A CONNECTIONIST PARSER FOR STRUCTURE UNIFICATION GRAMMAR

James B. Henderson*
Department of Computer and Information Science
University of Pennsylvania
200 South 33rd
Philadelphia, PA 19104, USA
(henders@linc.cis.upenn.edu)

ABSTRACT

This paper presents a connectionist syntactic parser which uses Structure Unification Grammar as its grammatical framework. The parser is implemented in a connectionist architecture which stores and dynamically manipulates symbolic representations, but which can't represent arbitrary disjunction and has bounded memory. These problems can be overcome with Structure Unification Grammar's extensive use of partial descriptions.

INTRODUCTION

The similarity between connectionist models of computation and neuron computation suggests that a study of syntactic parsing in a connectionist computational architecture could lead to significant insights into ways natural language can be parsed efficiently. Unfortunately, previous investigations into connectionist parsing (Cottrell, 1989, Fanty, 1985, Selman and Hirst, 1987) have not been very successful. They cannot parse arbitrarily long sentences and have inadequate grammar representations. However, the difficulties with connectionist parsing can be overcome by adopting a different connectionist model of computation, namely that proposed by Shastri and Ajjanagadde (1990). This connectionist computational architecture differs from others in that it directly manifests the symbolic interpretation of the information it stores and manipulates. It also shares the massive parallelism, evidential reasoning ability, and neurological plausibility of other connectionist architectures. Since virtually all characterizations of natural language syntax have relied heavily on symbolic representations, this architecture is ideally suited for the investigation of syntactic parsing.

*This research was supported by ARO grant DAAL 03-89-C-0031, DARPA grant N00014-90-J-1863, NSF grant IRI 90-16592, and Ben Franklin grant 91S.3078C-1.

The computational architecture proposed by Shastri and Ajjanagadde (1990) provides a rather general purpose computing framework, but it does have significant limitations. A computing module can represent entities, store predications over those entities, and use pattern–action rules to manipulate this stored information. This form of representation is very expressive, and pattern–action rules are a general purpose way to do computation. However, this architecture has two limitations which pose difficult problems for parsing natural language. First, only a conjunction of predications can be stored. The architecture cannot represent arbitrary disjunction. This limitation implies that the parser's representation of syntactic structure must be able to leave unspecified the information which the input has not yet determined, rather than having a disjunction of more completely specified possibilities for completing the sentence. Second, the memory capacity of any module is bounded. The number of entities which can be stored is bounded by a small constant, and the number of predications per predicate is also bounded. These bounds pose problems for parsing because the syntactic structures which need to be recovered can be arbitrarily large. This problem can be solved by allowing the parser to output the syntactic structure incrementally, thus allowing the parser to forget the information which it has already output and which it no longer needs to complete the parse. This technique requires that the representation of syntactic structure be able to leave unspecified the information which has already been determined but which is no longer needed for the completion of the parse. Thus the limitations of the architecture mean that the parser's representation of syntactic structure must be able to leave unspecified both the information which the input has not yet determined and the information which is no longer needed.

In order to comply with these requirements, the parser uses Structure Unification Grammar (Henderson, 1990) as its grammatical framework. SUG is a formalization of accumulating informa-

tion about the phrase structure of a sentence until a complete description of the sentence's phrase structure tree is constructed. Its extensive use of partial descriptions makes it ideally suited for dealing with the limitations of the architecture.

This paper focuses on the parser's representation of phrase structure information and on the way the parser accumulates this information during a parse. Brief descriptions of the grammar formalism and the implementation in the connectionist architecture are also given. Except where otherwise noted, a simulation of the implementation has been written, and its grammar supports a small set of examples. A more extensive grammar is under development. SUG is clearly an adequate grammatical framework, due to its ability to straightforwardly simulate Feature Structure Based Tree Adjoining Grammar (Vijay–Shanker, 1987), as well as other formalisms (Henderson, 1990). Initial investigations suggest that the constraints imposed by the parser do not interfere with this linguistic adequacy, and more extensive empirical verification of this claim is in progress. The remainder of this paper will first give an overview of Structure Unification Grammar, then present the parser design, and finally a sketch of its implementation.

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 information is specified in partial descriptions of phrase structure trees. An SUG grammar is simply a set of these descriptions. The descriptions cannot use disjunction or negation, but their partiality makes them both flexible enough and powerful enough to state what is known and only what is known where it is known. There is also a simple abstraction operation for SUG descriptions which allows unneeded information to be forgotten, as will be discussed in the section on the parser design. In an SUG derivation, descriptions are combined by equating nodes. This way of combining descriptions is extremely flexible, thus allowing the parser to take full advantage of the flexibility of SUG descriptions, and also providing for efficient parsing strategies. The final description produced by a derivation must completely describe some phrase structure tree. This tree is the result of the derivation. The design of SUG incorporates ideas from Tree Adjoining Grammar, Description Theory (Marcus *et al.*, 1983), Combinatory Categorial Grammar, Lexical Functional Grammar, and Head–driven Phrase Structure Grammar.

An SUG grammar is a set of partial descriptions of phrase structure trees. Each SUG grammar entry simply specifies an allowable grouping of information, thus expressing the information interdependencies. The language which SUG provides for specifying these descriptions allows partiality both in the information about individual nodes, and (crucially) in the information about the structural relations between nodes. As in many formalisms, nodes are described with feature structures. The use of feature structures allows unknown characteristics of a node to be left unspecified. Nodes are divided into nonterminals, which are arbitrary feature structures, and terminals, which are atomic instances of strings. Unlike most formalisms, SUG allows the specification of the structural relations to be equally partial. For example, if a description specifies children for a node, this does not preclude that node from acquiring other children, such as modifiers. This partiality also allows grammar entries to underspecify ordering constraints between nodes, thus allowing for variations in word order. This partiality in structural information is imperative to allow incremental parsing without disjunction (Marcus *et al.*, 1983). In addition to the immediate dominance relation for specifying parent–child relationships and linear precedence for specifying ordering constraints, SUG allows chains of immediate dominance relationships to be partially specified using the dominance relation. A dominance constraint between two nodes specifies that there must be a chain of zero or more immediate dominance constraints 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. Some examples of SUG phrase structure descriptions are given in figure 1, and will be discussed below.

A complete description of a phrase structure tree is constructed from the partial descriptions in an SUG grammar by conjoining a set of grammar entries and specifying how these descriptions share nodes. More formally, an SUG derivation starts with descriptions from the grammar, and in each step conjoins a set of one or more descriptions and adds zero or more statements of equality between nonterminal nodes. The description which results from a derivation step must be satisfiable, so the feature structures of any two equated nodes must unify and the resulting structural constraints must be consistent with some phrase structure tree. The final description produced by a derivation must be a complete description of some phrase structure tree. This tree is the result of the derivation. The sentences generated by a derivation are all those terminal strings which are consistent with the ordering constraints on the resulting tree. Fig-

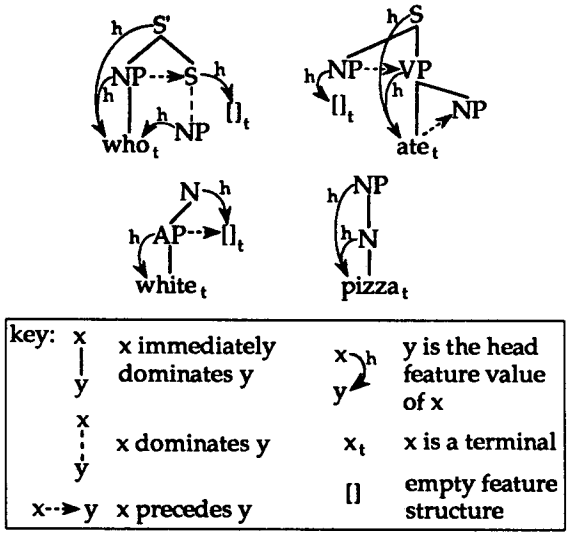

Figure 1: Example grammar entries. They can be combined to form a structure for the sentence "Who ate white pizza?".

ure 2 shows an example derivation with one step in which all grammar entries are combined and all equations are done. This definition of derivations provides a very flexible framework for investigating various parsing strategies. Any ordering of combining grammar entries and doing equations is a valid derivation. The only constraints on derivations come from the meanings of the description primitives and from the need to have a unique resulting tree. This flexibility is crucial to allow the parser to compensate for the connectionist architecture's limitations and to parse efficiently.

Because the resulting description of an SUG derivation must be both a consistent description and a complete description of some tree, an SUG grammar entry can state both what is true about the phrase structure tree and what needs to be true. For a description to be complete it must specify a single immediate dominance tree and all terminals mentioned in the description must have some (possibly empty) string specified for them. Otherwise there would be no way to determine the exact tree structure or the word for each terminal in the resulting tree. A grammar entry can express grammatical requirements by not satisfying these completion requirements locally. For example, in figure 1 the structure for "ate" has a subject node with category NP and with a terminal as the values of its *head* feature. Because this terminal does not have its word specified, this NP must equate with another NP node which does have a word for the value of its *head* feature. The unification of the two NP's feature structures will cause the equation of the two *head* terminals. In this way the structure for "ate" expresses the fact that it obligatorily subcategorizes for a subject NP. The structure for "ate" also expresses its subcategorization for an object NP, but this object is not obligatory since it does not have an underspecified terminal head. Like the subject of "ate", the root of the structure for "white" in figure 1 has an underspecified terminal head. This expresses the fact that "white" obligatorily modifies N's. The need to construct a single immediate dominance tree is used in the structure for "who" to express the need for the subcategorized S to have an NP gap. Because the dominated NP node does not have an immediate parent, it must equate with some node which has an immediate parent. The site of this equation is the gap associated with "who".

THE PARSER

The parser presented in this paper accumulates phrase structure information in the same way as does Structure Unification Grammar. It calculates SUG derivation steps using a small set of operations, and incrementally outputs the derivation as it parses. The parser is implemented in the connectionist architecture proposed by Shastri and Ajjanagadde (1990) as a special purpose module for syntactic constituent structure parsing. An SUG description is stored in the module's memory by representing nonterminal nodes as entities and all other needed information as predications over these nodes. If the parser starts to run out of memory space, then it can remove some nodes from the memory, thus forgetting all information about those nodes. The parser operations are implemented in pattern–action rules. As each word is input to the parser, one of these rules combines one of the word's grammar entries with the current description. When the parse is finished the parser checks to make sure it has produced a complete description of some phrase structure tree.

THE GRAMMARS

The grammars which are supported by the parser are a subset of those for Structure Unification Grammar. These grammars 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. Such grammar entries specify what information is known about the phrase structure of the sentence given the presence of the word, and can be used (Henderson, 1990) to simulate Lexicalized Tree Adjoining Grammar (Schabes, 1990). Nonlexical grammar entries are rooted tree fragments with no words. They can be used to express constructions like reduced relative clauses, for which no lexical information is necessary. The

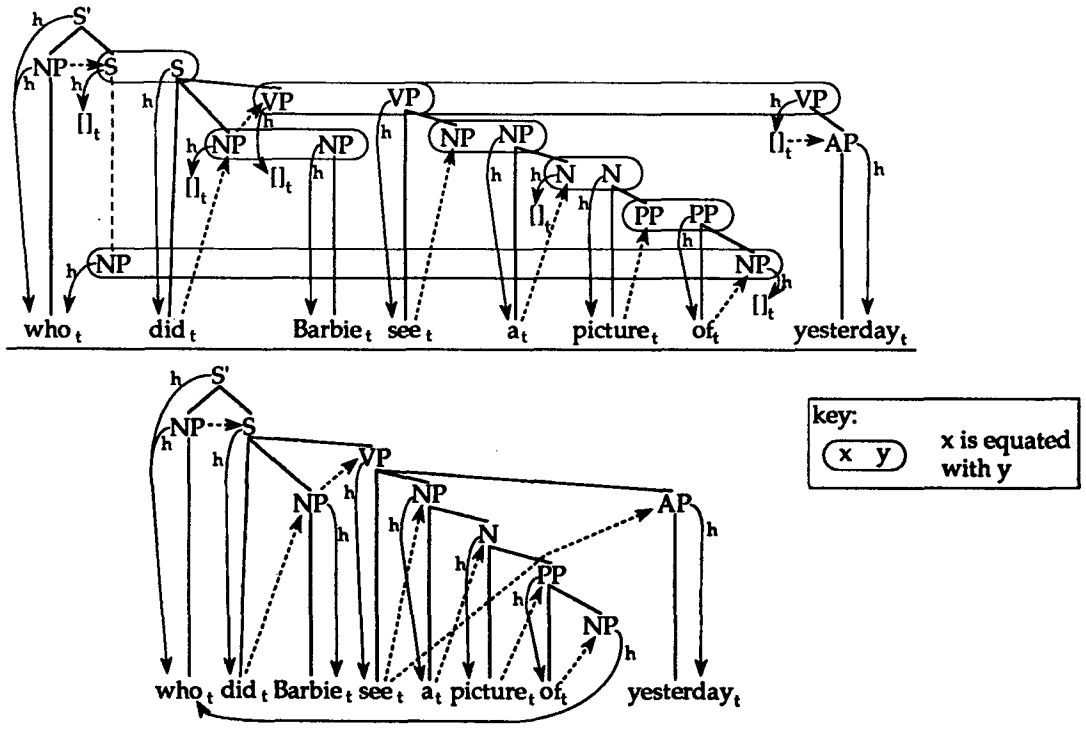

Figure 2: A derivation for the sentence "Who did Barbie see a picture of yesterday".

current mechanism the parser uses to find possible long distance dependencies requires some information about possible extractions to be specified in grammar entries, despite the fact that this information currently only has meaning at the level of the parser.

The primary limitations on the parser's ability to parse the sentences derivable with a grammar are due to the architecture's lack of disjunction and limited memory capacity. Technically, constraints on long distance dependencies are enforced by the parser's limited ability to calculate dominance relationships, but the definition of an SUG derivation could be changed to manifest these constraints. This new definition would be necessary to maintain the traditional split between competence and performance phenomena. The remaining constraints imposed at the level of the parser are traditionally treated as performance constraints. For example, the parser's bounded memory prevents it from being able to parse arbitrarily center embedded sentences or from allowing arbitrarily many phrases on the right frontier of a sentence to be modified. These are well established performance constraints on natural language (Chomsky, 1959, and many others). The lack of a disjunction operator limits the parser's ability to represent local ambiguities. This results in some locally ambiguous grammatical sentences being unparsable. The existence of such sentences for the human parser, called garden path sentences, is also well documented (Bever, 1970, among others). The representations currently used for handling local ambiguities appear to be adequate for building the constituent structure of any non-garden path sentences. The full verification of this claim awaits a study of how effectively probabilistic constraints can be used to resolve ambiguities. The work presented in this paper does not directly address the question of how ambiguities between possible predicate-argument structures are resolved. Also, the current parser is not intended to be a model of performance phenomena, although since the parser is intended to be computationally adequate, all limitations imposed by the parser must fall within the set of performance constraints on natural language.

THE PARSER DESIGN

The parser follows SUG derivations, incrementally combining a grammar entry for each word with the description built from the previous words of the sentence. Like in SUG the intermediate descriptions can specify multiple rooted tree fragments, but the parser represents such a set as a list in order to represent the ordering between terminals in the fragments. The parser begins with a description containing only an S node which needs a head. This description expresses the parser's expectation for a sentence. As each word is read, a grammar entry for that word is chosen and combined

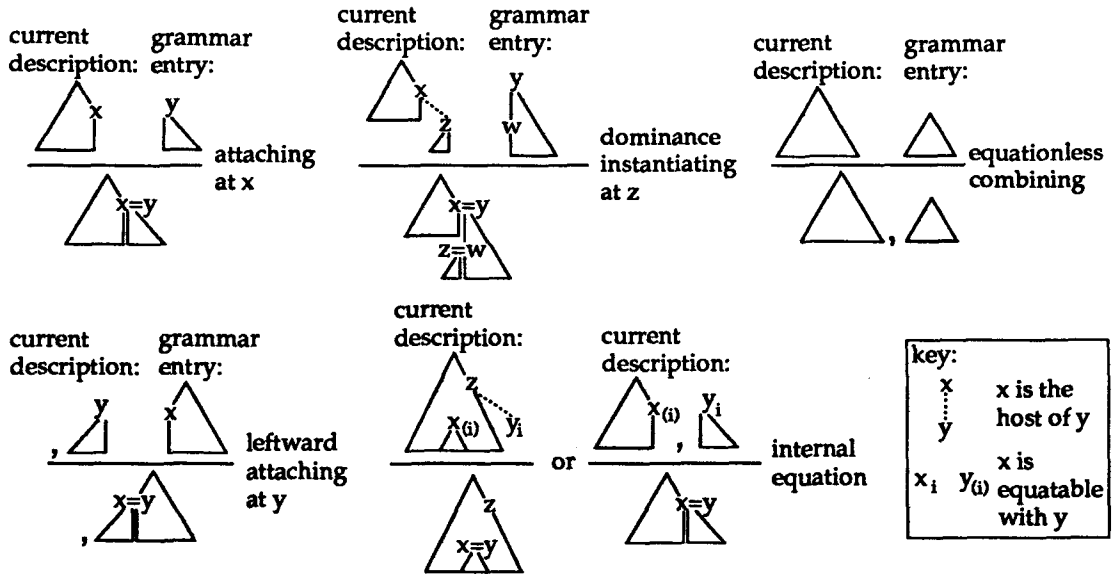

Figure 3: The operations of the parser.

with the current description using one of four combination operations. Nonlexical grammar entries can be combined with the current description at any time using the same operations. There is also an internal operation which equates two nodes already in the current description without using a grammar entry. The parser outputs each operation it does as it does them, thus providing incremental output to other language modules. After each operation the parser's representation of the current description is updated so that it fully reflects the new information added by the operation.

The five operations used by the parser are shown in figure 3. The first combination operation, called attaching, adds the grammar entry to the current description and equates the root of the grammar entry with some node already in the current description. The second, called dominance instantiating, equates a node without a parent in the current description with a node in the grammar entry, and equates the *host* of the unparented node with the root of the grammar entry. The *host* function is used in the parser's mechanism for enforcing dominance constraints, and represents the fact that the unparented node is potentially dominated by its current *host*. In the case of long distance dependencies, a node's *host* is changed to nodes further and further down in the tree in a manner similar to slash passing in Generalized Phrase Structure Grammar, but the resulting domain of possible extractions is more similar to that of Tree Adjoining Grammar. The equationless combining operation simply adds a grammar entry to the end of the tree fragment list. This operation is sometimes necessary in order to delay attachment decisions long enough to make the right choice. The leftward attaching operation equates the root of the tree fragment on the end of the list with some node in the grammar entry, as long as this root is not the initializing matrix S[1]. The one parser operation which does not involve a grammar entry is called internal equating. When the parser's representation of the current description is updated so that it fully reflects newly added information, some potential equations are calculated for nodes which do not yet have immediate parents. The internal equating operation executes one of these potential equations. There are two cases when this can occur, equating fillers with gaps and equating a root of a tree fragment with a node in the next earlier tree fragment on the list. The later is how tree fragments are removed from the list.

The bound on the number of entities which can be stored in the parser's memory requires that the parser be able to forget entities. The implementation of the parser only represents nonterminal nodes as entities. The number of nonterminals in the memory is kept low simply by forgetting nodes when the memory starts getting full, thereby also forgetting the predications over the nodes. This forgetting operation abstracts away from the existence of the forgotten node in the phrase structure. Once a node is forgotten it can no longer be equated with, so nodes which must be equated with in order for the total description to be complete can not be forgotten. Forgetting nodes may eliminate some otherwise possible parses, but it will never allow parses which violate

[1] As of this writing the implementation of the tree fragment list and these later two combination operations has been designed, but not coded in the simulation of the parser's implementation.

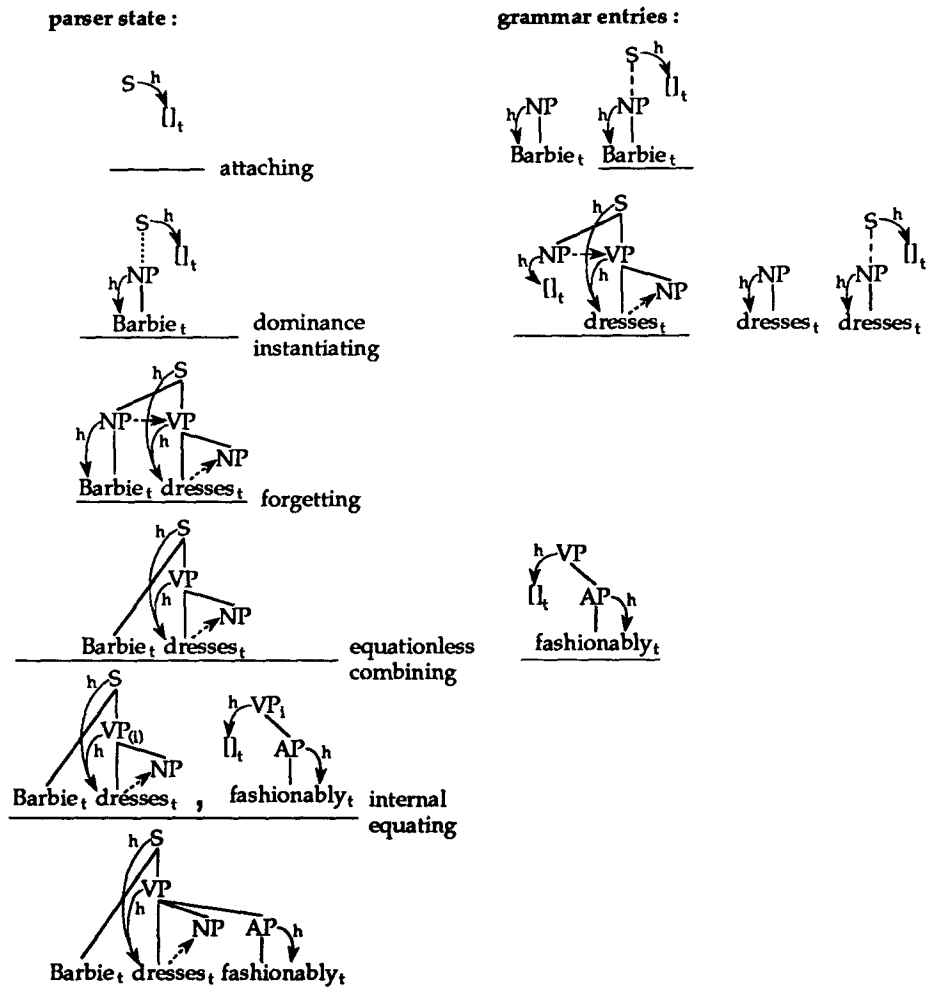

Figure 4: An example parse of "Barbie dresses fashionably".

the forgotten constraints. Any forgetting strategy can be used as long as the only eliminated parses are for readings which people do not get. Several such strategies have been proposed in the literature.

As a simple example parse consider the parse of "Barbie dresses fashionably" sketched in figure 4. The parser begins with an S which needs a head, and receives the word "Barbie". The underlined grammar entry is chosen because it can attach to the S in the current description using the attaching operation. The next word input is "dresses", and its verb grammar entry is chosen and combined with the current description using the dominance instantiating operation. In the resulting description the subject NP is no longer on the right frontier, so it will not be involved in any future equations and thus can be forgotten. Remember that the output of the parser is incremental, so forgetting the subject will not interfere with semantic interpretation. The next word input is "fashionably", which is a VP modifier. The parser could simply attach "fashionably", but for the purposes of exposition assume the parser is not sure where to attach this modifier, so it simply adds this grammar entry to the end of the tree fragment list using equationless combining. The updating rules of the parser then calculate that the VP root of this tree fragment could equate with the VP for "dresses", and it records this fact. The internal equating operation can then apply to do this equation, thereby choosing this attachment site for "fashionably". This technique can be used to delay resolving any attachment ambiguity. At this point the end of the sentence has been reached and the current description is complete, so a successful parse is signaled.

Another example which illustrates the parser's ability to use underspecification to delay disambiguation decisions is given in figure 5. The feature decomposition $\pm$A,$\pm$V is used for the major categories (N, V, A, and P) in order to allow the object of "know" to be underspecified as to whether it is of category N ([$-$A,$-$V]) or V ([$-$A,+V]). When

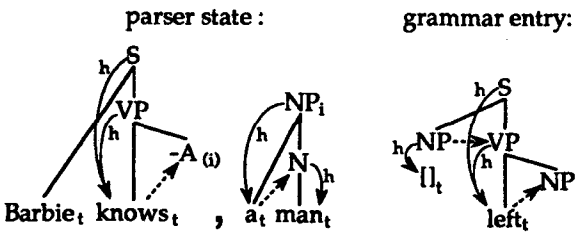

Figure 5: Delaying the resolution of the ambiguity between "Barbie knows a man." and "Barbie knows a man left."

"a man" is input the parser is not sure if it is the object of "know" or the subject of this object, so the structure for "a man" is simply added to the parser state using equationless combining. This underspecification can be maintained for as long as necessary, provided there are resources available to maintain it. If no verb is subsequently input then the NP can be equated with the $-$A node using internal equation, thus making "a man" the object of "know". If, as shown, a verb is input then leftward attaching can be used to attach "a man" as the subject of the verb, and then the verb's S node can be equated with the $-$A node to make it the object of "know". Since this parser is only concerned with constituent structure and not with predicate–argument structure, the fact that the $-$A node plays two different semantic roles in the two cases is not a problem.

THE CONNECTIONIST IMPLEMENTATION

The above parser is implemented using the connectionist computational architecture proposed by Shastri and Ajjanagadde (1990). This architecture solves the variable binding problem[2] by using units which pulse periodically, and representing different entities in different phases. Units which are storing predications about the same entity pulse synchronously, and units which are storing predications about different entities pulse in different phases. The number of distinct entities which can be stored in a module's memory at one time is determined by the width of a pulse spike and the time between periodic firings (the period). Neurologically plausible estimates of these values put the maximum number of entities in the general vicinity of 7±2. The architecture does computation with sets of units which implement pattern–action rules. When such a set of units finds its pattern in the predications in the memory, it modifies the memory contents in accordance with its action and

[2]The variable binding problem is keeping track of what predications are for what variables when more than one variable is being used.

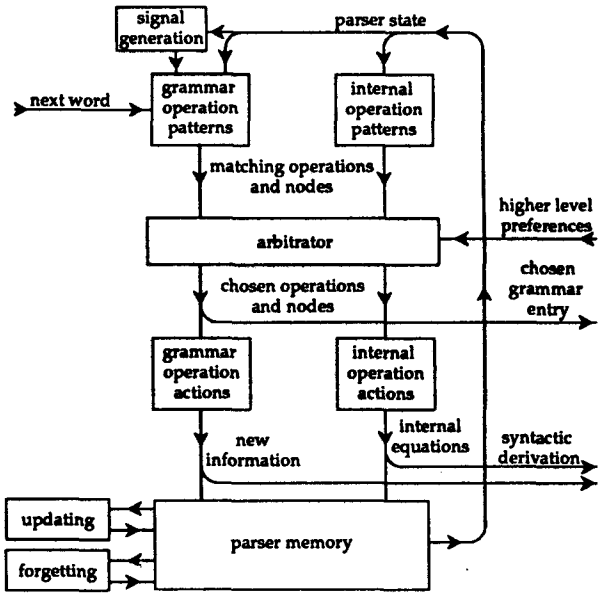

Figure 6: The architecture of the parser.

the entity(s) which matched.

This connectionist computational architecture is used to implement a special purpose module for syntactic constituent structure parsing. A diagram of the parser's architecture is shown in figure 6. This parsing module uses its memory to store information about the phrase structure description being built. Nonterminals are the entities in the memory, and predications over nonterminals are used to represent all the information the parser needs about the current description. Pattern–action rules are used to make changes to this information. Most of these rules implement the grammar. For each grammar entry there is a rule for each way of using that grammar entry in a combination operation. The patterns for these rules look for nodes in the current description where their grammar entry can be combined in their way. The actions for these rules add information to the memory so as to represent the changes to the current description which result from their combination. If the grammar entry is lexical then its rules are only activated when its word is the next word in the sentence. A general purpose connectionist arbitrator is used to choose between multiple rule pattern matches, as with other disambiguation decisions[3]. This arbitrator

[3]Because a rule's pattern matches must be communicated to the rule's action through an arbitrator, the existence and quality of a match must be specified in a single node's phase. For rules which involve more than one node, information about one of the nodes must be represented in the phase of the other node for the purposes of testing patterns. This is the purpose

weighs the preferences for the possible choices and makes a decision. This mechanism for doing disambiguation allows higher level components of the language system to influence disambiguation by adding to the preferences of the arbitrator[4]. It also allows probabilistic constraints such as lexical preferences and structural biases to be used, although these aspects of the parser design have not yet been adequately investigated. Because the parser's grammar is implemented in rules which all compute in parallel, the speed of the parser is independent of the size of the grammar. The internal equating operation is implemented with a rule that looks for pairs of nodes which have been specified as possible equations, and equates them, provided that that equation is chosen by the arbitrator. Equation is done by translating all predications for one node to the phase of the other node, then forgetting the first node. The forgetting operation is implemented with links which suppress all predications stored for the node to be forgotten. The only other rules update the parser state to fully reflects any new information added by a grammar rule. These rules act whenever they apply, and include the calculation of equatability and *host* relationships.

CONCLUSION

This paper has given an overview of a connectionist syntactic constituent structure parser which uses Structure Unification Grammar as its grammatical framework. The connectionist computational architecture which is used stores and dynamically manipulates symbolic representations, thus making it ideally suited for syntactic parsing. However, the architecture's inability to represent arbitrary disjunction and its bounded memory capacity pose problems for parsing. These difficulties can be overcome by using Structure Unification Grammar as the grammatical framework, due to SUG's extensive use of partial descriptions.

This investigation has indeed led to insights into efficient natural language parsing. This parser's speed is independent of the size of its grammar. It only uses a bounded amount of memory. Its output is incremental, monotonic, and does not include disjunction. Its disambiguation mechanism provides a parallel interface for the influence of higher level language modules. Assuming neurologically plausible timing characteristics for the computing units of the connectionist architecture, the parser's speed is roughly compatible with the speed of human speech. In the future the ability of this architecture to do evidential reasoning should allow the use of statistical information in the parser, thus making use of both grammatical and statistical approaches to language in a single framework.

of the *signal generation* box in figure 6. For all such rules, the identity of one of the nodes can be determined uniquely given the other node and the parser state. For example in the dominance instantiating operation, given the unparented node, the *host* of that node can be found because *host* is a function. This constraint on parser operations seems to have significant linguistic import, but more investigation of this possibility is necessary.

[4]In the current simulation of the parser implementation the arbitrators are controlled by the user.

REFERENCES

Bever, Thomas G (1970). The cognitive basis for linguistic structures. In J. R. Hayes, editor, *Cognition and the Development of Language.* John Wiley, New York, NY.

Chomsky, Noam (1959). On certain formal properties of grammars. *Information and Control*, 2: 137–167.

Cottrell, Garrison Weeks (1989). *A Connectionist Approach to Word Sense Disambiguation.* Morgan Kaufmann Publishers, Los Altos, CA.

Fanty, Mark (1985). Context-free parsing in connectionist networks. Technical Report TR174, University of Rochester, Rochester, NY.

Henderson, James (1990). Structure unification grammar: A unifying framework for investigating natural language. Technical Report MS-CIS-90-94, University of Pennsylvania, Philadelphia, PA.

Marcus, Mitchell; Hindle, Donald; and Fleck, Margaret (1983). D-theory: Talking about talking about trees. In *Proceedings of the 21st Annual Meeting of the ACL*, Cambridge, MA.

Schabes, Yves (1990). *Mathematical and Computational Aspects of Lexicalized Grammars.* PhD thesis, University of Pennsylvania, Philadelphia, PA.

Selman, Bart and Hirst, Graeme (1987). 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.

Shastri, Lokendra and Ajjanagadde, Venkat (1990). From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings. Technical Report MS-CIS-90-05, University of Pennsylvania, Philadelphia, PA. Revised Jan 1992.

Vijay-Shanker, K. (1987). *A Study of Tree Adjoining Grammars.* PhD thesis, University of Pennsylvania, Philadelphia, PA.