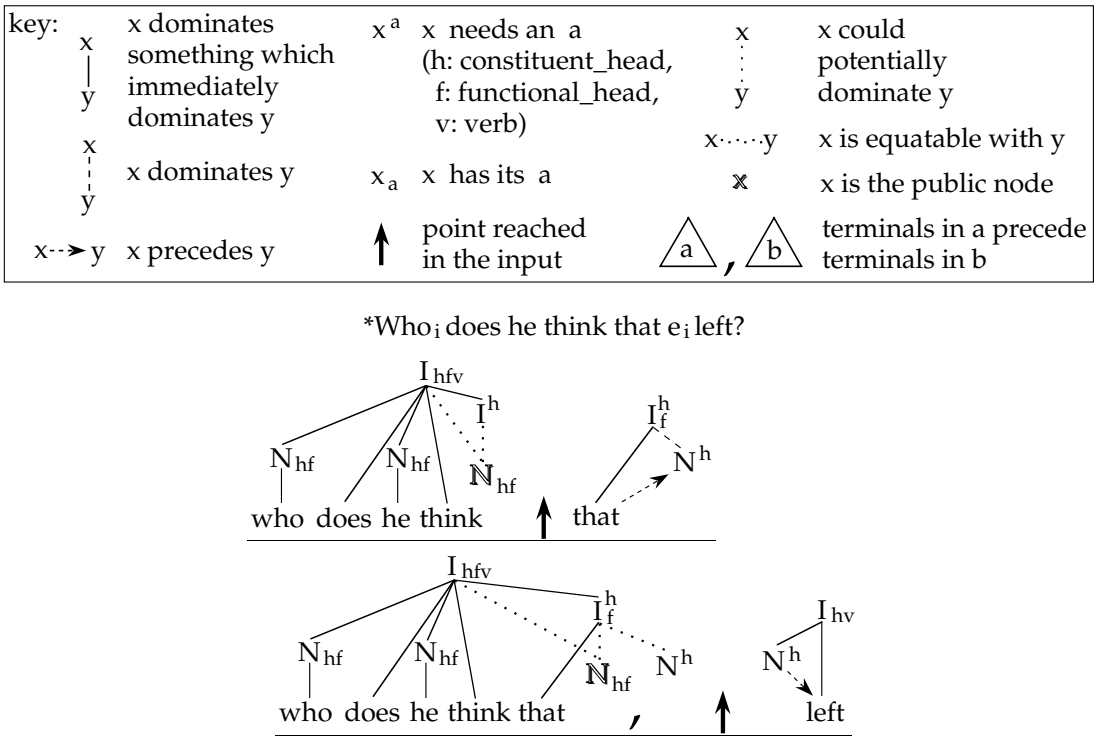


Figure 6.1: An example of how the that-trace effect is enforced.

To summarize the above discussion, for English, constraints (17) and (19) from (Kroch, 1989) can be simulated by NNEP using local constraints specified in grammar entries, plus the constraints imposed by using a stack for trace nodes, and requiring that no trace nodes which do not have filled heads be specified as potentially dominated by a node of type N.[18]

Therefore the expected stable state for a language like English is to have the that-trace effect. On the other hand, it is predicted that evil linguists could raise a child who did not have the that-trace effect in their dialect of English.

[17] Although I am not arguing for the language universality of the constraints proposed here, it is interesting that the above explanation for the that-trace effect does not work for pro-drop languages. In pro-drop languages the presence of a complementizer does not imply the presence of an overt subject, and thus the grammar entry for a complementizer will not include the node which blocks the extraction of the subject. Pro-drop languages do not exhibit the that-trace effect.

[18] This last constraint is ugly. As far as I can tell the extraction of modifiers of NP's is always ungrammatical,

The only constraints in (Kroch, 1989) which have not yet been discussed are the constraints that there never be more than one fronted wh- word in a grammar entry, and that there never be more than one fronted wh- word adjoined to the same S′ during a derivation. The first implication of these constraints is that there can't be more than one wh- word adjoined to a single S′. The SUG representation of phrase structure analyses discussed in the previous section rules out this case because adjoined wh- words are specified with the distinguished position predicate *functional_head*. Since in SUG such a position is specified as the value of a feature in the projection's node's feature structure, only one word can fill this position for a given projection. The nonlocal effects of this constraint are enforced with NNEP's stack discipline constraint on trace nodes. Since Kroch's constraint on allowable derivation structures is outside the formal mechanisms of TAG, it is desirable to eliminate it from the competence theory. By providing an explanation of the same phenomena at the level of the parser, this competence constraint can be eliminated, thereby simplifying the competence theory.

In general the public node stack will block wh- island violations because only one trace node can be passed down the tree at a time. Because only the most recently introduced trace node can have potential dominance relationships calculated for it, any previous trace nodes will be blocked from finding a gap in the same domain. The exception to this generalization occurs in a situation where two trace nodes have been specified as potentially dominated by the same node. If double attaching is used to attach a subtree at the potentially dominating node and equate the most recently introduced trace node with its gap, then the movement rules can apply to the other trace node. In this way one trace node can be passed over another, thereby violating the wh- island. All the acceptable violations of wh- islands given in (Kroch, 1989) occur in these circumstances.

As an example of how wh- islands can be violated, consider sentence (60a) and (60b) from (Kroch, 1989).

(60a) I know which book$_i$ the students would forget who$_j$ e$_j$ wrote e$_i$.
(60b) I know which book$_i$ the TAs told us that the students would forget who$_j$ e$_j$ wrote e$_i$.

The pertinent portion of the parsing of sentence (60a) is shown in figure 6.2. In both sentences, the trace node introduced by "which book" is first calculated to be potentially dominated by the object of "forget". Another trace node is then introduced by "who", and "who"'s grammar entry specifies that it is potentially dominated by the same node. Double attaching is then used to simultaneously attach the trace node from "who" as the subject of "wrote" and the I for "wrote" as the object of "forget". Because the trace node for "who" has now been equated with its gap, the trace node for "which book" is now on the top of the stack, thus making it available to be checked for equatability with the object of "wrote". After the *equatable* relation is specified, internal trace equating can be used to do this equation. In contrast, if the trace node for "who" had to be passed farther down the tree before it found its gap, then the trace node for "which book" would not have the necessary potential dominance relationships for the movement rules to apply to it, as illustrated in figure 6.3.

The acceptable violations of a wh- islands given in (Kroch, 1989) which involve extracted adverbial phrases also come under the above exception.

(56a) ?What$_i$ were you wondering [how$_j$ to say e$_i$ e$_j$]?

which subsumes most cases covered by this constraint. The remaining cases involve the extraction of PP arguments to NP's, such as in "?[About whom]$_i$ did you write a book e$_i$?". The constraint rules out sentences like "*[About whom]$_i$ did you make a copy of a book e$_i$?". Hopefully a more detailed investigation of such extractions would result in a more general and elegant characterization of this phenomena. (Kroch, 1989) does not discuss these extractions.

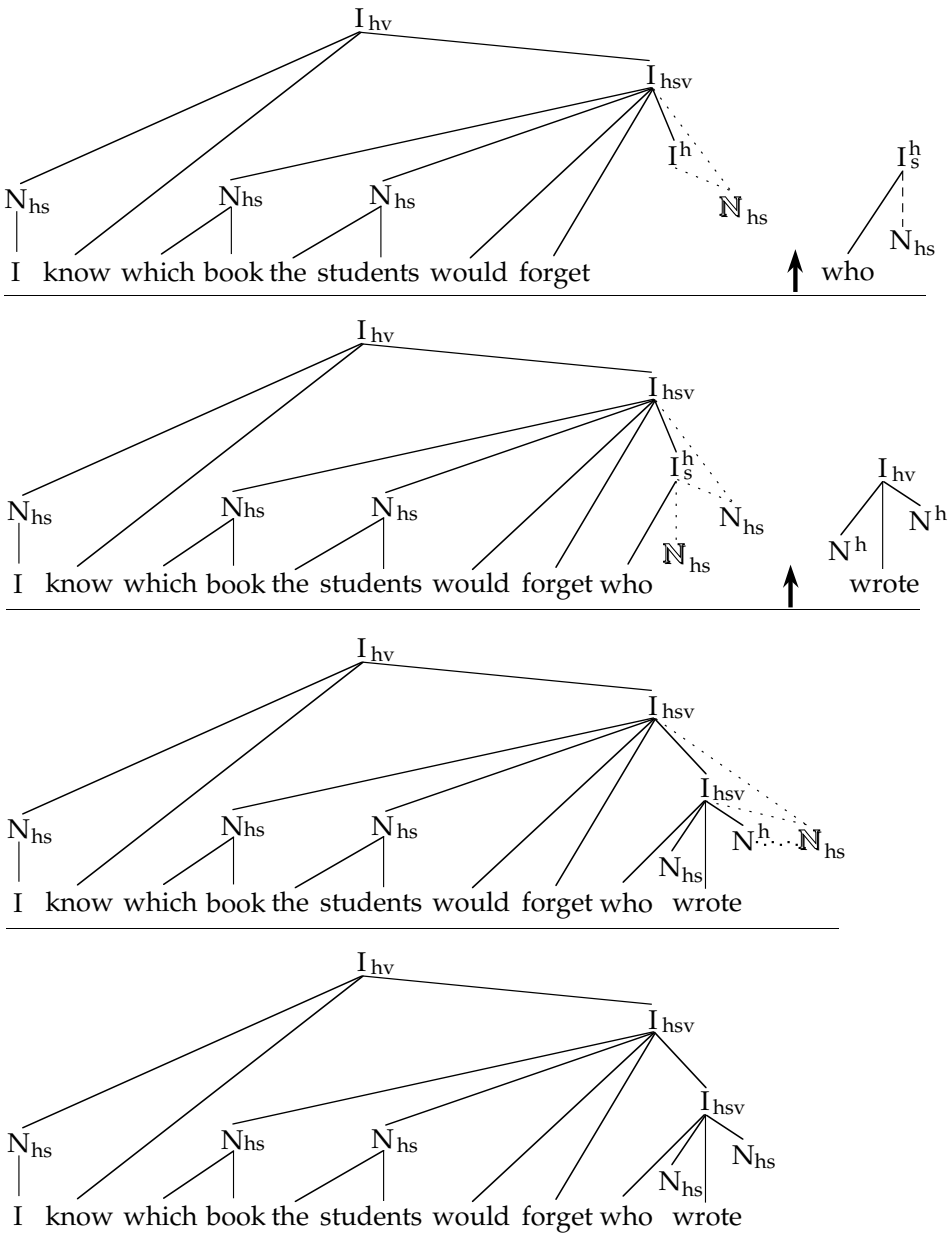


Figure 6.2: The end of the parse for sentence (60a).

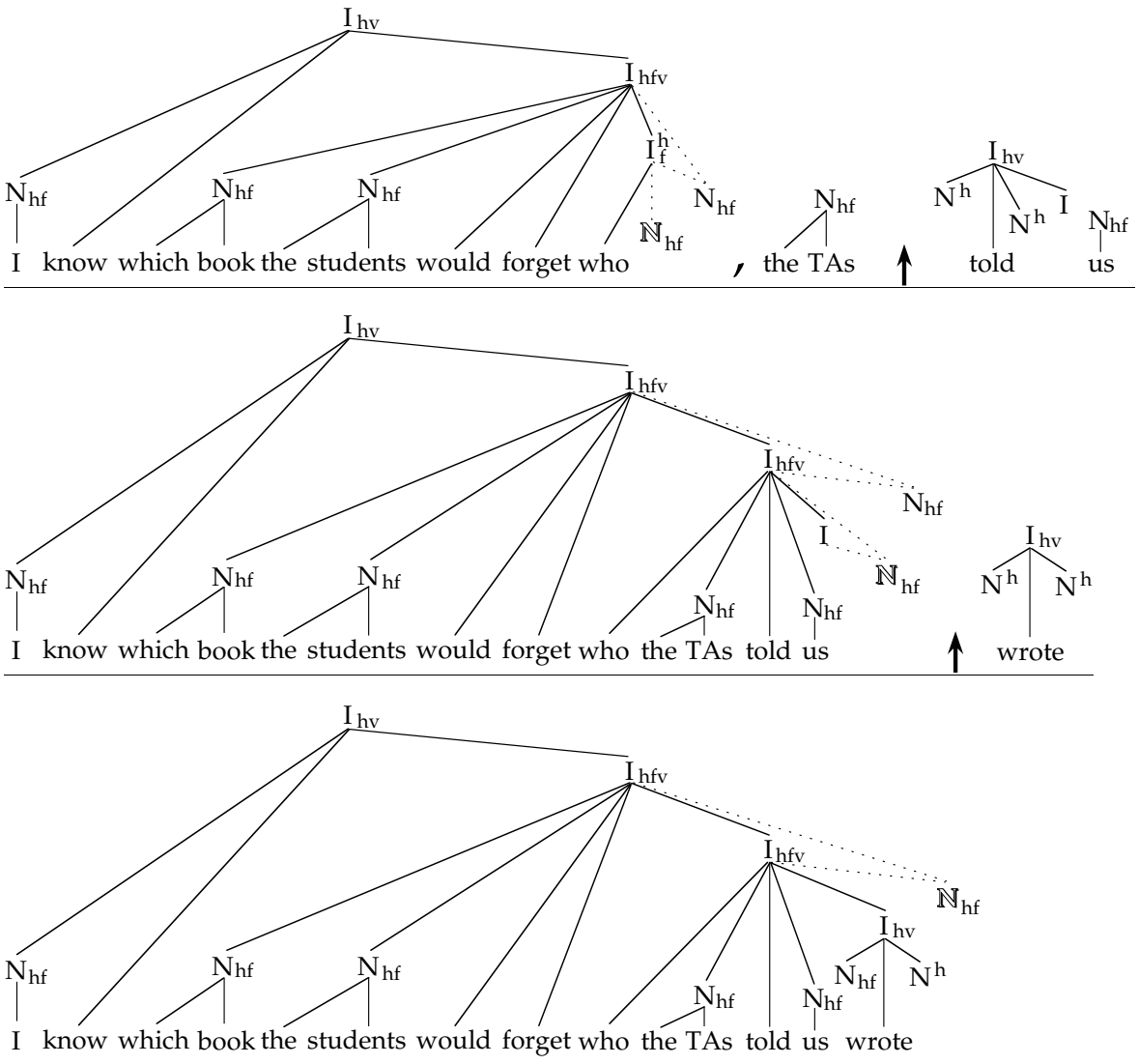


Figure 6.3: An example of how wh- islands are enforced by the stack constraint on trace nodes.

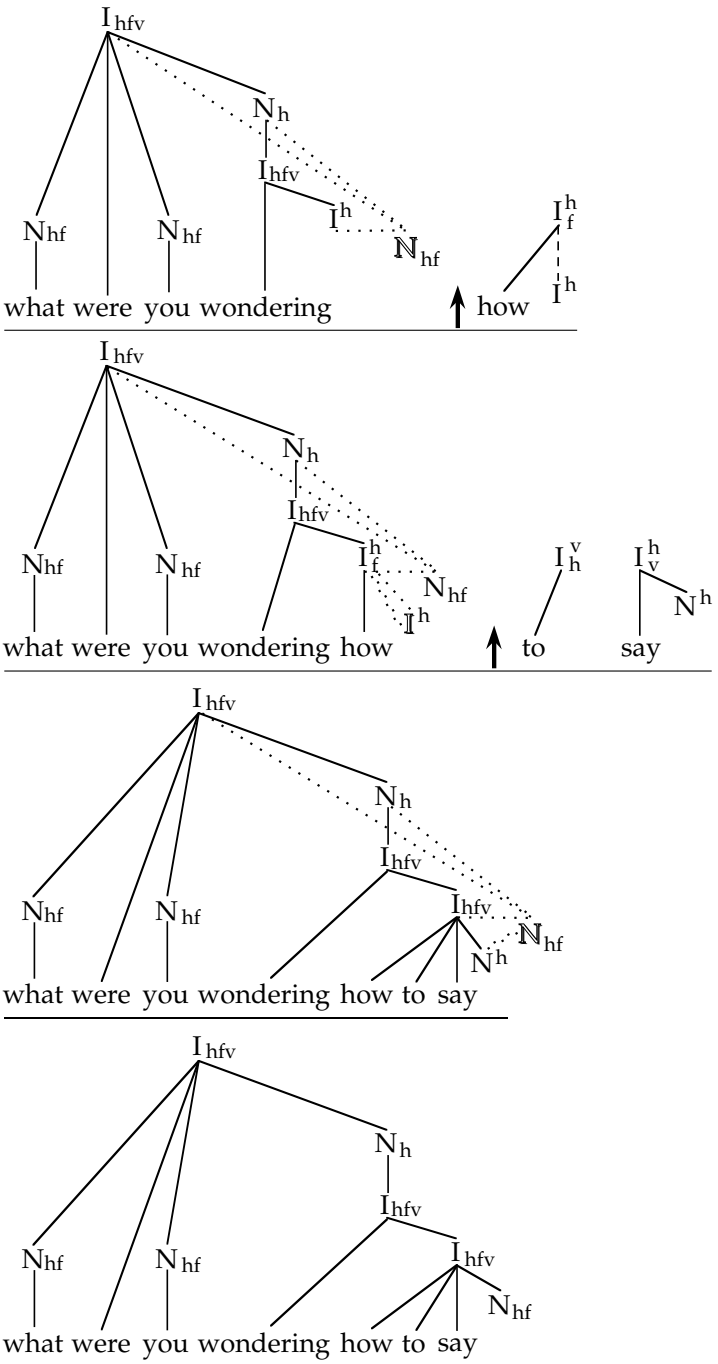


Figure 6.4: The end of the parse for sentence (56a).

The representation of the gaps shown in Kroch's (56a) show a pattern of crossed dependencies, which are ruled out by the stack constraint on trace nodes. However, the SUG analysis of the extraction of modifiers does not actually coindex the modifiers, it expresses the dependency on the modified node. Thus the trace node introduced by an adverbial wh- word must equate with the modified node, as shown in figure 6.4. Therefore, under this analysis (56a) does not have a crossed dependency. The trace node for "how" can be equated with the I node for "say", and then the trace node for "what" can be equated with the object of "say".[19]

Unfortunately the simple generalization just given will treat as acceptable several sentences which are unacceptable. Sentences (61a) and (61b) from (Kroch, 1989) are examples of this.

(61a) *I know which book$_i$ the students would forget who$_j$ e$_j$ told us that Dorthy Sayers wrote e$_i$.
(61b) *I know which book$_i$ the students would forget who$_j$ e$_j$ told us who$_k$ e$_k$ wrote e$_i$.

Since in (60a) the trace node for "which book" is available to be tested for equatability with the object of "wrote", in (61a) it should be available for the calculation of a potential dominance relationship with the object of "wrote". This would result in NNEP accepting (61a) and (61b). This can be fixed by stipulating that the time course of the movement mechanism is such that, after a double attaching combination, checking for possible equations for trace nodes is delayed until after the subject trace node is popped from the stack, while calculating potential dominance relationships is not. Such a time course is compatible with NNEP's implementation. This allows violations of wh- islands where the outer coindexation goes one grammar entry below the inner coindexation, but does not allow the outer coindexation to go any further, as it does in (61a) and (61b).[20]

The remaining cases from (Kroch, 1989) for which the above analysis over generates are the following:

(52a) *How$_i$ did he wonder [what$_j$ you had said e$_j$ e$_i$]?
(56b) *How$_i$ were you wondering [what$_j$ to say e$_j$ e$_i$]?

The exclusion of these sentences is complicated by the acceptability of:

(26) On Thursday$_i$, what$_j$ will you buy e$_j$ e$_i$?

Since in this analysis topicalization is treated with the same mechanism as wh- movement, it is hard to distinguish between moved adverbs and topicalized adverbs. However, the ungrammaticality of the following sentence points to a distinction between these cases:

[19]Given the frequency of constructions using "how to", it is plausible that the parser would decide to equate the trace node for "how" as soon as it sees "to", rather than waiting to see "say". This would have interesting consequences in conjunction with the constraint about to be proposed. In particular, it would predict the grammaticality of "?What$_i$ were you wondering [how$_j$ to take a picture of e$_i$ e$_j$]?", as opposed to the ungrammaticality of "*What$_i$ were you wondering [who$_j$ e$_j$ took a picture of e$_i$]?". This may also help explain why sentences like "?What$_i$ were you wondering [how$_j$ the actor said e$_i$ e$_j$]?" are less acceptable than (56a) (footnote 14 in (Kroch, 1989)).

[20]The fact that the mechanism for calculating long distance dependencies needs to treat the double attaching operation as a special case may help explain why all the examples involving a wh- trace being extracted over a wh- subject gap are somewhat degraded in their acceptability. It is possible that some people have learned this special case for calculating equatability, some people have learned it for calculating potential dominance, and some people haven't learned it at all. A statistical version of this argument might be able to explain the grey scale. This would predict that the more time you spend reading these examples, the more acceptable they seem, which appears to be the case.

(3.) *On Thursday$_i$, John wondered [what$_j$ you will buy e$_j$ e$_i$]?

This pattern can be captured with the generalization that an adverb extraction cannot violate a wh- island if there is an alternative extraction site for the adverb which is syntactically (although possibly not semantically) allowable. In (52a), (56b), and (3.) the adverb could have been extracted from the matrix sentence, which would not violate the wh- island. In (26) there is no alternative but to interpret the topic as extracted from the same phrase as "what". This generalization can be incorporated into NNEP's mechanism for disambiguating between possible gaps. If the following sentence is ungrammatical, then the generalization could be extended to all types of trace nodes:

(4.) ??Who$_i$ did you want _ to forget how$_j$ to find e$_i$ e$_j$?

At the point when "how" is encountered it is possible that the gap for "who" is the subject of the infinitival sentence, as in "Who$_i$ did you want e$_i$ to forget how$_j$ to speak e$_j$?".

Other investigations have handled the sentences from (Kroch, 1989) discussed in the previous paragraph through a distinction between referential and nonreferential fillers. "On Thursday" is referential, while "how" is not. If this is the appropriate generalization, then the phenomena is outside the scope of this investigation, since no attempt has been made to combine NNEP with a discourse level of representation.

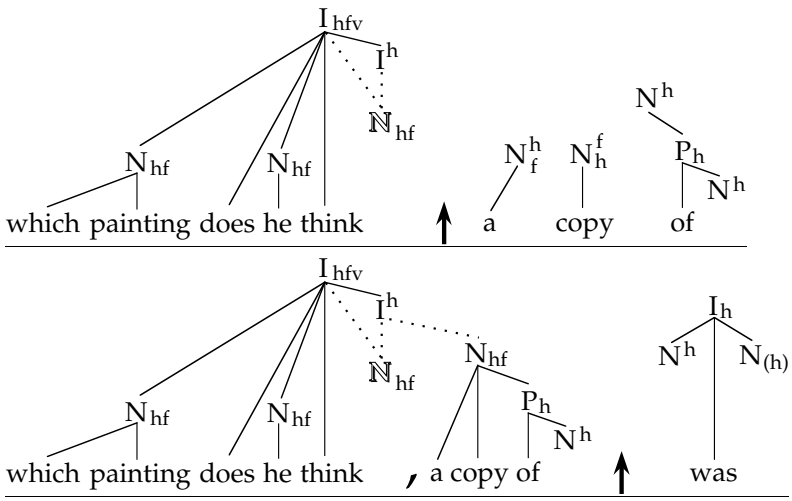


Figure 6.5: An example of how subject islands are enforced.

It is worth noting that the mechanisms NNEP uses to comply with the locality constraint on rules have implications beyond those discussed above. In particular, they rule out extraction out of pre-inflection subjects. Above it is implied that the fact that extraction out of pre-inflection subjects is disallowed is specified in grammar entries in the same way as other positions which aren't lexically governed. However, NNEP's mechanism for recovering long distance dependencies cannot handle extraction out of pre-inflection subjects, regardless of the lexical specification. The reason for this is the same as the reason that the node introduced by the grammar entry for a complementizer to express its expectation for a subject blocks the extraction of the subject. When a pre-inflection subject node is added to the parser state, it has no immediate parent because its verb has not yet been input. Thus, as illustrated in figure 6.5, the subject must be added to the list of tree fragments using equationless combining. Since there is no equation at a node potentially dominating the trace,

the movement rules do not apply, and no node in the subject will be checked for equatability with the trace. Thus NNEP can't look for gaps in a pre-inflection subject for fillers which precede the subject, which is the constraint against extraction out of subjects. This fact means that the double attaching operation does not need to depend on the extractability of the grammar entry's lower equated node. Only the local movement rule needs to be sensitive to this information.

6.3 Representing Local Ambiguities

Because NNEP is a deterministic parser, it is important to test whether NNEP can handle the local ambiguities in natural language which people do not have trouble with. This section characterizes how NNEP can represent these local ambiguities. The question of how to resolve ambiguities when a choice must be made will also be touched on. This test looks at the acceptable example sentences given in the chapters on ambiguity resolution in (Gibson, 1991).[21] The representations necessary for parsing these sentences are presented, along with some discussion of disambiguation strategies. Some ambiguities can be represented in the parser state using partial specification, thereby delaying their resolution until disambiguating information is found. One ambiguity (that of "have") seems to require multiple posthead subcategorization frames to be stored in the parser state. However, the mutual exclusion between frames can be represented using only a conjunction of SUG predications, given the way NNEP interprets SUG predicates. Both the above representations typically require additional resources to maintain the ambiguity, so interactions are predicted between the ability to maintain ambiguities and other resource demands. This prediction has not been investigated due to a lack of available data. There are no mechanisms (other than partial specification) for delaying the resolution of ambiguities which involve portions of a grammar entry which precedes or dominates the grammar entry's word. Some ambiguities which appear to be of this kind can be handled by dividing each of the grammar entries into one lexical component which they share and other nonlexical components which can be added after the lexical component. The ambiguity between NP's and NP's which are the leading edge of unmarked relative clauses is of this later kind. The remaining lexical ambiguities of the former kind are predicted to be resolvable using only left context and (if needed) the subsequent word in the sentence. The information available for this resolution may not be conclusive, in which case a garden path is predicted for one of the alternatives. There is one case which may be a problem for this prediction, but conclusive evidence would require experiments which have not been done.

The work presented in this section is primarily concerned with how to represent the local ambiguities which people do not have trouble maintaining. If the local ambiguity is eventually resolved, then NNEP can detect that there is only one logically possible operation and perform it. However, often the sentence is globally ambiguous, NNEP does not permit the ambiguity to be maintained, or resource limitations force a choice. This requires that NNEP be able choose between logically possible alternatives. As discussed in section 4.5, NNEP is designed to make these choices on the basis of the preferences of other language modules and estimates of the probabilities of the alternatives given the available information about the sentence. Under current assumptions the

[21] The current simulation of the connectionist implementation of NNEP has not been run on the data from these chapters. This is because of the large number of sentences (96 pertinent acceptable sentences), and the time required to handcode the various grammar entries and disambiguation strategies discussed in this section. In future work, I intend to work on automatically training a disambiguation mechanism for NNEP. Such a trained mechanism would presumably subsume all the specific strategies discussed in this section, thus avoiding the time consuming task of handcoding them. For this reason, implementing a simulation which would handle these examples is being postponed until automatic learning mechanisms are investigated.

probability for a combination at a given node with a given grammar entry for the next word is estimated using two independent estimates, one based on the relation of the given node to other nodes in the parser state ($P(x \mid s)$), and one based on information about the node, the grammar entry, the next word, and (if necessary) the part of speech tag(s) of the subsequent word in the input ($P(g, o, w, t' \mid ld(x, s))$). The part of speech tags need to distinguish classes such as finite verbs and words which start NP's. There are many questions to be answered about what information about a node is needed for the second estimate, but the initial hypothesis is that only independently motivated features are necessary. Since the above probabilities have not been estimated, in this section I will offer specific disambiguation strategies which intuitively comply with this more general strategy. The primary concern of these proposals is to determine when NNEP has the evidence it needs to make a decision, and when NNEP needs to be able to represent the ambiguity.

The first local ambiguity discussed in (Gibson, 1991) is between NP complements and S complements. There are two cases of these, one where the S is an infinitival clause and one where it is a finite clause. Examples of both these cases are given in (225).[22]

(225)
a. Bill expected Mary.
b. Bill expected Mary to like John.
c. Bill knew John.
d. Bill knew John liked Mary.

As discussed in section 6.1, infinitival complements with overt subjects are treated as two arguments to the verb, one for the subject and one for the infinitival VP. Thus this case can be handled by treating the infinitival VP like any other optional argument. In both (225a) and (225b) "Mary" is attached as an argument to "expected", as shown on the left in figure 6.6. Then either the sentence is finished or the infinitival VP is attached as a separate argument to "expected". In the case where the S complement is a finite clause, the NP following the verb is either an argument of the verb or an argument of the S complement's verb. A verb like "knew" has a single argument which is underspecified as to whether it is an NP or an S. When the NP following "knew" is input, NNEP can't be sufficiently certain that the NP is the argument to "knew", so the grammar entry for "John" must be added to the end of the list of tree fragments using equationless combining, as shown on the right in figure 6.6. This combination allows for both possible completions. In (225c) this is the end of the sentence, so the NP node for "John" is equated with the argument node for "knew", thus making "John" the object of "knew". In (225d) the sentence is continued with the embedded finite verb, "liked". The grammar entry for "liked" can be combined with "John" and the object for "knew" using double attaching, to produce the desired structure. Thus NNEP's ability to delay attachments allows it to resolve this local ambiguity correctly. Note that the possibility of either completion can be maintained no matter how long the NP is, provided NNEP has enough resources to maintain the ambiguity.

The next set of sentences in (Gibson, 1991) introduce another ambiguity into the above types of sentences. As is illustrated in (228), many words which are prehead constituents of NP's could themselves be heads of NP's.

[22] Except where noted otherwise, the example sentences, their numbering, and the judgments of their acceptability are from (Gibson, 1991).

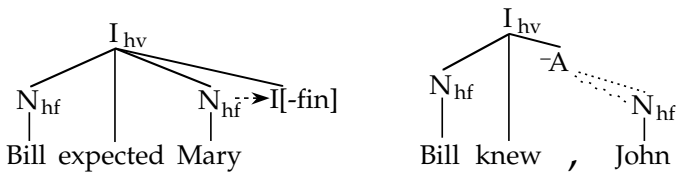


Figure 6.6: NP/S complement ambiguities.

(228)
a. We expected her.
b. We expected her mother to like Fred.
c. The populace believed the American.
d. The populace believed the American president ate broccoli on Tuesdays.

NNEP must be able to recognize whether a word in an NP is the head or not. I take this to be a separate issue from the one just discussed, since those solutions remain unchanged by the solution to this problem. There are two possible solutions to the problem of identifying when a word is the head of an NP. The more straightforward one is that there are both head and nonhead grammar entries for words like "her" or "American", and that one of these grammar entries is chosen by looking at the subsequent word. If the subsequent word could be part of the prehead or head portion of the NP, then the nonhead grammar entry is chosen, and otherwise the head grammar entry is chosen. The alternative is that there is a single grammar entry for the head and nonhead readings of the word, but that the statement of headedness in the grammar entry is noncategorical. As shown in figure 6.7, the grammar entry for "her" would be an NP which has its determiner and has a nonzero nonone probability of having its head. If there are no words following "her" which could be the head of the NP then the nonzero probability that "her" is the head allows it to fulfill the NP's need for a head. Otherwise the nonzero probability that "her" is not the head allows a subsequent word to be the head.[23] The later alternative will be assumed here because it does not require subsequent word lookahead, which will be important below.[24]

The next section in (Gibson, 1991) discusses ambiguities in thematic role assignment to object NP's, illustrated in (233).

(233)
a. I loaded the truck on the ship.
b. I loaded the truck with beans.

Because there is no difference in the Case assignment structures for the different thematic roles, this is not a problem for this parser. Such disambiguation is assumed to occur at a different level of representation.

The sentences in (242) illustrate a case of PP attachment ambiguity.

[23] An alternative to using noncategorical feature specifications in such cases is to distinguish between feature specifications that needs fulfill expectations and feature specifications that restrict iteration. In this case, the former would be specified but the later would not. In SUG, word valued features are used for both purposes, but this and related phenomena seem to indicate that this conflation is not justified. Resolving this question will be left for later work.

[24] This solution introduces the problem of how the parser knows which of the words in an NP which could be the head is the head. I assume that there is a language specific parameter that simply stipulates that the last possible head is the head. This issue is not important to the constituent structure parser, although at some point in processing it needs to be determined.

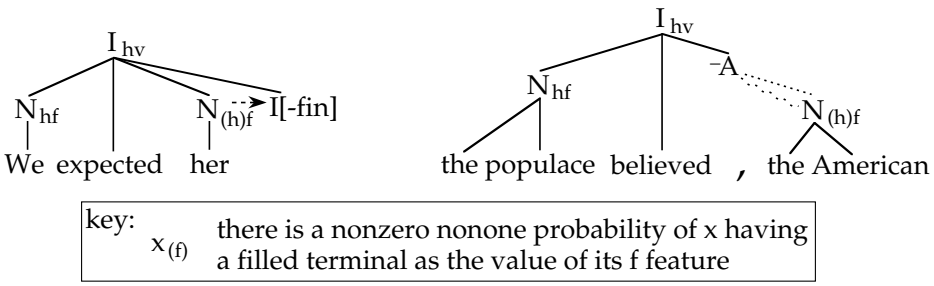


Figure 6.7: Head/prehead ambiguities.

(242)
a. The minister warned the president of the danger.
b. The minister warned the president of the republic of the danger.

The first "of" PP can either attach as an argument of "president" or as an argument of "warned". The resolution of this ambiguity depends on the object of the PP. In (242a) the first PP is an argument of "warned", since people warn of danger and aren't presidents of danger. In (242b) the first PP is an argument of "president", since people are presidents of republics and don't warn of republics. This local ambiguity can be represented by adding the PP to the list of tree fragments without attaching it, as shown in figure 6.8.[25] The object of the PP can then be attached to the PP before the attachment of the PP is chosen. Internal attaching is used to do the chosen PP attachment. The arbitrator which decides what internal equations to do and when to do them can be influenced by the preferences of higher level language modules, as is needed in this example. The calculation of such higher level preferences is not part of the model presented here, but any model of these preferences would have to provide this type of information.

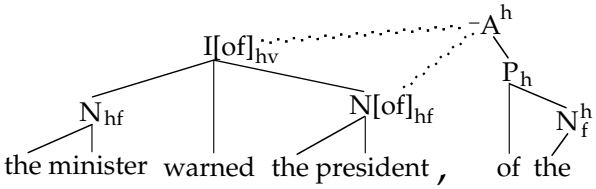


Figure 6.8: A PP attachment ambiguity.

The only sentences in (Gibson, 1991) which are possible counter examples to the constraints on disambiguation being investigated here are the following.

(247a) The bird found in the room was dead.
(249a) The bird found in the room enough debris to build a nest.

Gibson judges both these sentences as acceptable, although he notes that (249a) causes some "slight difficulty". In my informal survey some people said they had no trouble with (249a), but others

[25]The "of" feature shown in figure 6.8 is short for "θ-to-of". As was discussed in section 6.1, the predication "θ-to-of" specifies that this node semantically subcategorizes for a PP headed by "of". This was expressed in a feature rather than with a PP argument node because it simplifies the analysis of long distance movement. Here we see another motivation for this choice, since if there were different grammar entries for argument and adjunct PP's, disambiguation between argument and adjunct attachments would involve disambiguation between grammar entries, rather than just attachment decisions.

garden pathed. Sentences (5.) and (6.) (my examples) illustrate the same problem, but in this case the preference seems to go the other way.

(5.) The bird found yesterday with a broken wing is doing better today.
(6.) The bird found yesterday that someone had destroyed its nest.

The local ambiguity in these sentences arises because "found" can either be the matrix verb or the head of a reduced relative clause. If there is no heavy shift this ambiguity can be resolved by looking at the word following "found". If this word can start an NP or S complement, then the matrix verb grammar entry can be chosen, and otherwise the reduced relative grammar entry can be chosen. This strategy is illustrated in figure 6.9. The strategy works because "find" is obligatorily transitive, and thus if there is no object then it must be a reduced relative. The problem comes when there is heavy shift, such as in the above examples. In these cases the strategy of choosing the reduced relative reading when there is no adjacent object doesn't work, because the object has been shifted across an adjunct. The above strategy will always choose the reduced relative reading in such cases. We can modify the strategy to choose a reading on the basis of whatever available information we like, but any strategy for NNEP must make a choice, and thus NNEP predicts that either (247a) or (249a), and either (5.) or (6.) must produce some difficulty in any given context. At the moment the question of whether this prediction is correct is an open one. Informal informant judgments are not very good at answering such questions because of the difficulty in controlling for context. Thus the evidence discussed above should be taken as only suggestive of a problem. Indeed, Gibson (personal communication) agrees that experiments are needed to determine whether one of the sentences is a garden path. If we take "slight difficulty" as evidence for a garden path in (249a), then the above strategy makes the right prediction. Assuming that (5.) is usually the garden path for the pair (5.),(6.) (as is my intuition), the above strategy makes the wrong prediction. It can be corrected by adding that in the case of a one word adverbial following "found", choose the matrix verb reading. This is justified because one word adverbials are much more likely the participate in heavy shift than other adjuncts, and thus seeing such an adverbial following the verb doesn't tell you much about whether heavy shift has occurred or not. Thus when a one word adverbial is seen following "found", this information doesn't help disambiguate, and the decision must be made on other grounds, such as the frequency of the two grammar entries, which presumably support the matrix verb reading.

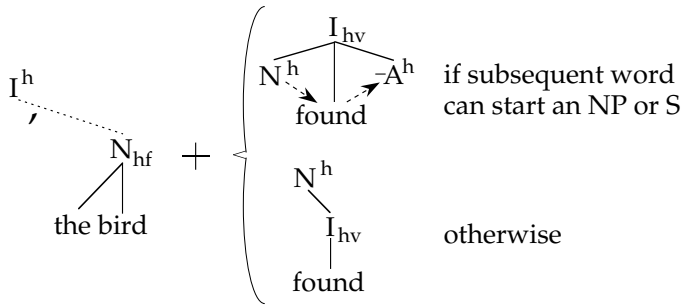


Figure 6.9: A main verb/reduced relative clause ambiguity for obligatorily transitive verbs.

Although at this stage in this investigation the primary concern is the ability of NNEP to represent the ambiguities which do not cause garden paths, the previous discussion points to an interesting prediction about ambiguities which do cause garden paths. If rather than an obligatorily transitive verb like "found", the ambiguous verb in the above examples was often intransitive, then even

without heavy shift a garden path would be predicted. This is the case in the standard example of a garden path, (245a) (originally from (Bever, 1970)). Even without the prepositional phrases, both (245a) and (245b) are predicted to be garden paths, as is the case.[26] This prediction is because for optionally transitive verbs, looking at the subsequent word to see if there is an object present won't help disambiguate between the matrix verb and reduced relative clause readings.

(245a) # The horse raced past the barn fell.
(245b) # The boat floated down the river sank.

The next ambiguity which Gibson discusses is between sentential complements and relative clauses. As is illustrated in (255a) and (256a), when a verb takes both an NP complement and an S complement, an initial segment of an S following the NP complement may be part of the S complement or may be a relative clause modifying the NP.

(255a) John told the man that kissed Mary that Bill saw Phil.
(256a) John told the man that Mary kissed Bill.
(259a) # John told the man that Mary kissed that Bill saw Phil.

Altmann and Steedman (1988) showed that the resolution of this ambiguity is heavily influenced by discourse factors. In a context where no further restrictions are expected on the referent of the NP complement, (256a) is the preferred reading. In contexts where further restrictions are expected, (259a) is the preferred reading. Since Gibson's judgments are done using an empty prior context, he gets the contrast shown. Thus in these cases NNEP simply needs to find what the choices are and allow other modules to provide the disambiguating information. The acceptability of (255a) in the empty context shows that this construction must be parsable despite the contradicting preferences of other modules. Thus this case must be decided within the parsing module itself. The crucial difference between (255a) and (259a) is that the information that the first embedded S is a relative clause immediately follows "that". Thus the question of what grammar entry to use for "that" can be decided on the basis of subsequent word lookahead. If the word following "that" is a finite verb then "that" must begin a relative clause, and otherwise other factors must be taken into consideration to make the decision. The later circumstance is the case in (256a) and (259a), as desired.

The following examples also have an ambiguity between sentential complements and relative clauses, but in this case it seems that the ambiguity can be maintained until the presence or lack of a gap site can be found. This could be because the interpretations of the two alternatives are sufficiently similar that introspection does not always notice the garden path, or it could be because people can delay the resolution of this ambiguity if the two alternatives are equally likely. The discourse factors which can be used to resolve the previous ambiguity cannot be used to determine the relative likelihood of these alternatives.

(262a) The report that the president sent to us helped us make the decision.
(263a) The report that the president sent the troops into combat depressed me.
(266a) The report that motivated our investigation was submitted two days late.

[26] Technically, all that is predicted is that either these sentences or their counterparts (e.g. "The horse raced past the barn") are garden paths for a given listener in a given context, but the preference in this case is clear.

There is a way that the resolution of this ambiguity can be delayed, using the nonlexical grammar entry shown in figure 6.10 and a nonstandard use of the movement mechanism. This grammar entry introduces the structure for a relative clause, and can be used for relative clauses without overt wh- words. The resolution of the ambiguity in (262a) and (263a) can be delayed by adding "that" to the rooted tree fragment list as the complementizer of an S′, and building any further portions of this S′ there, as shown in figure 6.10. If a complete S′ is built, then it can be attached as the complement of "report", which has an optional argument node for this complement. If a gap is found in the S′, then it can be attached to the S′ in the nonlexical grammar entry, using either leftward attaching or double attaching. Just doing this operation will not allow NNEP to find the gap, but since the triggering of the movement rules is compiled into the actions of operation rules, the action for this combination can trigger the movement rules used for internal attaching. These rules will behave as if the S′ for "that" has just been internally equated with the S′ for the relative clause, and therefore will find the gap site in the rightmost tree fragment.

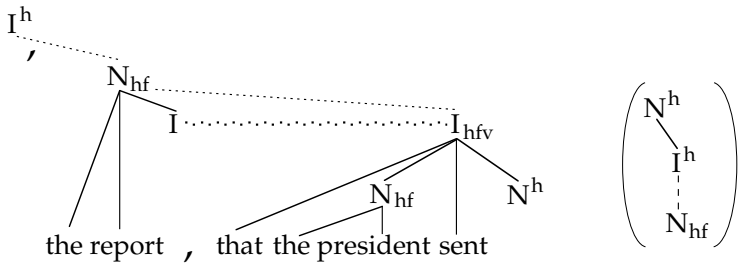


Figure 6.10: A sentential complement/relative clause ambiguity.

The remaining examples in chapter 6 of (Gibson, 1991) are concerned with lexical ambiguity. There are two strategies which NNEP can employ for handling these ambiguities. NNEP can either guess on the basis of the left context and subsequent word, or NNEP can use a representation which is ambiguous in the same way as the word. The later strategy may employ nonlexical grammar entries to express the differences between the readings. The addition of nonlexical grammar entries to the memory can be delayed, unlike the addition of lexical grammar entries. When NNEP picks between grammar entries, it does so on the basis of preferences from other language modules and estimates of the probabilities of the alternatives given the available information about the sentence, as was discussed in section 4.5. In the following discussion I will offer specific disambiguation strategies which intuitively comply with this more general statistically based method, as was done above for "found" for sentences (247a) and (249a). The importance of these proposals is not that they provide evidence for the statistically based method, but that they demonstrate that the information which is needed for disambiguation is present at the point at which disambiguation decisions have to be made.

Most of the noun/verb ambiguities discussed in (Gibson, 1991) are for verbs which take posthead complements. As for "found" above, this permits a strategy where if the subsequent word could begin a complement to the verb then the verb reading is chosen, and otherwise the noun reading is chosen. For the examples given, the alternative strategy of choosing the noun reading when the subsequent word is a finite verb and choosing the verb reading otherwise would also be possible (since all these nouns are in subject position). The ambiguities which can be handled with either of these strategies are given in (273), (276), (277), and (278).[27] Since "lies" in (275) does not take

[27] As in the sentences in (228), the problem of determining whether the word preceding the noun/verb ambiguous word ("warehouse" in (273), etc.) is the head of its phrase or not is handled by giving that word a nonzero nonone

a complement in its verb reading, only the later of these strategies will work for this ambiguity.[28] These strategies are illustrated in figure 6.11.

(273)
a. The warehouse fires numerous employees each year.
b. The warehouse fires harm some employees each year.
(276)
a. The American places that I like to visit most are in California.
b. The American places the book on the table.
(277)
a. The paint can be applied easily with a new brush.
b. The paint can fell down the stairs.
(278)
a. The building blocks are red.
b. The building blocks the sun.
(275)
a. The official lies are very transparent.
b. The official lies often.

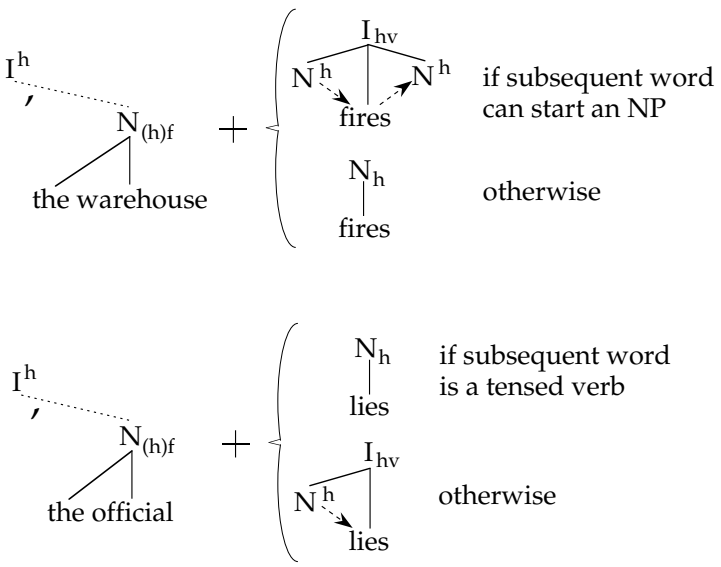


Figure 6.11: Noun/verb ambiguities.

As mentioned above, the lexical ambiguity between different uses of "have" seems to require NNEP to simultaneously represent multiple posthead subcategorization frames for a single lexical item. Nodes for both these frames can be added to the parser state, but the mutual exclusion between these nodes still has to be represented. There are two possible ways of representing this mutual exclusion. The first is to simply say "it's not my problem". The two subcategorization frames are for mutually exclusive predicate-argument structures, so the module which calculates predicate-argument structure will rule out the case where positions in both frames are filled. Thus it is

probability of being the head of its NP. Note that unlike in (228), a subsequent word lookahead would not be sufficient for choosing between multiple grammar entries for these two possibilities in this case.

[28]Since these strategies make use of subsequent word lookahead, they are susceptible to a number of problems, but the data on these issues is not clear.

not necessary for the constituent structure parser to enforce this constraint. Expressing mutual exclusion at the predicate-argument structure level is necessary anyway to express the mutual exclusion between some NP arguments and PP arguments (as for "give"), since the subcategorization for a PP argument is represented in the predicate-argument structure and not in the constituent structure. While it is unappealing to place responsibility for this phenomena on another module, this would allow this phenomena to be subsumed under a more general mechanism.

The other alternative is to represent the mutual exclusion between the nodes in the two subcategorization frames directly in the parser state. Given the way NNEP interprets the predicates of SUG, no additional predicate is necessary to express this mutual exclusion. If two nodes are mutually exclusive, this means they can't both dominate terminals. NNEP only checks for violations of linear precedence constraints on terminals. If there is an inconsistency between linear precedence constraints on two nonterminals, this will always show up as a violation on terminals, provided both nonterminals dominate terminals. If, however, one of the two nonterminals does not dominate any terminals, then the violation will not be detected by NNEP. Thus mutual exclusion can be expressed by having the two nonterminals linear precede each other. If two nonterminals linearly precede each other, then one but not both nonterminals can ever dominate terminals, as desired.[29] At the moment this looks like a hack, but it would be an easy matter to redefine linear precedence in SUG so that only constraints between terminals can result in inconsistency, thus making it a perfectly legitimate constraint to express in an SUG grammar. It also doesn't violate the determinism constraint, as it has been defined here. At the level of the constituent structure parser, the structural information it stores is still a conjunction of predications which grows monotonically, and all of which is part of the output. The relationship between this constituent structure and predicate-argument structure does contain disjunction, since two mutually exclusive nodes belong to two distinct predicate-argument structures, but there are other cases where this is also assumed to be true. It could be that both this mechanism and the previous one are used, and it could even be that a better analysis of "have" would eliminate this problem. Since the only data I currently have on the need for mutually exclusive subcategorized arguments is that for "have", it is hard to distinguish between these possibilities. Given that this phenomena is rare, it is better to explain it with an unusual use of existing mechanisms (such as incompatible ordering constraints) than to introduce new mechanisms to handle it.

The ambiguity between the main and auxiliary verb readings of "have", appears to require the representation of multiple posthead subcategorization frames. Gibson discusses (282a) and (282b), but the data I'm responsible for from the Brown corpus further complicates the analysis. All this data is shown below, split into the main verb and the auxiliary verb cases. The analysis of "have" involves many issues which I am not terribly knowledgable about, but an analysis will be presented which at least covers the data I am aware of.

(282a) Have the boys take the exam.
1248: Mr. D'Albert has a firm, attractive tone, which eschews an overly sweet vibrato.
1269: So, what was the deepest music on her program had the poorest showing.
(7.) Have the boys taken to school.
(8.) Have the boys clean.

[29] Although this use of precedence constraints results in two nodes which are preceded by other nodes, it does not necessarily violate NNEP's constraints on the use of linear precedence. As discussed in chapter 4, NNEP needs to be able to represent linear precedence constraints with two unary predicates, *preceding* and *preceded*. If two nodes linearly precede each other, then they both are both *preceding* and *preceded*. If any other nodes are preceded by either of these nodes, then this constraint requires that it precede both these nodes. For the one case where this mechanism is used here, this is in fact the case.

(9.) Have the boys in bed.

(282b) Have the boys taken the exam?
988: The famed Yankee Clipper, now retired, has been assisting as a batting coach.
1062: Allied Arts had booked Marlene Dietrich into McCormick Place Dec. 8 and 9.
1113: One of those capital-gains ventures, in fact, has saddled him with Gore Court.
1129: Cherkasov does not caricature him, as some actors have been inclined to do.

Figure 6.12 shows the three grammar entries used in the analysis for "have" proposed here. All the main verb cases can be handled with the grammar entry shown on the left. The subject is optional, to allow for (282a) and (7.) through (9.). The second object also doesn't expect anything, to allow for 1248 and 1269. This second argument could be an N or an I, and already has its head.[30] In (282a) this node would be equated with the I from "take", and the head would satisfy "take"'s need for inflection. In (7.) through (9.) (my examples), the second object node would equate with the N which "taken", "clean", and "in" modify, and the head would satisfy their need for a noun. In other words, "taken", "clean", and "in" would modify the second argument as if it were a normal NP.[31] This type of analysis will also be used for the copula, below. The middle grammar entry shown in figure 6.12 can handle all the auxiliary verb cases. The subject N can be either before or after the verb, so both the question and declarative forms are covered. Assuming the "-en" affix shows up as an "-ed" affix on some verbs, the object I covers all of (282b), 988, 1062, 1113, and 1129. The rightmost grammar entry in figure 6.12 represents the ambiguity we are concerned with. This grammar entry would have to be used in (282a), (282b), and (7.) through (9.), since at the point when "have" is encountered there is insufficient evidence to choose between the main verb and the auxiliary verb cases. In all the other cases the subsequent word is sufficient evidence to disambiguate. The subsequent word would also be sufficient if the NP after "have" were marked for case, since in this situation the case would be on the first (and only) word in the NP. If the first NP is ambiguous as to its case, then the rightmost grammar entry needs to be used. The NP is attached to the first N argument, and then the remainder of the sentence can be attached to either, but not both, of the other argument nodes. In the case of (282b) and (7.) there is a further ambiguity. This is the same reduced relative/main verb ambiguity discussed for "found" above, and since the main verb "taken" is also very likely to be transitive, it can be disambiguated as in that case, using subsequent word lookahead. While it is unappealing to have this additional grammar entry just for dealing with this one case of ambiguity, at least it has been shown that NNEP does have the ability to handle it. Perhaps a more extensive investigation of auxiliaries would lead to a better analysis.

Gibson gives (282c) and (282d) as examples of the same phenomena as (282a) and (282b), but under the analysis of the copula being used here, these examples are simply an ambiguity between modifier and argument attachments. Under this analysis "is" takes two arguments, a subject NP, and an object NP which has a nonzero nonone probability of already being headed. The "in" PP

[30] The need to specify this generalization about this argument is the reason inflection is being used as the head of a verb projection rather than the verb. If this is an N argument then it needs to already have its head noun, and if it is an I argument then it needs to already have its inflection. As will be discussed below, the need to represent the different uses of "that" in a single grammar entry forces *functional_head* to be used for both having a determiner and having a complementizer. Thus this set of generalizations of terminal valued features across categories is the only one allowed by the data considered here. The need for descriptions to be partially specified in the ways which express the required local ambiguities is the central consideration for determining the proper generalizations in the language for specifying constituent structure. This is in sharp contrast to the motivations for such generalizations in competence linguistics, which is not concerned with ambiguity.

[31] This version of "taken" is the head of a reduced relative clause.

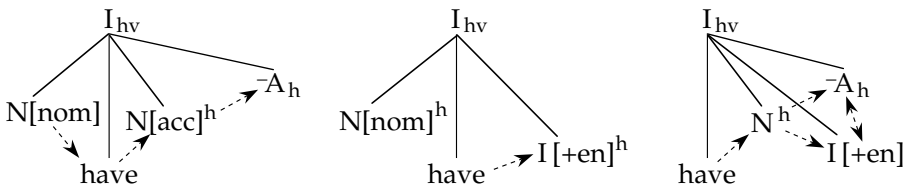


Figure 6.12: Grammar entries for "have".

can either modify the subject NP or the object NP. If it modifies the object NP, then that NP gets its headedness from "is". The resolution of the ambiguity can be delayed by adding the "in" PP to the tree fragment list, as shown in figure 6.13. If the next word can attach to the object NP of "is", then the "in" PP attaches to the subject NP. If the sentence ends without any other possible object, then the "in" PP attaches to the object NP.[32]

(282)
c. Is the crowd in the room yet?
d. Is the crowd in the room happy?

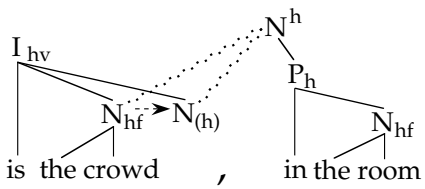


Figure 6.13: The copula and a modifier/argument ambiguity.

The next lexical ambiguity in (Gibson, 1991) is for the word "that". Strange as it may seem, NNEP can express the essential information for all uses of "that" in a single grammar entry. "That" is the *functional_head* (determiner or complementizer) of a −A (N or I) node which has a nonzero nonone probability of already being headed, as shown in figure 6.14. In (285a) "that" is both the functional head and the head of the object of "know". In (285b) "that" is the determiner of "that boy" but not the head. In (285c) "that" is the complementizer but not the head of "that boys should do it". Since this grammar entry makes "that" lexically unambiguous, this problem is reduced to an attachment ambiguity, which can be handled by delaying the attachment using the tree fragment list. In order to maintain the analysis of the that-trace effect given in section 6.2, the complete grammar entry for "that" also needs a node for the anticipated subject, but this node can be specified as equatable with the other node, thus still allowing the determiner and pronoun alternatives.

(285)
a. I know that.
b. I know that boy should do it.
c. I know that boys should do it.

[32]The exact mechanism to control this type of indirect attachment decision has not yet been designed. There are several ways that it could be done, but the issues involved are beyond the scope of this work. This issue must be investigated in the context of a theory of how disambiguation decisions are made, which is not addresses at a sufficient level of detail in this work.

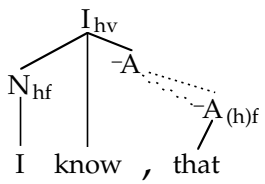


Figure 6.14: A vague grammar entry for "that".

The last ambiguity discussed in chapter 6 of (Gibson, 1991) is for the word "to". As shown in (292), "to" can either be a preposition or supply inflection for infinitival sentences. Infinitival sentences can either be complements or purpose adjuncts. For the sentences in (292) this lexical ambiguity can be resolved on the basis of subsequent word lookahead, as shown in figure 6.15. If the subsequent word is an untensed verb, then "to" is an inflection marker, and otherwise it is a preposition. To test this hypothesis we need to look at sentences like those given in (10.) and (11.) (my examples), where the subsequent word is ambiguous between an untensed verb and a noun. If for all such pairs one of the two sentences is difficult in any given context, then the above strategy can be maintained. Otherwise we need an analysis where "to" has the same grammar entry in both cases. Since an informal test of these sentences suggests that (10.) and similar examples are difficult to understand on first hearing, it appears that the subsequent word lookahead analysis can be maintained. Further investigation of this question is needed.[33] The ambiguity between infinitival sentences which are complements and those which are purpose adjuncts can be handled with a nonlexical grammar entry which expresses the modification relationship between sentences and infinitival sentences, which is shown in parenthesis in figure 6.15.

(292)
a. I opened the letter to Mary.
b. I opened the letter to impress Mary.
c. I shouted to mow the lawn to Bob.
d. I shouted to Bob to mow the lawn.

(10.)
a. ?John walked a mile to water where he could fish.
b. ?John added the grain to feed for his cow.
(11.)
a. John walked a mile to water his friend's plants.
b. John added the grain to feed all his cows.

Chapter 7 of (Gibson, 1991) is primarily concerned with attachment ambiguities. For NNEP there are no problems delaying these decisions if it is necessary, so the primary question is how the statistically based method discussed in section 4.5 can be used to make the right disambiguation decisions. Since any substantive comments on this question would require training and testing this

[33] A technical memorandum by Hindle called *Disambiguation "to"* (March 1988, AT&T) investigated this question using the Brown Corpus. Using a deterministic structure building parser and a few words lookahead (usually one), he was able to disambiguate 99% of the 26000 tokens of "to" in the Brown Corpus. Since the strategy presented here is of the same type, Hindle's success suggests that the strategy presented here can work. As with all such techniques, it is assumed here that other modules will help in disambiguation, and thus some errors are expected in a system which does not have this additional input.

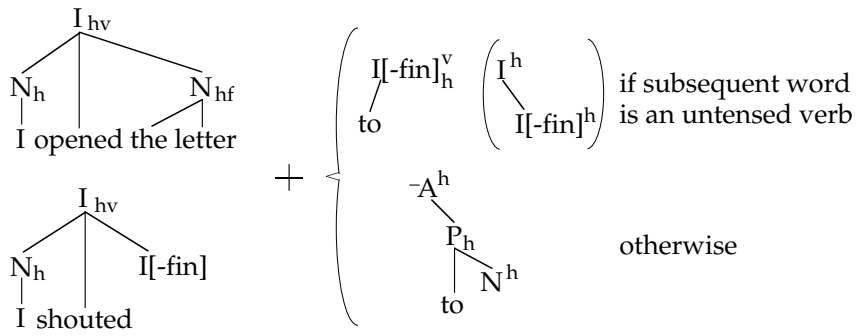


Figure 6.15: Disambiguating "to".

mechanism, and such work is outside the scope of this dissertation, these examples will not be discussed here.

Section 7.3 of (Gibson, 1991) discusses some acceptable sentences which involve more than simple attachment decisions. These sentences are given in (337), below. In these sentences "earrings" and "boxes" might be part of the preceding NP or might be a separate NP. The resolution of this ambiguity can be delayed using mechanisms already discussed. First, "her" and "company" have a nonzero nonone probability of being the heads of their NP's. This allows "earrings" and "boxes" to either be the heads of these same NP's or not. Second, the list of tree fragments can be used to delay the decision of which argument "her" and "company" are a part of. The decision of whether to attach "earrings" and "boxes" to the preceding NP or to the second NP argument of the verb can be done using one word lookahead.

(337)
a. I gave her earrings on her birthday.
b. I gave her earrings to Sally.
c. John sent his company boxes for storage.
d. John sent his company boxes to the recycling plant.

To conclude, NNEP has several mechanisms which allow it to handle locally ambiguous sentences. Central to the success of these mechanisms is the extensive use of partial descriptions to allow information which the parser is not yet sure about to be left unspecified. One crucial mechanism is the list of tree fragments, which allows attachment decisions to be delayed. Another is the use of nonlexical grammar entries. This allows the addition of some information to be delayed, rather than having to introduce it at the same time as the grammar entry for some word. The use of probabilistic information allows the specification of attachment preferences, plus it allows partial commitment to predications such as the headedness of a node. Because NNEP only enforces ordering constraints on terminals, incompatible ordering constraints can be used to state that only one of two nonterminals can dominate terminals. This allows NNEP to represent mutually exclusive posthead subcategorization frames by adding the nodes for both the frames and stating their mutual exclusion with incompatible ordering constraints. This mechanism is rather resource intensive, but it can be used sparingly to delay resolution of lexical ambiguities which have only posthead differences. When necessary NNEP can look at the subsequent word to make lexical disambiguation decisions. This lookahead mechanism is prone to error, so it should be used sparingly. Although some of the necessary data is not clear at the moment, it appears that this set of mechanisms is adequate for representing the local ambiguities which people do not have trouble with, until they can be resolved.

6.4 Handling the Parser's Resource Bounds

Because various of NNEP's computational resources are bounded, it is important to test it on sentences which might require the violation of these bounds. This section characterizes how NNEP can handle the bounds on its resources, and discusses their linguistic implications. At any one time, NNEP can only store at most ten nonterminals, two nodes on the public node stack, and four tree fragments. The constraint on the number of instantiations of binary predications that can be stored is subsumed by the later two constraints. There is also a constraint on the representation of linear precedence constraints. The implications and lack of implications of these bounds are investigated using the example sentences given in the chapters on processing overload in (Gibson, 1991).[34] Unfortunately, (Gibson, 1991) does not include examples which illustrate people's limited ability to attach modifiers to the right frontier of a long right branching sentence, and I know of no source for such examples. This limits the investigation of the implications of the bound on the number of nonterminals. However, as discussed in section 4.1.3, theories of restrictions on such modification are compatible with this bound. Also, the examples addressed here do provide good evidence that this bound does not need to be violated in other cases, and they do address the issues pertinent to the bounds on NNEP's public node stack and tree fragment list. These later bounds are consistent with the acceptable sentences, and account for many of the unacceptable sentences.

As discussed in chapter 4, the bounds on NNEP's resources are motivated by the limitations of the S&A architecture. Biological constraints restrict the number of variables to at very most ten. In NNEP variables are used to refer to nonterminal nodes, so it can store information about at most ten nonterminal nodes at a time. The public node stack and the tree fragment list are implemented using predicates to distinguish between their different positions and rules which manipulate those predicates appropriately as their configurations change. The maximum depth of the stack and the maximum length of the list are determined by the number of different positions which can be distinguished using the predicates. This means these sizes have to be bounded, since a given implementation has a fixed number of predicates. The actual bounds can be arbitrarily large, since an arbitrary number of predicates can be dedicated to this use. However, there is a natural assumption to make about the number of predicates which are used. There are some predicates which are needed for rules other than those used to manipulate the configurations of the stack and list. Since these predicates are independently motivated, it is natural to assume that these predicates are available for the rules which manipulate the configurations of the stack and list. It is also natural to assume that these are the only predicates used for this purpose. Any additional predicates would only be needed when the stack or list was beyond its otherwise maximum size, and the circumstances in which this added size would be useful seem to be rather rare. If we assume that all and only the independently motivated predicates are used in manipulating the stack and the list, then their maximum sizes are the ones used for NNEP. The list of tree fragments can be four long and the public node stack can be two deep.

The independently motivated predicates which NNEP can use to represent the list of tree fragments are *right_fragment, adjacent_fragment,* and *matrix_fragment. Right_fragment* is needed for virtually all NNEP's operations, *adjacent_fragment* is needed for internal attaching and sometimes

[34] As for the other sentences from (Gibson, 1991), the current simulation of the connectionist implementation of NNEP has not been run on the data from these chapters. This is because of the large number of sentences (55 acceptable sentences), and the time required to handcode the various grammar entries. Also, because the simulation's disambiguation mechanism does not have the statistical information it needs to make effective decisions, it uses more resources than are necessary to avoid making these decisions. This would require a lot of time consuming manual intervention into disambiguation decisions in order to test the resource requirements of the underlying model.

for double attaching, and *matrix_fragment* is needed to prevent leftward attaching from applying to the matrix root. These predicates allow NNEP to distinguish between four positions on the list of tree fragments, three for the three predicates, plus one for all the remaining nodes in the parser state. Thus the list of tree fragments can be at most four deep before something goes wrong.

For the public node stack, NNEP needs to know what node is the public node. In the case of trace nodes, this results in a maximum stack depth of two, since only trace nodes are neither *parented* nor *anchored*, so the non-public trace node on the stack can always be determined. In the case of tree fragment roots, the description of NNEP given in the previous chapters doesn't provide any predicates which distinguish those which are on the public node stack from those which are not (other than for the public node itself). However, as was mentioned in chapter 4, the constraint on the storage of binary relations requires that the binary relations for nodes on the public node stack and the binary relations for other tree fragment roots need to be stored in separate predicates. Thus the tree fragment roots on the public node stack can be distinguished from the other tree fragment roots by which binary predicates refer to them. Thus also in this case, the public node stack can be two deep, one position for the public node and the other for the other node referred to by the public node's binary predicates. This division of the binary predicates also means that once a tree fragment root is placed on the public node stack, it can't be dropped from it. If it were dropped, it would lose all the information stored in the binary relations.

The constraint necessary for implementing the inheritance of linear precedence relationships is not stressed in this chapter because the data on this issue is rather limited. Gibson (1991) does not discuss any data which is pertinent to this issue, but there is only one pertinent case, and I have devised some sentences to illustrate the phenomena. Unfortunately the judgments in this case are not entirely clear. Partly for this reason and partly because the data is so limited, there are several possible analyses which I have not been able to tease apart. The possibility I am assuming is that at any one time linear precedence constraints are only represented for one set of nonterminals which are preceded by other nonterminals and do not dominate any words. Other constraints, such as a stack, are possible, but these options make more assumptions about the cases in which these nodes can occur. For the purposes of this investigation I will stick with the simpler constraint.

The examples which are pertinent to this last constraint are given in (12.) through (14.) (my examples). Ordering constraints with terminals are represented with predicates and by restricting the applicability of grammar entries, so only ordering constraints between nonterminals are actually represented as linear precedence relationships. In addition, once a nonterminal is no longer on the right frontier, ordering constraints on it are no longer important. Thus the only linear precedence relationships which need to be represented are those between two nodes which are both on the right frontier. Such constraints only occur in the case of ditransitive verbs. Sentences (12.), (13.), and (14.) contain one, two, and three ditransitive verbs, respectively. Sentence (12.) is fine, but (13.) is surprisingly difficult, and (14.) is unacceptable. A state in the parse of (13.) is shown in figure 6.16. After the second ditransitive verb, the parser state must hold two nodes which have linear precedence constraints on them, which is too many for NNEP to process correctly under the constraint being assumed here. This may explain the unnaturalness of this sentence. On the other hand this sentence is not completely out. This may be because the parser can simply throw away the first node's linear precedence constraints and still parse the sentence effectively, using a recency preference or something equivalent. This may be the right explanation for these phenomena, but on the other hand the data is so limited and unclear that any number of other factors could explain the pattern. In any case, the complete unacceptability of (14.) shows that the number of nodes which NNEP needs to represent and inherit linear precedence relationships for is at most two. The limited situation in which these nodes occur means that several possible strategies are possible for

processing these (at most) two nodes within the locality constraint on rules.

(12.) John gave the man money to buy roses with.
(13.) ? John gave the man who bought the woman roses money to buy them with.
(14.) # John gave the man who bought the woman that told Mary stories roses money to buy them with.

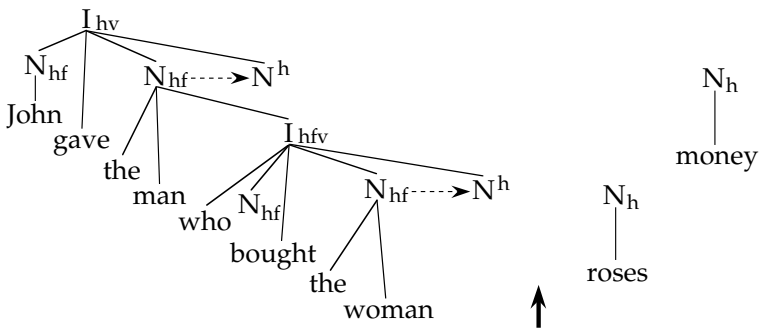


Figure 6.16: A sentence with two nodes which are preceded by other nonterminals and do not dominate terminals.

NNEP can represent four tree fragments at a time. The roots of these tree fragments can be on the public node stack or not, but trace nodes are not treated as tree fragments. Trace nodes can move across lexical material, and thus are not subject to the same constraints as nodes which dominate words. The strategy NNEP uses for when to specify a tree fragment root as the public node will be discussed below, but it is not pertinent to these examples. In order to isolate the effects of the tree fragment stack from the effects of the public node stack, the following examples use nominal complements rather than the relative clauses often found in examples of center embedding. (370a), (377a), and (393a) each have four tree fragments just before the first verb is encountered ((370a) originally from (Cowper, 1976)). (370a), and (393a) are acceptable, as predicted. The pertinent point in the parsing of (370a) is shown in figure 6.17. (393a) is similar to (391b), which is shown in figure 6.21. Gibson reports that (377a) is acceptable to many readers. The difficulty this sentence causes for other readers may be due to additional resources necessary to represent the ambiguity between the complement and relative clause possibilities for the most deeply embedded sentence. The other two sentences give the parser time to resolve this ambiguity before the maximum number of tree fragments are needed. As shown in figure 6.18, NNEP can parse (377a) because (if properly disambiguated) it only requires four tree fragments. (15.), (16.), and (394a) require five tree fragments, and they are definitely unacceptable, as is predicted by this constraint ((15.) and (16.) are my examples).

(370a) The possibility that the man who I hired is incompetent worries me.
(377a) ?# The fact that the idea that John is gay would upset his parents disturbs me.
(393a) What the rumor that the accused man had robbed a bank influenced was the judge's decision.

(15.) # The possibility that the fact that the man who I hired lied was covered up worries me.
(16.) # The possibility that the fact that the rumor that the nominee is gay would cause controversy had scuttled the nomination disturbs me.
(394a) # What the information that the weapons that the government built didn't work properly affected was the enemy's defense strategy.

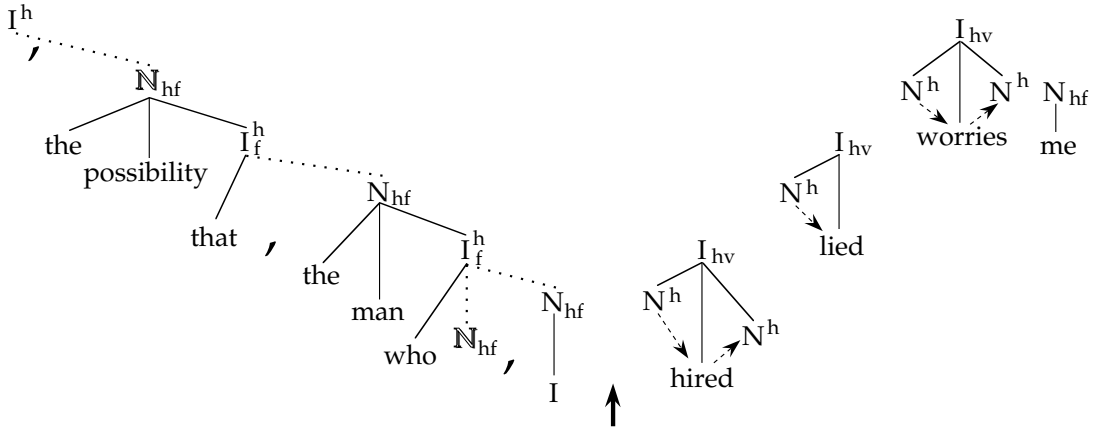


Figure 6.17: A sentence with four tree fragments and two stacked nodes.

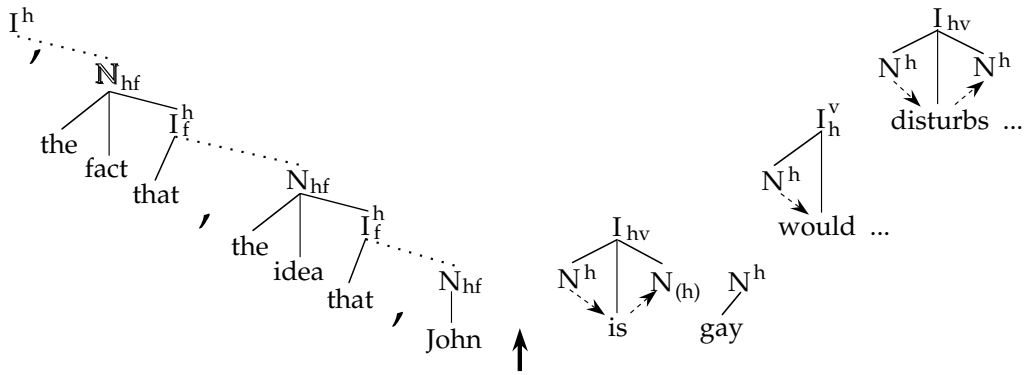


Figure 6.18: The parse of (377a).

Before the bound on the depth of the public node stack can be discussed, a strategy needs to be chosen for when to specify the roots of tree fragments as the public node. If a node is specified as the public node, then double attaching can be used, instead of leftward attaching and then internal attaching. On the other hand, only two nodes can be stored on the public node stack at one time, so this resource should be conserved. NNEP uses a compromise strategy, where a tree fragment root is specified as the public node only if there is no public node at the time. Thus a sentence initial subject will always be specified as the public node, unless the first word of that subject introduces a trace node. Trace nodes must always be on the public node stack, regardless of whether there are other nodes already on this stack.

The fact that the above strategy specifies sentence initial subjects as the public node is what explains the standard contrast in center embedded sentences, illustrated in (349), (342a), and (342b). As shown in the parse state for (342a) in figure 6.19, the first subject is specified as the public node because at that point in the parse there is no public node. The first relative clause then introduces a trace node, which must be on the public node stack. When the second relative clause is encountered, that trace node must also be put on the public node stack, but the stack is already full, so the sentence is predicted to be unacceptable, as desired. Sentence (370a) demonstrates that this problem goes away when one of the relative clauses is replaced with a nominative complement, as shown above in figure 6.17. In this example the subject is still specified as the public node, but since only one trace node is subsequently introduced, the sentence is acceptable.

(349) The man that Mary likes eats fish.
(342a) # The man that the woman that won the race likes eats fish.
(342b) # The man that the woman that the dog bit likes eats fish.
(370a) The possibility that the man who I hired is incompetent worries me.

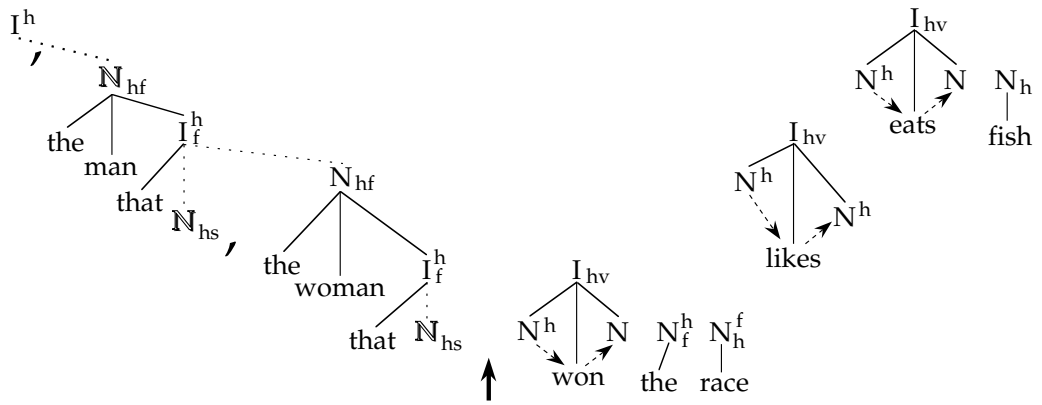


Figure 6.19: A parse with one too many stacked nodes.

Sentences (351a) and (351b) (originally from (Eady and Fodor, 1981)) illustrate that the previous effect is not just due to the number of sentence embeddings. These sentences are the same as (342a) and (342b), except "the man" gets its immediate parent before the first relative clause. When the first pertinent subject is encountered, a trace node has already been introduced by the first relative clause, so the subject is not specified as the public node. Since the two trace nodes fit on the stack by themselves, there are no constraint violations. This is shown in the parser state for (351b) given in figure 6.20. Note that this parser state has the same number of subjects and trace nodes as shown in figure 6.19 for (342a), but since they are introduced in a different order, different predictions are made.

(351a) I saw the man that the woman that won the race likes.
(351b) I saw the man that the woman that the dog bit likes.

The correlation between the acceptabilities of (17.) through (19.) and (20.) through (22.) (my examples) further illustrate the parallel between subjects which are introduced before any trace nodes and trace nodes themselves. In (17.) through (19.) "the man" is introduced before any trace nodes, so it is specified as the public node. In (20.) through (22.) "what" introduces a trace node before "the man" is encountered. This trace node fills the same position on the stack as "the man" in (17.) through (19.), so the pattern of acceptability is predicted to be the same, as desired.

(17.) I said that the man that the woman likes eats fish.
(18.) # I said that the man that the woman that won the race likes eats fish.
(19.) # I said that the man that the woman that the dog bit likes eats fish.
(20) I wondered what the man that the woman likes eats.
(21.) # I wondered what the man that the woman that won the race likes eats.
(22.) # I wondered what the man that the woman that the dog bit likes eats.

Wh- questions pattern the same as the indirect question in (20.) through (22.), as shown in (23.) (my example), (381a), and (381b), and their resource usage is also the same.

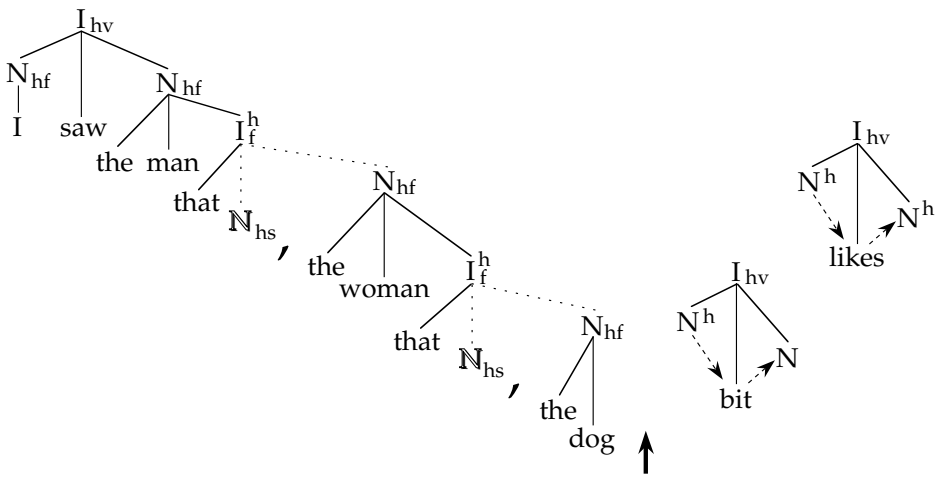


Figure 6.20: The parse of (351b).

(23.) Who did John donate the furniture that the repairman found to?
(381a) # Who did John donate the furniture that the repairman that the dog bit found to?
(381b) # Who did the salesman distribute the leaflets that the secretary that works efficiently printed to?

Pseudo-clefts are a particularly interesting test of the strategy for when to specify a node as the public node, since the first word of the sentence initial subject introduces a trace node. As shown in figure 6.21 for (391b), the strategy does not specify the subject as the public node in these circumstances. This results in a parser state in which both the public node stack and the tree fragment list are completely full, but not overflowing. Sentence (394a) illustrates that one more subject will cause the tree fragment list to overflow, and (24.) (my example) illustrates that one more trace node will cause the public node stack to overflow.

(391b) What the woman that John married likes is smoked salmon.
(394a) # What the information that the weapons that the government built didn't work properly affected was the enemy's defense strategy.
(24.) # What the woman that what was done pleased likes is smoked salmon.

The bound on the number of nonterminals which can be stored in the parser state is not necessary to account for any of the unacceptable sentences in the processing overload sections of (Gibson, 1991), but it also does not interfere with the parsing of any of the acceptable sentences. The sentence which requires the most nonterminals is (391b). Just after the first verb ("married") is combined with the parser state shown in figure 6.21, there are ten nonterminal nodes. This is the most nodes which might need to be stored in parsing the example sentences from (Gibson, 1991). Immediately after or even during this combination, the node for "John" can be forgotten, since it is then not on the right frontier. Thus NNEP needs to store nine nodes while it is computing an operation. This is the most nodes which need to be stored during a computation for any of the example sentences from (Gibson, 1991). If there were examples of sentences which have a large right frontier, followed by a modifier, argument PP, or particle, then the effects of this memory bound would come into play. This bound would restrict how many nodes on the right frontier would be available for such phrases to attach to. As was discussed in section 4.1.3, the available evidence is compatible with

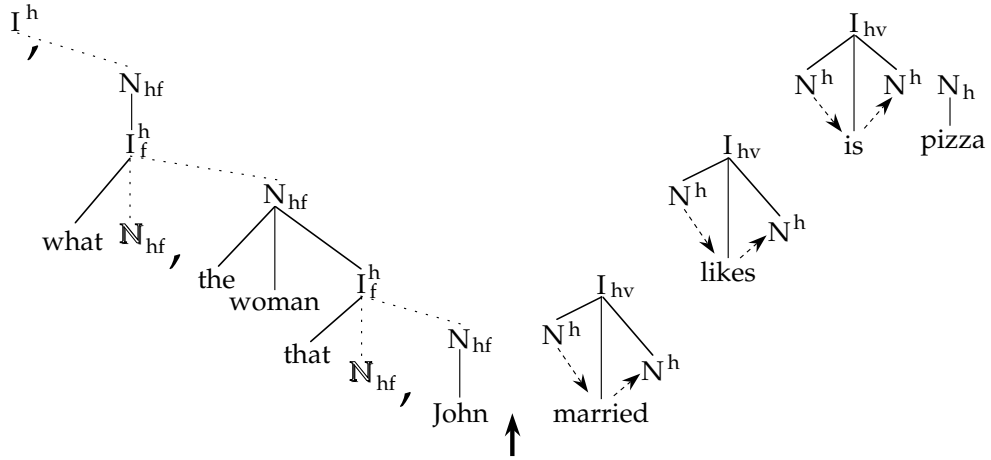


Figure 6.21: The parse of (391b).

this restriction. Further investigation of this issue will have to wait until an appropriate set of data can be found.

NNEP can parse all the acceptable sentences given in the processing overload sections of (Gibson, 1991), but as mentioned above, this model does not account for all unacceptable center embedded sentences. Since the primary objective of this dissertation is demonstrating the computational adequacy of the S&A architecture, the emphasis of the investigation has been on accounting for acceptable sentences. Also, there are sources of unacceptability which have not been investigated here, and thus there is expected to be unacceptable data which is not accounted for. One such source of unacceptability is the resource demands of delaying the resolution of ambiguities. Such an argument was suggested above for explaining why some people find (377a) unacceptable. This same argument may apply to (372a) (originally from (Cowper, 1976)), which Gibson judges to be worse than (370a). Delaying the resolution of the ambiguity between a nominal complement analysis and a relative clause analysis for the sentence following "possibility" may account for the difficulty. A similar argument may apply to (383a), which this model predicts to be no worse than (370a), and to (388a), which this model predicts to be no worse than (391b). Of course, there is also always the possibility that the judgments are incorrect, but that possibility cuts both ways. Sentence (410a) is also predicted to be no worse than (391b), possibly because the wrong analysis of this construction is being used here. In addition to these cases which are similar to the ones which are accounted for, there are a whole set of sentences given as unacceptable in (Gibson, 1991) and not accounted for here, which involve constructions with "that that" or "that for". In these constructions "that" is a complementizer and "for" gives Case to the subject of an infinitival sentence. Sentences (356a) and (356c) illustrate this phenomena ((356a) originally from (Kimball, 1973)). Following (Koster, 1978), Lewis (1993) proposed that these cases are ruled out by a competence constraint on sentential subjects, but Gibson (1991) points out some problems with this approach.

(372a) # The man who the possibility that students are dangerous frightens is nice.
(383a) # Who does the information that the weapons that the government built don't work properly affect most?
(388a) # It was the enemy's defense strategy that the information that the weapons that the government built didn't work properly affected.

(401a) # Surprising though the information that the weapons that the government built didn't work properly was, no one took advantage of the mistakes.
(356a) # That that Joe left bothered Susan surprised Max.
(356c) # That for Joe to smoke would annoy me is obvious.

Despite the fact that not all unacceptable examples of center embedding have been accounted for by this model, an interesting and substantial subset of them can be explained in terms of the constraints imposed by the mechanisms which NNEP needs to compensate for the computational limitations of the S&A architecture.

6.5 The Diversity of Language

Because there is no formal proof that the specific phenomena discussed above completely cover the problems that the limitations of the Shastri and Ajjanagadde architecture pose for constituent structure parsing, it is necessary to check for errors in the analysis by testing NNEP's ability to handle the diversity of phenomena found in natural language. This section discusses such a test. The test uses a set of 50 thirteen word sentences which were randomly selected from the Brown corpus by Ezra Black in 1991.[35] Exceptions to this test were only allowed for phenomena which have been explicitly excluded from the phenomena the parser is intended to cover. In particular, NNEP can not parse coordinations, or gaping in comparatives, and there are many disambiguation decisions it can't make. Sentences which include these phenomena were still required to have all the other aspects of their parse done properly. The rest of this section discusses why the excluded phenomena should not pose any particular problem for extensions to NNEP within the S&A architecture.

The problem of choosing between multiple possible parses has already been discussed in sections 4.5 and 6.3. Section 6.3 shows that NNEP is capable of representing the information that the parser needs to know for disambiguation when it needs to know it, without having to represent information which it can't be sure of. In order to make disambiguation decision using this information, the parser needs to make use of a large amount of statistical information about distributions in the language. Connectionist architectures are rather good at representing and calculating with statistical information through the use of continuous valued features and soft constraints. Section 4.5 showed one way in which this ability could be used in NNEP. Because this mechanism uses its statistical information in a massively parallel fashion, the large quantity of this information does not pose a problem for NNEP's processing. The one aspect of disambiguation which prevents the testing of the proposed disambiguation mechanism is the collection of this large amount of information. Connectionist architectures are also rather good at this aspect of statistical decision making. Other approaches to learning and probability estimation could also be used to collect the

[35]Currently, the simulation of the implementation of NNEP has been run on the first 20 of these sentences. For the excluded phenomena discussed in this section, the portion of the sentence which contained this phenomena was excised before it was run. In the case of coordinations, the sentence was run twice, once for each of the conjuncts. This resulted in 24 runs. There were 10 manual interventions, three in one sentence, two in two sentences, and one in three sentences. Three of these were lexical choices, and the rest were internal attaching choices. No post hoc adjustments of lexically specific parameters were made to eliminate these interventions, since there are so many such parameters that this would almost certainly result in no remaining interventions. These 20 sentences and these interventions are given in appendix A. Grammar entries and parses have been specified by hand for all the sentences. For the excluded phenomena discussed in this section, the desired result was specified, but the process by which that result is achieved was not.

required information, given the clear relationship between the calculations described in section 4.5 and probability estimation. Since the analysis in section 4.5 broke the estimates up into simple statistically independent components, learning them should be tractable. Since the question of learning is beyond the scope of this dissertation, questions of disambiguation were not addressed in any significant way when testing NNEP on the Brown corpus sentences.[36]

Other than disambiguation, the most common phenomena which NNEP is not currently designed to handle is coordination. Coordination is a very hard phenomena, and many parsers have difficulty with it. One approach which has been rather successful at characterizing coordination is Combinatory Categorial Grammar (Steedman, 1985, 1987). As discussed in chapter 3, there is a rather close relationship between CCG and SUG. In particular, the forgetting operation defined in section 3.4 allows NNEP to abstract away from information in SUG descriptions in the same way as CCG abstracts away from information in its categories. It is this ability to abstract that allows CCG to handle coordination. By abstracting away from the irrelevant differences between phrases, CCG can use the simple generalization that phrases of the same type can be coordinated to produce a phrase of that type. In SUG forgetting can be used to abstract away from the differences between two tree fragment descriptions. NNEP could process coordinations by finding phrases with the same abstract SUG description on either side of the coordinating word, and parsing the sentence as if those phrases were a single phrase with that SUG description as its type. Parsing coordinations would then involve constructing the SUG description for the second conjunct, finding nodes in the SUG description for the preceding portion of the sentence which match with nodes in the second conjunct's description, forgetting the nodes in the coordinated phrases' descriptions which are not involved in this match, and constructing the result of conflating these pairs of matched nodes. The process by which the parser finds these matches and constructs the resulting structure has not yet been developed. The issue of coordination obviously needs much more investigation.

Another problem arose with the construction "more than". The three examples of this construction are given below.

120: We're getting more pro letters than con on horse race betting, said Ratcliff.
631: India is the most populous United Nations member with more than 400,000,000 inhabitants.
697: The odds favor a special session, more than likely early in the year.

All these examples involve a missing constituent in the argument to "than". In 120 it is "letters", in 631 it is "inhabitants",[37] and in 697 it is "early in the year". This is similar to the kind of gaping that is found in coordination, and perhaps a CCG style analysis of this phenomena could be incorporated into NNEP. This is a very complicated phenomena, but I know of no reason why a parser like NNEP would have any more difficulty with it than other parsers.

[36]The simulation of the implementation does support continuous valued parameters, and this ability was used to represent distinctions such as the preference for argument attachments over adjunct attachments. Some disambiguation decisions were made on the basis of this kind of information.

[37]There is an alternative structure for 631 which has "inhabitants" as part of the argument to "than". This is presumably not the intended structure of the sentence, since this would mean the additional things are not inhabitants, which would have nothing to do with population.

Chapter 7

Conclusion

This dissertation has shown that the Shastri and Ajjanagadde connectionist computational architecture is adequate for recovering the syntactic constituent structure of natural language sentences, and has provided examples of significant predictions that the limitations of the architecture make about the nature of language. This was done by first characterizing the nature of computation in the S&A architecture in terms of symbolic computation under a set of constraints. A specific parsing model was then designed to comply with these constraints and constraints from the nature of the parsing task. This model was used to demonstrate the adequacy of the S&A architecture by implementing it in the architecture, and testing it on all the phenomena which are of particular concern, plus an approximately unbiased sample of text. The results of these tests show that the constraints imposed by the architecture do not interfere with a parser's ability to recover the syntactic structure of natural language sentences. In addition, the results of these tests give examples of some significant properties of language that can be explained by these computational constraints. Thus the study of parsing in this connectionist architecture is not merely a question of implementing pre-existing theories of grammar and parsing, but can help inform the study of the nature of language. The biological motivations for the architecture make this later point more significant, since they provide independent motivations for this particular set of computational constraints, and therefore give them explanatory power. These insights into the nature of language in turn help in the development of natural language processing technology. Now that we know how symbolic approaches to language processing can be adapted for use in a connectionist network, the particular abilities of connectionist networks can be used to solve those problems which symbolic approaches have had the most trouble with.

In its first section, this chapter concludes the argument for the adequacy and significance of the S&A architecture by arguing that the tests given in the last chapter address all the phenomena that are of particular concern given the limitations of the architecture. Then the second section summarizes the work presented in this dissertation. Some significant aspects of this work that are not directly part of the argument for adequacy and significance are presented in section 3. This leads to a discussion of some promising future work that the work done so far makes possible. That discussion concludes this dissertation.

7.1 The Adequacy and Significance of the Architecture

The adequacy of the S&A connectionist architecture for constituent structure parsing was demonstrated by addressing all the implications of the architecture's computational constraints in the testing of a parser which has been implemented in this architecture. The specific phenomena which are of concern given these constraints were identified and data sets were selected for testing in these areas. To make sure nothing was missed in this analysis, the parsing model was also tested on an essentially unbiased set of sentences. The proposed parsing model was successful in all these tests. In addition, the results of these tests provided examples of significant predictions that the architecture's constraints make about the nature of language.

The constraints imposed by the S&A architecture and the nature of the parsing task were identified in chapter 2, and are listed again below. The implications of each of these constraints were addressed in testing the adequacy of the S&A architecture. The architecture's predictions about the nature of language were mostly due to constraint 4 and the bounds on the data structures that are needed to comply with this constraint.

1. information about at most ten nodes stored at a time
2. no explicit representation of disjunction between predications
3. at most three tuples of unary predicates for storing relations
4. at most one variable can appear in both a rule's antecedent and consequent
5. input is incremental
6. parse in quasi-real time
7. produce maximally incremental output
8. produce monotonic output

The bound on the number of nonterminal nodes was addressed by testing NNEP on center embedding data (section 6.4). This test found that this bound was not a problem for any of the acceptable sentences. While this data addressed the cases where a large number of nodes have to be stored because they are unambiguously needed, it did not address the cases where a large number of nodes have to be stored just in case one of them is needed. No appropriate data set was found for these cases, but previously proposed theories of constraints on this phenomena restrict the number of such nodes which need to be stored to within the bound of ten (section 4.1.3).

The prohibition against disjunction combines with the requirements for incremental and monotonic output to constrain the parser's representation phrase structure information. To produce incremental output, the parser needs to be able to represent everything it knows about the phrase structure of the sentence. To produce monotonic output, the parser needs to be able to avoid representing information that it can't be sure of. Thus the parser must be able to accurately characterize the phrase structure information about a sentence, and do so without using disjunction. The test of NNEP's ability to express phrase structure analyses (section 6.1) demonstrated this ability for the unambiguous case. For the case where multiple analyses are locally possible, this ability was demonstrated using data pertaining to ambiguity resolution (section 6.3). For all the ambiguities which did not cause garden paths in either of their completions, NNEP was able to represent both possible completions until there was enough evidence to resolve the ambiguity (except one, where the data is unclear).

Parsing in quasi-real time means that the processing done between any two words and between the last word and the completion of the parse can only take a bounded amount of time. Each parser

action takes a bounded amount of time, so this constraint requires that the number of actions done between any two words be bounded. Only one action is necessary to combine a word's grammar entry with the parser state. Since each internal operation results in one less unparented node, and there are only a bounded number of unparented nodes in the parser state, only a bounded number of internal operations can be done between any two words. The only other action the parser can take is to combine a nonlexical grammar entry with the parser state. Nonlexical grammar entries can generally be precombined with lexical grammar entries, thereby eliminating the need for separate parser actions to use them, except in the cases where nonlexical grammar entries are needed to delay resolving an ambiguity. In the test of NNEP's ability to delay commitments long enough to find disambiguating information (section 6.3), never more than one nonlexical grammar entry had to be used between any two words. Thus NNEP only needs to do a bounded number of actions between any two words, and parsing can be done in quasi-real time.

The constraint that bounds the number of instantiations of relations that can be stored is closely tied to the locality constraint on rules, since only the processing of relations might require pairs of nodes to be propagated from a rule's antecedent to its consequent. This is manifested in NNEP by the fact that constraint 3 is subsumed by the bounds on the data structures that are used to compensate for constraint 4. Thus no additional testing was needed for the bound on the number of instantiations of relations that can be stored.

The locality constraint on rules can have implications for any computation that involves relationships between nonterminal nodes. NNEP is designed so as to minimize the use of relations. The only relations that are in principle needed involve nodes that represent traces for long distance dependencies, the roots of unattached tree fragments, and nonterminals that are specified as preceded by other nonterminals. In the later case, the simplest solution is to use two unary predicates rather than a binary predicate to represent the ordering constraints. This can only work if the nesting of the relevant constructions is highly constrained. The center embedding data on this issue (section 6.4) indicates that this prediction holds. For relations involving the roots of unattached tree fragments, the locality constraint on rules requires that rules that calculate these relations and attach the tree fragments must be constrained to only apply to one tree fragment root at any given time. The nested nature of English sentences ensures that the rightmost tree fragment root can always be this unique node. All the testing bears this out. In order to keep track of which tree fragment is the rightmost one, a list of tree fragments is needed. This data structure can be implemented with a set of unary predicates to keep track of the list order, and rules to update these predicates. However, since a network can only have a fixed number of predicates, the length of the list must be bounded. If only independently motivated predicates are used to represent the state of the list, then it can only be four tree fragments long. Testing on center embedding data (section 6.4) indicates that this length is sufficient, and this constraint correctly predicts some center embedded sentences to be unacceptable.

The most interesting implications of the locality constraint on rules involve the mechanisms NNEP needs to use to process long distance dependencies. Recovering long distance dependencies requires calculating which nodes might be the gap site for a trace node. To calculate these relationships with rules that can propagate information about only a single node, these rules must be restricted to only apply to one trace node at a time. For English this is the most recently introduced trace node, so a stack data structure is needed to keep track of which trace node was introduced most recently. Testing on long distance dependency data and a comparison with the analysis of long distance dependencies given in (Kroch, 1989) (section 6.2) showed that this stack constraint did not interfere with recovering long distance dependencies. This comparison also showed that this constraint correctly predicted several constraints on long distance dependencies, including some

that required Kroch to go outside of the power of his formalism to characterize. As with the list of tree fragments, this stack data structure can only be of bounded depth. Assuming only independently motivated predicates are used to represent the state of the stack, it can only be two deep. Given a particular independently motivated strategy for when to make use of this stack, this bound does not interfere with the parsing of the acceptable sentences in the center embedding data, and it correctly predicts the unacceptability of a number of these sentences (section 6.4).

The tests done in specific areas address all the implications of computational constraints on the parsing model, but to make sure that nothing has been missed in this argument, NNEP was also tested on an essentially unbiased set of sentences. This test pointed out three phenomena which NNEP is not currently equipped to handle, but these are all difficult problems for any parsing model, and they do not seem to be any harder for models implemented in the S&A architecture. In fact, there are reasons to believe that each of them will be easier for this type of model then they are for most parsers. The most important of these phenomena is resolving ambiguity. As argued in section 6.3, when NNEP needs to resolve an ambiguity, it has the information needed to do so. This information includes the current representation of the sentence's phrase structure, the next two words in the input, and any input from other language processing modules. Any of this information may be noncategorical, so statistical information can be used in this decision. Section 4.5 gave one plausible mechanism for statistical ambiguity resolution, and in general connectionist networks are quite good at these kinds of tasks. Robust ambiguity resolution in general requires both automatic learning of statistical information, and the ability to use that information in decision making. Connectionist networks have both these abilities.

The other two phenomena which were encountered but not handled were coordination and gaping in comparatives. As sketched in section 3.4, there is a close relationship between the derivation structures of SUG and those of Combinatory Categorial Grammar (CCG). Since NNEP follows SUG derivation structures when parsing, it should be possible to adapt CCG's successful analysis of coordination for use by NNEP. As discussed in section 6.5, under one such analysis NNEP would build the right conjunct as a separate tree fragment, and then find a matching subtree in the adjacent tree fragment. Nodes in these subtrees which were not involved in the match would be abstracted away from using forgetting, the matching nodes in the subtrees would be conflated, and the parse would be continued using the single conflated subtree. At the moment this analysis is just speculation, but it provides reason to believe that a model of parsing like NNEP could be extended to handle coordination. As also discussed in section 6.5, a similar argument applies to gaping in comparatives. Such gaping is similar to gaping in coordination, so there is reason to believe that a similar mechanism can be used for this phenomena.

7.2 Summary

This work has provided substantial evidence that the Shastri and Ajjanagadde connectionist computational architecture is adequate for recovering the constituent structure of natural language sentences. A specific model of constituent structure parsing in the S&A architecture was presented which addresses the limitations of this architecture. This model, called a Neural-network Node Equating Parser (NNEP), was tested in the areas which are of particular concern given these limitations, and with an approximately unbiased sample of text. The results show that the limitations of the architecture do not interfere with a parser's ability to recover the constituent structure of natural language sentences. In addition, the results of these tests indicate that some interesting properties of language can be explained by these limitations. This makes the study of parsing

within this architecture interesting for the study of the nature of language, as well as for the study of parsing and connectionism.

The specific areas which were addressed in the testing of the proposed parsing model were: expressing phrase structure analyses, recovering long distance dependencies, representing local ambiguities, and staying within the architecture's resource bounds. For the first area a general technique was given for mapping phrase structure analyses from theories of linguistic competence to a grammatical representation which is appropriate for NNEP. A specific mapping was given for the phrase structure analyses used in (Kroch, 1989). (Kroch, 1989) was chosen because it surveys some important issues in the study of linguistic competence, and because it uses a representational framework which is similar to the one used in this dissertation. The issues addressed in (Kroch, 1989) are primarily concerned with the nature of long distance dependencies, so it also formed the basis of the argument that NNEP can recover all acceptable long distance dependencies. NNEP's representations are adequate for expressing the information which Kroch shows is necessary for determining what the possible long distance dependencies are, and its computations are adequate for successfully using that information in parsing. In addition, due to the computational limitations of the S&A architecture, NNEP is unable to recover some classes of dependencies which are ruled out by Kroch within his competence theory. These computational limitations were used to explain pre-inflection subject islands, the that-trace effect, and the limited possible extractions out of wh- islands. The later phenomena are particularly interesting, because accounting for this data required Kroch to go outside the power of his grammatical framework. If the burden of characterizing this phenomena is shifted from the competence theory to the processing theory, then the competence theory would be significantly simplified.

The testing of NNEP's ability to represent local ambiguities and stay within its resource bounds was done using the data in the appropriate chapters of (Gibson, 1991). (Gibson, 1991) was chosen because it provides a survey of the phenomena involved in ambiguity resolution and processing overload. For each acceptable example of a local ambiguity given in (Gibson, 1991), representations are given which demonstrate how the resolution of that ambiguity can be delayed until disambiguating information is found. The question of what constitutes disambiguating information is addressed by providing specific disambiguation strategies. There is one pair of sentences ((247a) and (249a)) for which an appropriate representation could not be found, but in this case my intuitions disagree with Gibson's, and we both agree that experiments are needed to resolve the issue. No significant attempt was made to explain when people do have trouble with local ambiguities, and only one such situation was discussed. For the acceptable example sentences in (Gibson, 1991) pertaining to processing overload, parses were given which do not violate any of NNEP's resource bounds. These resource bounds require that at any one time the parser state can have no more than ten nonterminals, four tree fragments, two nodes on the public node stack, and one linear precedence constraint between sets of nodes. The first constraint was not adequately tested because of a lack of data, but theories of the relevant phenomena are compatible with this constraint. Also, this constraint never had to be violated for the data which is available. The other constraints were thoroughly tested, using both the data from (Gibson, 1991) and a few sentences which were devised to address specific questions pertaining to these constraints. A strategy for when to use the public node stack was also proposed and argued for, and with this strategy many of the unacceptable sentences discussed in (Gibson, 1991) were predicted to violate NNEP's resource bounds. Although several other unacceptable sentences remain unaccounted for, the close fit with the data which is addressed makes the prospects of developing this model into a complete theory of these phenomena promising. It also provides independent evidence for the data structures which were involved in the explanation of constraints on long distance dependencies.

In addition to the four specific areas in which NNEP was tested, an essentially unbiased sample of text was used to make sure that no errors were made in the identification of the phenomena which are of particular concern for this investigation. This sample consists of 50 thirteen word sentences which were randomly selected from the Brown corpus by Ezra Black. There are some phenomena in this sample which NNEP is not currently equipped to handle, but none of them would be any harder for the S&A architecture than they are in general. These phenomena are: disambiguation decisions, coordination, and gaping in comparatives. With the exception of these phenomena, grammar entries and parses were found for each of the sentences.

The model of syntactic parsing used in the above tests is designed to compensate for the limitations of the S&A architecture while imposing as few additional constraints as possible. Towards this end, NNEP uses Structure Unification Grammar (SUG) as its grammatical framework. SUG makes extensive use of partial descriptions, including the ability to partially specify structural information. This allows a parser to specify what it needs to know about the phrase structure of the sentence independently from what it can't be sure of, or doesn't need to know. By providing NNEP an expressive language for specifying what it knows, SUG allows NNEP to represent the necessary phrase structure and long distance dependency information. By not forcing NNEP to store more than it knows, SUG allows NNEP to represent the necessary local ambiguities. By not forcing NNEP to store what it no longer needs to know, SUG allows NNEP to parse within its resource bounds. SUG also helps NNEP compensate for its resource bounds by providing for a representation of constituent structure that has a minimal number of nodes. Other than the prohibition against the use of disjunction imposed by the S&A architecture, SUG does not restrict its derivations beyond the requirement that the resulting description be consistent and complete. To be this general, nonterminal node equation is used as the operation for combining partial descriptions, since all formalisms involve node equations when viewed as accumulating partial descriptions of phrase structure trees.

NNEP stores SUG partial descriptions in its memory and follows SUG derivations to recover the phrase structure of a sentence. After each step in which NNEP recovers new information about the phrase structure, that information is output to other language modules. The set of such steps which are consistent with the limitations of the S&A architecture can be characterized with six operations. Four of these operations (equationless combining, attaching, leftward attaching, and double attaching) are used to combine a grammar entry for the next word with the phrase structure description built from the previous words of the sentence. Two of these operations (internal attaching, and internal trace equating) perform equations between nonterminal nodes which are already represented in the parser state. The ability to delay attachment decisions using equationless combining or leftward attaching, and then perform them using internal attaching, allows NNEP to represent attachment ambiguities until enough information is available to resolve them. In order to deal with the bounded number of nonterminals which NNEP can store, there is also an operation for forgetting nodes which won't be equated with during the remainder of the parse. In addition to these seven operations, NNEP has rules which iteratively calculate possible long distance dependencies. One of these rules calculates what nodes in an attached tree fragment might be a gap position, and one calculates what nodes' constituents might contain a gap position. These rules are adequate for finding possible long distance dependencies, and the domain over which they apply is sufficiently large to express the necessary constraints on possible long distance dependencies. Because these rules do not apply to tree fragments which have not yet been attached to the tree fragment which contains the wh- word, they can not find gap positions within pre-inflection subjects. Because these rules must be constrained to only apply for the most recently introduced trace node, many constraints on possible extractions out of wh- islands are predicted.

NNEP has been implemented in the S&A architecture as a special purpose module for doing syntactic parsing. In this module's memory, nonterminal nodes are the entities, and all the information recovered so far about the phrase structure of the sentence is stored as predications over those nodes. All but two of these predicates are unary predicates. The two binary predicate represent what nodes might eventually be equated with what other nodes, and what nodes might eventually dominate what other nodes via links which have not yet been introduced. Pattern-action rules are used to perform computations over this information. Most of these rules implement the grammar and the combination operations. For each grammar entry, there is a set of patterns and a set of actions which compute how the grammar entry can be combined with the description in the parser state. The patterns and actions are separated by an arbitrator, which decides which of the matching patterns should have its action applied. There are also patterns, actions, and arbitrators for the two internal equating operations. As required by the limitations of the S&A architecture, all these operation rules match and apply for a single nonterminal node. The movement rules and other rules which calculate indirectly implied information are in part compiled into the actions of the operation rules, but are in part distinct. Similarly, the information about nodes which indirectly influence the applicability of an operation is calculated with rules which are in part distinct from the operation rule patterns. Because of constraints imposed by the S&A architecture, all these rules for indirectly implied information are specific to a uniquely identifiable node or set of nodes. This constraint requires that the movement rules and the double attaching operation only apply for a single node, called the public node. A stack is used to keep track of what node is the current public node, thereby constraining rules involving trace nodes to only apply to the most recently introduced trace node.

The S&A connectionist computational architecture can be used to implement NNEP because it supports symbolic computation. Units are used to represent properties, links are used to compute rule applications, and temporal synchrony of unit outputs is used to represent which entities have which properties. These mechanisms allow predications over variables to be stored, and allow pattern-action rules to dynamically manipulate those predications, thereby supporting symbolic computation. The limitations of this architecture are that the number of variables is bounded, there is no explicit representation of logical connectives, and relationships between variables are costly to store and compute with. Biological considerations put the bound on the number of variables at at most ten. Because the default connective is conjunction, the memory cannot explicitly represent disjunction. Using temporal synchrony, a unit's activation can only be used to explicitly represent information about one variable or the situation as a whole, so any relationships between variables must be represented in the identity of the unit. This means that any rule at the implementation level can only generalize over one entity. This restriction is a form of locality constraint on rules. This constraint means rules which involve relationships between individual entities, such as the movement rules, must have one of those entities be uniquely identifiable when the rule applies. The difficulty with representing relationships between variables also motivates a constraint on the number of instantiations of a relation that can be stored at any one time.

Some additional constraint are imposed on the parsing model by the input to and output from the syntactic parsing module. For spoken language, and to a large degree for reading, words are input to the parser one at a time, with a bounded amount of time between words. Thus the parser must accept incremental input, and parse in quasi-real time. The output of the parser needs to be incrementally interpretable by other language modules. Thus the parser's output must be incremental and monotonic. In conjunction with the prohibition against the use of disjunction, these later constraints imply that the parser must be deterministic, in the sense of (Marcus, 1980). The total set of constraints on the parsing model are that the input is incremental, the representation of

the sentence's phrase structure can only include at most ten nonterminals and three instantiations of each relation, the representation cannot use disjunction, the rules must apply to each variable independently, parsing must be done in quasi-real time, and the output must be incremental and monotonic.

7.3 Discussion

The work presented in this dissertation is interesting for reasons other than the specific claims that are argued for. It has implications for both natural language processing technology and cognitive models of sentence processing.[1] These implications are in part because of the computational characteristics of the proposed model. In addition to being incremental, deterministic, having bounded memory, and parsing in quasi-real time, NNEP has other interesting characteristics. Because the rules which implement the grammar compute in parallel, NNEP's speed is independent of the size of its grammar. Even the propagation delays introduced by a larger grammar won't slow down the parser, provided they are less than one period long. Also, if two such rules have the same affect on the parser state, then they share the same units in the implementation. Thus the number of units in the implementation grows with the number of different structures in the grammar, not with the number of words. In addition, because the computational architecture is biologically plausible, a large amount of neurological evidence which has been difficult to apply to abstract parsing models could be brought to bear. Finally, there is a clear relationship between the connectionist primitives in the network and their symbolic interpretations. This allows any future work on such parsers to take advantage of both the results from investigations into connectionism and the results from investigations into symbolic sentence processing. Since the successes of these two approaches are generally complementary, this prospect holds great promise for advances in the study of sentence processing.

For the development of natural language processing technology, NNEP's efficiency and the prospect of taking advantage of connectionist methods make NNEP an interesting model. Because NNEP is deterministic and massively parallel, a parallel implementation of it could be extremely fast. Also, the success of connectionist networks in other learning problems makes the possibility of studying grammar induction with a model like this one promising. This fact and the ability to do evidential reasoning within the S&A architecture suggest that such a model could also be good at resolving syntactic ambiguities. Grammar induction and disambiguation are currently the greatest challenges for syntactic parsers. The symbolic interpretation of the network should allow connectionist methods to be used in these investigations while still taking advantage of our current understanding of the issues involved. For example, because the current parser shares components of the network across grammar entries for different words, learning in this network would inherently generalize training examples pertaining to one word to the other words which share its grammar entry's components. Because we know what linguistic constructs different portions of the network implement, other types of sharing can be used to place linguistically motivated constraints on how the parser generalizes in automatic grammar induction.

For the investigation of cognitive models, NNEP's computational characteristics and the work which has already been done on center embedding and constraints on long distance dependencies

[1] Contrary to common practice, these are not unrelated topics. The way people communicate with language is finely tuned by years of linguistic interaction with other people. This level of communication ability requires both knowledge of what people can understand and what they can't understand. While it is possible that research efforts which ignore the data from cognitive modeling could develop a natural language processing system which has both these abilities, I believe that developing such a system will eventually require a theory of human sentence processing.

suggest that it may be possible to develop this parser into a model of human sentence processing. NNEP's incrementality, monotonicity, bounded memory, parallelism, speed, size, and ability to interact with other modules comply with many researcher's beliefs about the nature of the human parser. The successful use of connectionist networks in other cognitive models also bodes well for this endeavor. Perhaps most significantly, the ability to use biological measurements to make fine grained predictions based on the time course of processing in such a model could be a very powerful tool in developing such a model. The number of words NNEP can parser per second has already been shown to be in the right general area based on biologically determined parameters. Recent work on event related potentials (ERP's) could be particularly interesting for such an effort, since they provide on-line measures of processing at an extremely fine time scale.

The fact that the underlying computational mechanisms which are used in this investigation are the same as those used in the work on reflexive reasoning in the S&A architecture (Shastri and Ajjanagadde, 1993) is also significant for cognitive modeling, since it strongly suggests that fundamentally different computational mechanisms did not have to be developed for language processing. What exactly did have to evolve for the development of the human linguistic capacity is clearly far beyond the scope of this work, but perhaps future work on grammar induction can address this issue.

7.4 Future Work

As suggested by the last section, there are several directions in which this work could be continued. Advances in the various areas of natural language processing technology discussed above could be applied to broad coverage parsing. As discussed above, this work could be used in cognitive models of sentence processing. Also, this model of constituent structure parsing could be interfaced with other language processing modules.

Developing NNEP into a broad coverage parser would involve addressing grammar induction, disambiguation, coordination, and network simulation. Grammar induction is necessary because a real grammar is too large to be hand coded. For NNEP, grammar induction could be done either by automatically decomposing structures from preparsed corpora into grammar entries, or by starting with a broad class of grammar entries and removing or keeping them based on their usefulness in parsing corpora. Connectionist methods use the later approach. One advantage of this approach is that it can subsume grammar induction under statistical parameter estimation. A grammar entry is in the grammar if its probability is nonzero. Statistical parameter estimation is needed for doing disambiguation. Disambiguation also requires that NNEP's arbitration mechanism be redesigned. Preliminary work on a Bayesian decision theoretic analysis of the parser's disambiguation task has begun this process. Using prior information about the distribution of both phrase structure trees (i.e. the grammar) and resource usage, these two constraints on the parser can be handled uniformly in terms of probability of parser failure. Thus a single arbitration mechanism can choose between both internal operations and combination operations. This analysis has revealed significant parallels between the learning problem faced by NNEP and those solved with the backpropagation learning algorithm (Rumelhardt *et al.*, 1986).

One issue in adapting connectionist learning techniques to a network like NNEP is how to handle the addition of the temporal synchrony variable binding mechanism. Given the locality constraint on rules that has been assumed here, this is rather simple. This constraint means that all adjustable parameters are in portions of the network that test or set information about individual phrase

structure nodes or information about the structure as a whole. Both these kinds of information are available within individual phases of the temporal pattern of activation. Thus the same techniques that apply for networks that don't use temporal synchrony can be applied during each phase in a network that does use temporal synchrony. The same techniques that are used for recurrent connectionist networks can be used to handle the fact that a computation in one phase can affect computation in another phase by setting predications about the structure as a whole.

Applying connectionist learning techniques to the network developed in this dissertation will require a couple things to be done. First, an adequate class of grammar entries of a tractable size needs to be found. The grammar entries used in chapter 6 should form a basis for determining an appropriate class. Second, the disambiguation mechanism used in the current parser needs to be redesigned to better match the kinds of networks which connectionist learning methods are designed for. The preliminary work mentioned above on a Bayesian approach to parsing decisions is a start in this project. While these problems will require a significant amount of work, the success that connectionist networks have had for similar problems makes the application of connectionist learning methods to grammar induction and disambiguation very promising.

Because of their fundamentally different nature, some linguistic phenomena are not likely to be addressed by simply adding grammar induction to the current model. Coordination is a very common example of such a phenomena. A special mechanism needs to be added in order to handle the gaping which occurs in coordination and some other phenomena. As was discussed above and in section 6.5, the close relationship between the derivation structures of NNEP's grammatical framework and the derivation structures of Combinatory Categorial Grammar suggests how such a mechanism can be developed.

In addition to the above natural language processing issues, broad coverage parsing with a parser like NNEP requires that the connectionist network be efficiently simulated. This in turn requires an adequate development environment for networks of this kind. Currently the Rochester Connectionist Simulator is being used, which is not adequate for developing large networks with the kinds of complex link interconnections, complex unit organization, and time sensitive behavior that a model like NNEP has. Mechanisms are needed for defining and using modular representations of link interconnections and unit organization. In addition to controlling complexity, such mechanisms would allow the network to be defined and simulated at different levels of abstraction, which could significantly speed up simulation on serial machines. Automatic mechanisms for accounting for propagation delays would also be extremely useful. For the work involving large corpora discussed above, a parallel simulation of the network may be necessary.

Future work on a parser like NNEP could address several issues in cognitive modeling. More work could be done on computational explanations for constraints on long distance dependencies, and for constraints on center embedding. Garden path phenomena were only touched on in this dissertation, and more work in this area would likely be fruitful, especially if combined with a theory of ambiguity resolution. A theory of ambiguity resolution would also be useful in an investigation of what nodes on the right frontier of a structure are available for modification. This investigation would be needed to answer the remaining questions about the implications of the architecture's bound on the number of nonterminals which can be stored. Since resource bounds and ambiguity resolution are likely to be involved in the explanation for the ordering preferences associated with heaviness phenomena, it would be interesting to investigate both the mechanisms and motivations for heavy shift in a model like NNEP. The biological plausibility of the computational architecture makes modeling on-line measures of sentence processing another interesting area for future work on a parsing model like NNEP.

The above research directions could all be done mostly within a model of constituent structure parsing. Eventually such a model needs to be embedded in a larger model of language processing, so it would also be fruitful to investigate interfacing a parser like NNEP with other language processing modules. From the input side, NNEP could be connected with a model of speech recognition. Under the current disambiguation mechanism, the speech recognition module would output probability distributions over word-tag pairs, where the word is the next word in the input and the tag is the category of the subsequent word in the input. From the output side, a module for constructing predicate-argument structure could be connected with NNEP. This would likely require a more complex interface, since some of the issues now addressed in NNEP are probably best handled within the predicate-argument structure module. In particular, what nodes are *foot* nodes and what nodes are *extractable* are probably determined within this module. This would require an interface which allowed constraints from the predicate-argument structure module to be communicated to the constituent structure module. Similar mechanisms would also be needed to interface NNEP with a discourse processing module. NNEP outputs a sequence of statements about how it is constructing the constituent structure of the sentence. Using a grammar which gave the relationships between constituent structure nodes and predicate-argument structure nodes, the predicate-argument structure module could interpret these statements as instructions for building the predicate-argument structure.[2] Although sometimes these instructions would be ambiguous, most of the structural ambiguities would be resolved at the level of the constituent structure.

It is indicative of the nature of this investigation that the work which it could lead to is so abundant and diverse. A question such as the adequacy of a computational architecture for a task inherently involves a broad range of issues in that task. Fortunately, the results of this investigation have been worth the effort. By demonstrating that there is no problem in constituent structure parsing which prevents the S&A architecture from performing this task in general, we are now justified in spending time investigating specific parsing issues using the S&A architecture.

[2]Synchronous Tree Adjoining Grammars (Schabes and Shieber, 1990) provides a similar analysis of the relationship between these two levels of representation.

Appendix A

Disambiguation Interventions

The following two lists give the sentences which have been run on the computer simulation of the connectionist implementation of the parsing model of this dissertation, and the interventions which were needed. Sentences with conjunctions were run twice, once for each conjunct, and these are labeled "n.1" and "n.2". In addition, "than con" was excised after "letters" in sentence 120.

The 18 Sentences from (Kroch 1989)

(2) On Thursday, what will you buy?

(3a) ? On that shelf, how_many books can fit?

(5) After the party, I wonder who will stay.

- picked "stay" for internal trace equating of topic's trace

(7) After the party, who will stay?

(23b) When does he think that we left?

- picked "left" for internal trace equating of "when"'s trace

(28a) I saw an old St. Bernard with a limp yesterday.

- picked "Bernard" for attaching of "with"

(28b) I saw yesterday an old St. Bernard with a limp.

(29a) An old St. Bernard with a limp came by yesterday.

(30) Which old St. Bernard came by yesterday?

(36) Which painting did you see?

(37) Which painting did you see a copy of?

(38) Which painting did you see a photograph of a copy of?

(50) ? Which book did you reject the idea that students should read?

- forced internal operation (attaching) after "idea"

(56a) ? What were you wondering how to say?

- forced internal operation (trace equating) after "to"

(60a) I knew which book the students would forget who wrote.

(60b) I knew which book the TAs told us that the students would forget who wrote.

(69) I know who John persuaded to visit the town.

(70) I know which town John persuaded you to visit.

The First 20 Sentences from the Brown Corpus Data

31.1 The petition listed the mayor's occupation as attorney.

31.2 The petition listed his age as seventy-one.

37 His political career goes back to his election to city council in nineteen-twenty-three.

- picked "election" for internal attaching of "in"

120 We're getting more pro letters on horse race betting said Ratcliff.

- picked "letters" for internal attaching of "on" before "betting"

137 It was one of a series of recommendations by the Texas Research League.

150 All Dallas members voted with Roberts except Rep. Bill Jones who was absent.

156.1 Most of the fire was directed by Cotten against Dallas.

- picked proper noun entry for "Dallas" (over compounding noun)

156.2 Most of the fire was directed by Cotten against Sen. Parkhouse.

172 Fifty-three of the one-hundred-fifty representatives immediately joined Grover as co-signers of the proposal.

198 Bellows made the request while the all-woman jury was out of the courtroom.

200 Some of the defendants strongly indicated they knew they were receiving stolen property.

214 He said evidence was obtained in violation of the legal rights of citizens.

- picked IP of "obtained" for internal attaching of "in"
- picked "rights" for internal attaching of "of"

222 He said this constituted a very serious misuse of the Criminal court processes.

270 He also asked Congress to approve establishment of a national child health institute.

286.1 Only eleven senators were on the floor.

286.2 There was no record vote.

337 Mr. Nixon for his part would oppose intervention in Cuba without specific provocation.

375 One of these men is former Fire Chief John A. Laughlin he said.

409 It declares that Sunday sales licenses provide great revenue to the local government.

- picked object of "declares" for attaching of "that" (to prevent it becoming a determiner)
- picked "provide" for internal attaching of "to"

419 The lawyer with whom I studied law steered me off the Socialist track.

426 Mr. Reama far from really being retired is engaged in industrial relations counseling.

- picked leftward attaching of "being" with "really"
- picked object of "being" for internal attaching of "retired"
- picked IP of "engaged" for internal attaching of "in"

451 At present all offenses must be taken to Sixth District Court for disposition.

516.1 He also expanded the radio system with a central control station.

516.2 He also modernized the radio system with a central control station.

Bibliography

(Aaronson, 1991) Jeffrey Aaronson. Dynamic fact communication mechanism: A connectionist interface. In *Proceedings of the Thirteenth Conference of the Cognitive Science Society*. Lawrence Erlbaum, 1991.

(Abeillé *et al.*, 1990) Anne Abeillé, Kathleen Bishop, Sharon Cote, and Yves Schabes. A Lexicalized Tree Adjoining Grammar for English. Technical Report MS-CIS-90-24, Department of Computer and Information Science, University of Pennsylvania, April 1990.

(Abney, 1986) Steven Abney. Licensing and parsing. In *Proceedings of NELS 16*, Amherst, MA, 1986.

(Altmann and Steedman, 1988) Gerry Altmann and Mark Steedman. Interaction with context during human sentence processing. *Cognition*, 30:191–238, 1988.

(Barnden and Srinivas, 1991) J. Barnden and K. Srinivas. Encoding techniques for complex information structures in connectionist systems. *Connection Science*, 3(3), 1991.

(Becker, 1993) Tilman Becker. *HyTAG: A New Type of Tree Adjoining Grammars for Hybrid Syntactic Representation of Free Word Order Languages*. PhD thesis, Universität des Saarlandes, Saarbrücken, 1993.

(Berwick and Weinberg, 1984) Robert Berwick and Amy Weinberg. *The Grammatical Basis of Linguistic Performance*. MIT Press, Cambridge, MA, 1984.

(Bever, 1970) Thomas G. Bever. The cognitive basis for linguistic structures. In J. R. Hayes, editor, *Cognition and the Development of Language*. John Wiley, New York, NY, 1970.

(Chomsky, 1959) Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 2:137–167, 1959.

(Church, 1980) Kenneth Church. On memory limitations in natural language processing. Master's thesis, Massachusetts Institute of Technology, 1980. MIT LCS Technical Report 245.

(Cottrell, 1989) Garrison Weeks Cottrell. *A Connectionist Approach to Word Sense Disambiguation*. Morgan Kaufmann Publishers, Los Altos, CA, 1989.

(Cowper, 1976) E.A. Cowper. *Constraints on Sentence Complexity: A Model for Syntactic Processing*. PhD thesis, Brown University, Providence, RI, 1976.

(Eady and Fodor, 1981) J. Eady and J.D. Fodor. Is center embedding a source of processing difficulty? Presented at the Linguistic Society of America annual meeting, 1981.

(Elman, 1990) J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–212, 1990.

(Elman, 1991) J. L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–225, 1991.

(Fanty, 1985) Mark Fanty. Context-free parsing in connectionist networks. Technical Report TR174, University of Rochester, Rochester, NY, November 1985.

(Fillmore *et al.*, 1988) Charles Fillmore, Paul Kay, and M.C. O'Connor. Regularity and idiomaticity in grammatical constructions: The case of let alone. *Language*, 64:501–538, 1988.

(Fodor and Pylyshyn, 1988) Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28:3–71, 1988.

(Gazdar *et al.*, 1985) Gerald Gazdar, Ewan Klein, Geoff Pullum, and Ivan Sag. *Generalized Phrase Structure Grammar*. Harvard University Press, Cambridge, MA, 1985.

(Gibson. *et al.*, 1993) E Gibson., N. Pearlmutter, E. Canseco-Gonzales, and G. Hickok. Cross-linguistic attachment preferences: Evidence from english and spanish. In *6th Annual CUNY Sentence Processing Conference*, 1993.

(Gibson, 1991) Edward Gibson. *A Computational Theory of Human Linguistic Processing: Memory Limitations and Processing Breakdown*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1991.

(Henderson, 1990) James Henderson. Structure unification grammar: A unifying framework for investigating natural language. Technical Report MS-CIS-90-94, University of Pennsylvania, Philadelphia, PA, 1990.

(Henderson, 1992) James Henderson. A structural interpretation of combinatory categorial grammar. Technical Report MS-CIS-92-49, University of Pennsylvania, Philadelphia, PA, 1992.

(Hindle, 1983) Donald Hindle. Deterministic parsing of syntactic nonfluencies. In *Proceedings of the 21st Annual Meeting of the ACL*. Association for Computational Linguistics, 1983.

(Jain, 1991) Ajay N. Jain. *PARSEC: A Connectionist Learning Architecture for Parsing Spoken Language*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1991.

(Johnson, 1990) Mark Johnson. Expressing disjunctive and negative feature constraints with classical first–order logic. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, Pittsburgh, PA, 1990.

(Joshi, 1987a) Aravind K. Joshi. An introduction to tree adjoining grammars. In Alexis Manaster-Ramer, editor, *Mathematics of Language*. John Benjamins, Amsterdam, 1987.

(Joshi, 1987b) Aravind K. Joshi. Word-order variation in natural language generation. In *AAAI 87, Sixth National Conference on Artificial Intelligence*, pages 550–555, Seattle, Washington, July 1987.

(Kaplan and Bresnan, 1982) Ronald Kaplan and Joan Bresnan. Lexical functional grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*. MIT Press, 1982.

(Kempen and Vosse, 1989) G. Kempen and T. Vosse. Incremental syntactic tree formation in human sentence processing: A cognitive architecture based on activation decay and simulated annealing. *Connection Science*, 1(3), 1989.

(Kimball, 1973) J. Kimball. Seven principles of surface structure parsing in natural language. *Cognition*, 2:15–47, 1973.

(Koster, 1978) J. Koster. Why subject sentences don't exist. In S.J. Keyser, editor, *Recent Transformational Studies in European Languages*. MIT Press, Cambridge, MA, 1978.

(Kroch, 1989) Anthony Kroch. Assymetries in long distance extraction in a tree adjoining grammar. In Mark Baltin and Anthony Kroch, editors, *Alternative Conceptions of Phrase Structure*. University of Chicago Press, 1989.

(Lambek, 1961) Joachim Lambek. On the calculus of syntactic types. In *Structure of Language and its Mathematical Aspects. Proceedings of the Symposia in Applied Mathematics, XII*, Providence, RI, 1961. American Mathematical Society.

(Lange and Dyer, 1989) T. Lange and M. Dyer. High-level inferencing in a connectionist network. *Connection Science*, 1(2), 1989.

(Lewis, 1993) R.L. Lewis. *An Architecturally-based theory of Human Sentence Comprehension*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1993.

(Marcus *et al.*, 1983) Mitchell Marcus, Donald Hindle, and Margaret Fleck. D-theory: Talking about talking about trees. In *Proceedings of the 21st Annual Meeting of the ACL*, Cambridge, MA, 1983.

(Marcus, 1980) Mitchell Marcus. *A Theory of Syntactic Recognition for Natural Language*. MIT Press, Cambridge, MA, 1980.

(McClelland and Kawamoto, 1986) J.L. McClelland and A. Kawamoto. Mechanisms of sentence processing: Assigning roles to constituents. In D. E. Rumelhardt and J. L. McClelland, editors, *Parallel Distributed Processing, Vol. 2*. MIT Press, Cambridge, MA, 1986.

(McClelland, 1986) J.L. McClelland. The programmable blackboard model of reading. In D. E. Rumelhardt and J. L. McClelland, editors, *Parallel Distributed Processing, Vol. 2*. MIT Press, Cambridge, MA, 1986.

(Miikkulainen and Dyer, 1991) R. Miikkulainen and M.G. Dyer. Natural language processing with modular pdp networks and distributed lexicon. *Cognitive Science*, 1991.

(Miller, 1956) G. A. Miller. The magical number seven plus or minus two. *Psychological Review*, 63:81–96, 1956.

(Pesetsky, 1982) D. Pesetsky. *Paths and Categories*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1982. [distributed by MIT Working Papers in Linguistics, Dept of Linguistics and Philosophy, MIT].

(Pollack, 1990) J.B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46:77–105, 1990.

(Pollard and Sag, 1987) Carl Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics. Vol 1: Fundamentals*. CSLI, 1987.

(Rounds and Kasper, 1986) William Rounds and Robert Kasper. A complete logical calculus for record structures representing linguistic information. In *IEEE Symposium on Logic and Computer Science*, 1986.

(Rounds and Manaster-Ramer, 1987) William Rounds and Alexis Manaster-Ramer. A logical version of functional grammar. In *Proceedings of the 25th Annual Meeting of the Association of Computational Linguistics*, 1987.

(Rounds, 1988) William C. Rounds. Set values for unification-based grammar formalisms and logic programming. Manuscript, CSLI and Xerox PARC, 1988.

(Rumelhardt *et al.*, 1986) D. E. Rumelhardt, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhardt and J. L. McClelland, editors, *Parallel Distributed Processing, Vol 1.* MIT Press, Cambridge, MA, 1986.

(Rumelhardt and McClelland , 1986) D. E. Rumelhardt, J. L. McClelland, and the PDP Reseach group. *Parallel Distributed Processing: Explorations in the microstructure of cognition, Vol 1.* MIT Press, Cambridge, MA, 1986.

(Schabes and Shieber, 1990) Yves Schabes and Stuart Shieber. Synchronous tree adjoining grammars. In *Proceedings of the 13th International Conference on Computational Linguistics*, Helsinki, 1990.

(Schabes, 1990) Yves Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars.* PhD thesis, University of Pennsylvania, Philadelphia, PA, 1990.

(Selman and Hirst, 1987) Bart Selman and Graeme Hirst. Parsing as an energy minimization problem. In Lawrence Davis, editor, *Genetic Algorithms and Simulated Annealing*, chapter 11, pages 141–154. Morgan Kaufmann Publishers, Los Altos, CA, 1987.

(Servan-Schreiber *et al.*, 1991) D. Servan-Schreiber, A. Cleeremans, and J. McClelland. Graded state machines: The representation of temporal contingencies in simple recurrent networks. *Machine Learning*, 7:161–193, 1991.

(Shastri and Ajjanagadde, 1990) Lokendra Shastri and Venkat Ajjanagadde. From simple associations to systematic reasoning: A connectionist representation of rules, variables, and dynamic bindings using temporal synchrony. Technical Report MS-CIS-90-05, University of Pennsylvania, Philadelphia, PA, 1990. Revised Jan 1992.

(Shastri and Ajjanagadde, 1993) Lokendra Shastri and Venkat Ajjanagadde. From simple associations to systematic reasoning: A connectionist representation of rules, variables, and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences*, 16, 1993.

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

(de Smedt and Kempen, 1991) Koenraad de Smedt and Gerard Kempen. Segment grammar: A formalism for incremental sentence generation. In C.L. Paris, W.R. Swartout, and W.C. Mann, editors, *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, pages 329–349. Kluwer Academic Publishers, Boston/Dordrecht/London, 1991.

(Smolensky, 1990) Paul Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46(1-2):159–216, 1990.

(Steedman, 1985) M. Steedman. Dependencies and coordination in the grammar of dutch and english. *Language*, 61:523–568, 1985.

(St. John and McClelland, 1992) Mark St. John and James McClelland. Parallel constraint satisfaction as a comprehension mechanism. In Ronan Reilly and Noel Sharkey, editors, *Connectionist Approaches to Natural Language Processing*, pages 97–136. Lawrence Erlbaum Associates, Hove, U.K., 1992.

(Steedman, 1987) Mark Steedman. Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory*, 5, 1987.

(Stevenson, 1994) Suzanne Stevenson. Competition and disambiguation in a hybrid network model of human parsing. *Journal of Psycholinguistic Research*, 23(6), 1994.

(Sun, 1992) Ron Sun. On variable binding in connectionist networks. *Connection Science*, 4(2), 1992.

(Vijay-Shanker, 1987) K. Vijay-Shanker. *A Study of Tree Adjoining Grammars*. PhD thesis, University of Pennsylvania, Philadelphia, PA, 1987.

(Vijay-Shanker, 1992) K. Vijay-Shanker. Using descriptions of trees in a tree adjoining grammar. *Computational Linguistics*, 18:481–517, 1992.