An Incremental Connectionist Phrase Structure Parser

James Henderson*
U. of Pennsylvania, Dept of Computer and Information Science
200 South 33rd
Philadelphia, PA 19104

1 Introduction

This abstract outlines a parser implemented in a connectionist model of short term memory and reasoning[1]. This connectionist architecture, proposed by Shastri in [Shastri and Ajjanagadde, 1990], preserves the symbolic interpretation of the information it stores and manipulates, but does its computations with nodes which have roughly the same computational properties as neurons. The parser recovers the phrase structure of a sentence incrementally from beginning to end and is intended to be a plausible model of human sentence processing. The formalism which defines the grammars for the parser is expressive enough to incorporate analyses from a wide variety of grammatical investigations[2]. This combination gives a theory of human syntactic processing which spans from the level of linguistic theory to the level of neuron computations.

2 The Connectionist Architecture

In order to store and manipulate information in a connectionist net quickly, the information needs to be represented in the activation of nodes, not the connections between nodes[3]. A property of an entity can be represented by having a node for the entity and a node for the property both active at the same time. However, this only permits information about one entity to be stored at any one time. The connectionist architecture used here solves this problem with nodes which, when active, fire at regular intervals. A property is predicated of an entity only if their nodes are firing synchronously. This permits multiple entities to be stored at one time by having their nodes firing in different phases. However, the number of entities is limited by the number of distinct phases which can fit in the interval between periodic firings. Such boundedness of human conscious short term memory is well documented, where it is about seven entities.

Computation using the information in the memory is done with pattern–action rules. A rule is represented as a collection of nodes which look for a temporal pattern of activation, and when it finds the pattern it modifies the memory contents. Rules can compute in parallel.

3 The Grammar Formalism

I will describe grammar entries through the examples given in figure 1. Each entry must be a rooted tree fragment. Solid lines are immediate dominance links, dashed arrows are linear precedence constraints, and dotted lines, called dominance links, specify the need for a chain of immediate dominance links. Plus subscripts on nodes designate that the node is headed and minus subscripts that the node still needs a head. The parent of a dominance link must always be an unheaded node. Node labels are feature structures, but coreference of features between nodes is not allowed. In the figure, the structure for "likes" needs heads for both its NP's, thereby expressing the subcategorization for these arguments. The structure for "white" expresses its modification of N's by having a headless root N. The structure for "who" subcategorizes for an S and specifies that somewhere within that S there must be a headless NP. These four words can combine to form a complete phrase structure tree through the appropriate equations of nodes.

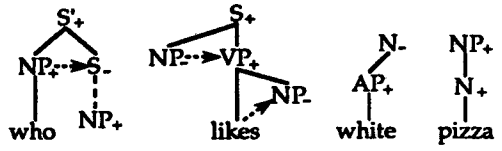

Figure 1: Example grammar entries.

There are four operations for combining adjacent

*This research was supported by DARPA grant number N0014-90-J-1863 and ARO grant number DAAL03-89-C0031PRI.

[1] As of this writing the parser has been designed, but not coded.

[2] A paper about a closely related formalism was submitted to this year's regular ACL session under the title "A CCG–Like System of Types for Trees", and an older version of the later was discussed in my masters thesis ([Henderson, 1990]), where its linguistic expressiveness is demonstrated.

[3] This section is a very brief characterization of the core system presented in [Shastri and Ajjanagadde, 1990].

tree fragments[4]. The first equates a node in the left tree with the root of the right tree. As in all equations, at least one of these nodes must be unheaded, and their node labels must unify. If the node on the left is unheaded then it is subcategorization, and if the root is unheaded then it is modification. The second combination operation satisfies a dominance link in the left tree by equating the node above the dominance link to the root of the right tree and the node below the dominance link to another node in the right tree. This is for things such as attaching embedded subjects to their verbs and filling subject gaps. The third combination operation also involves a dominance link in the left subtree, but only the parent and root are equated and the dominance relationship is passed to an unheaded node in the right tree. This is for passing traces down the tree. The last operation only involves one tree fragment. This operation satisfies a dominance link by equating the child of the link with some node which is below the node which was the original parent of this dominance link, regardless of what nodes the link has been passed to with the third operation. This is for gap filling. Limitations on the parser's ability to determine what nodes are eligible for this equation force some known constraints on long distance movement. All these operations are restricted so that linear precedence constraints are never violated.

The important properties of this formalism are its use of partiality in the specification of tree fragments and the limited domain affected by each combination operation. Partiality is essential to allow the parser to incrementally specify what it knows so far about the structure of the sentence. The fact that each combination operation is only dependent on a few nodes is important both because it simplifies the parser's rules and because nodes which are no longer going to be involved in any equations can be forgotten. Nodes must be forgotten in order to parse arbitrarily long sentences with the memory's very limited capacity.

4 The Parser

The parser builds the phrase structure for a sentence incrementally from beginning to end. After each word the short term memory contains information about the structure built so far. Pattern–action rules are then used to compute how the next word's tree can be combined with the current tree. In the memory, the nodes of the tree are the entities, and predicates are used to specify the necessary information about these nodes and the relationships between them. The tree in the memory is used as the left tree for the combination operations. The tree for the next word is the right tree. For every grammar entry there are pattern–action rules for each way it could participate in a combination. When the next word is identified its grammar entries are activated and their rules each try to find a place in the current tree where their combination can be done. The best match is chosen and that rule modifies the memory contents to represent the result of its combination. The gap filling combination operation is done with a rule which can be activated at any time. If the parser does not have enough space to store all the nodes for the new word's tree, then any node which has both a head and an immediate parent can be removed from the memory without changing any predications on other nodes. When the parse is done it succeeds if all nodes have heads and only the root doesn't have an immediate parent.

Because nodes may be forgotten before the parse of a sentence is finished, the output of the parser is not a complete phrase structure tree. The output is a list of the combinations which were done. This is isomorphic to the complete phrase structure, since the structure can be constructed from the combination information. It also provides incremental information about the progression of the parse. Such information could be used by a separate short term memory module to construct the semantic structure of the sentence in parallel with the construction of the syntactic structure.

Several characteristics make this parser interesting. Most importantly, the computational architecture it uses is compatible with what we know about the architecture of the human brain. Also, its incrementality conforms to our intuitions about the incrementality of our sentence processing, even providing for incremental semantic analysis. The parallelism in the combination process provides for both lexical ambiguity and uncertainty about what word was heard. Only further work can determine the linguistic adequacy of the parser's grammars, but work on related formalisms provides evidence of its expressiveness.

[4]This formalism has both a structural interpretation and a Categorial Grammar style interpretation. In the later interpretation these combination operations have a more natural specification. Unfortunately space prevents me discussing it here.

References

[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.

[Shastri and Ajjanagadde, 1990] Lokendra Shastri and Venkat Ajjanagadde. *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, 1990.