Report Number: WUCS-89-31

1989-01-01

Determinism and Connectionism in a Rule-Based Natural Language System

Stan C. Kwasny and Kanaan A. Faisal

The processing of Natural Language is, at the same time, natural symbolic and naturally symbolic and naturally sub-symbolic. It is symbolic because ultimately symbols play a critical role. Writing systems, for example, owe their existence to the symbolic nature of language. It is also sub-symbolic because of the nature of speech, the fuzziness of concepts, and the high degree of parallelism that is difficult to explain as a purely symbolic phenomenon. This report details a set of experiments which support the claim that Natural Language can be syntactically processed in a robust manner using a connectionist deterministic parser. The... **Read complete abstract on page 2.**

Determinism and Connectionism in a Rule-Based Natural Language System

Stan C. Kwasny and Kanaan A. Faisal

Complete Abstract:

The processing of Natural Language is, at the same time, natural symbolic and naturally symbolic and naturally sub-symbolic. It is symbolic because ultimately symbols play a critical role. Writing systems, for example, owe their existence to the symbolic nature of language. It is also sub-symbolic because of the nature of speech, the fuzziness of concepts, and the high degree of parallelism that is difficult to explain as a purely symbolic phenomenon. This report details a set of experiments which support the claim that Natural Language can be syntactically processed in a robust manner using a connectionist deterministic parser. The model is trained based on a deterministic grammar and tested with sentences which are grammatically and ones that ill-formed. Tests are also conducted with sentences containing lexically ambiguous items. Some new directions for this work are explored in the final section. The goal of fully connectionistic parsing is discussed and a detail plan for its achievements is present.

DETERMINISM AND CONNECTIONISM IN A RULE-BASED NATURAL LANGUAGE SYSTEM

Stan C. Kwasny
Kanaan A. Faisal

WUCS-89-31

Center for Intelligent Computing Systems
Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

ABSTRACT

The processing of Natural Language is, at the same time, naturally symbolic and naturally sub-symbolic. It is symbolic because ultimately symbols play a critical role. Writing systems, for example, owe their existence to the symbolic nature of language. It is also sub-symbolic because of the nature of speech, the fuzziness of concepts, and the high degree of parallelism that is difficult to explain as a purely symbolic phenomenon.

This report details a set of experiemnts which support the claim that Natural Language can be syntactically processed in a robust manner using a connectionist deterministic parser. The model is trained based on a deterministic grammar and tested with sentences which are grammatical and ones that are ill-formed. Tests are also conducted with sentences containing lexically ambiguous items.

Some new directions for this work are explored in the final section. The goal of fully connectionistic parsing is discussed and a detail plan for its achievement is presented.

The sponsors of the Center are McDonnell Douglas Corporation and Southwestern Bell Telephone Company.

1. Introduction

In the field of Artificial Intelligence, one of the primary goals is to build intelligent systems. Since the 1970s, many such systems have been most easily built symbolically as rule-based systems. In many application areas, the most popular of these are known as rule-based expert systems. These systems have become ubiquitous in their application to everything from medical diagnosis to oil exploration to stock market trading. Typically, these systems develop from consultations with recognized experts who provide knowledge and experience during the process of rule development and debugging.

Most expert systems use rules in some capacity. Usually, the rule formalism becomes the de facto ''programming language'' for the application and the rules are executed like statements in the language. Building such systems is no trivial task, often occupying the full-time efforts of dozens of people. Much of the effort is spent talking with human experts, gaining their perspective in an application domain, and teasing out the salient information that allows the expert to perform his task with a high degree of skill. Once acquired, such knowledge becomes the basis for system organization and rules that mimic the expertise.

Even with the most cooperative experts and best tools, rules are often difficult to formulate. Ad hoc mechanisms that relate premise to conclusion have been invented to capture some of the vagueness and uncertainty of the relationships articulated by the expert. For example, certainty factors were invented for this purpose. Extensive evidence showing the difficulty of knowledge acquisition can be found in a recent special issue of SIGART (Westphal & McGraw, 1989). Once formulated, rules are perhaps even more difficult to debug. Improvement of the rules is also an ongoing maintenance problem. In XSEL/XCON, for example, it is reported that 40% of the rules require changes each year to adapt to new products and for other reasons (Barker & O'Connor, 1989). Systems like TEIRESIAS (Davis, 1982) have been specifically designed to aid in the critiquing and re-formulation of rule-based systems by the experts themselves. Even with good tools, however, it may prove difficult to achieve improvements after a certain point. This may be due to fundamental limitations of the system which require fundamental changes in design to overcome.

Rules also play an important role in the processing of Natural Language. In fact, the comparison is close indeed and many of the same difficulties have been encountered. Many linguistic rule systems have been proposed, although the rules in these systems may not always appear explicitly as in rule-based expert systems. Symbolic rules tend to be an important vehicle of specification for the symbolic processing requirements of such systems. While progress in Natural Language Processing (NLP) continues to be made, why does this continue to be a largely unsolved problem? Or, more specifically, why is there no complete grammar of English? Certainly, it is not for lack of experts. Perhaps, if we agree with Winograd & Flores (1987), the answer is that ''*..computers cannot understand language*'' (p. 107). We think not. In any event, these are embarrassing questions to ask, especially in a document that describes work on Natural Language.

Noam Chomsky (1965) popularized a fundamental linguistic distinction between competence and performance in language.[2] Competence is the knowledge one possesses who knows a particular language. Performance deals with language as it is used in actual situations. Linguistics has comfortably centered on the study of competence, while virtually ignoring the more difficult issue of performance except, for example, in certain psycholinguistic studies. Developers of NLP systems have had to deal more with performance issues despite the lack of a solid theoretical foundation for doing so. This has led to some elegant Natural Language systems which nonetheless contain ad hoc components to deal with performance, if they deal with it at all.

The problems raised for rule-based expert systems as well as those mentioned for rule-based NLP could be attacked through learning. Given sufficient flexibility, a trainable system can be taught the rules articulated by the expert and then be led to adapt to those cases where the rules fail. In a similar fashion, the competence rules of a grammar can be taught in concert with the performance examples of NLP. While strictly symbolic learning may be possible, such systems typically need to invent new symbols to extend their capabilities and this imposes limitations on the approach. Connectionist (neural) networks hold promise for attacking these problems.

This report will center on examples derived from NLP, although more general lessons are to be learned. Although lessons general to rule-based systems can be derived, our discussion will be limited to NLP.

2. Determinism and NLP

The determinism hypothesis imposes important restrictions on Natural Language Processing (NLP). It states that

> *''Natural Language can be parsed by a mechanism that operates 'strictly deterministically' in that it does not simulate a nondeterministic machine...''*
>
> *(p.11, Marcus, 1980)*

It follows from this hypothesis that NLP need not depend in any fundamental way on backtracking. Furthermore, no partial structures are produced during parsing which fail to become part of the final structure.

PARSIFAL (Marcus, 1980) was the first of a number of systems to demonstrate how deterministic parsing of Natural Language can be performed using a rule-based grammar. Extensions to PARSIFAL have been researched independently including the parsing of ungrammatical sentences in PARAGRAM (Charniak 1983), the resolution of lexical ambiguities in ROBIE (Milne 1986), and the acquiring of syntactic rules from examples in LPARSIFAL (Berwick 1985).

Why have these researchers chosen to focus on extensions with these rather narrow goals? The answer, in part, lies in the general difficulty of the task of NLP and the limitations of conventional, symbolic means. We have found it beneficial to combine these

[2] Similar distinctions were certainly made much earlier by Ferdinand de Saussure (1955) between *langue* and *parole*.

tasks into one implementation which is partly symbolic and partly sub-symbolic. The results of our experiments hold implications for NLP as well as rule-based systems generally.

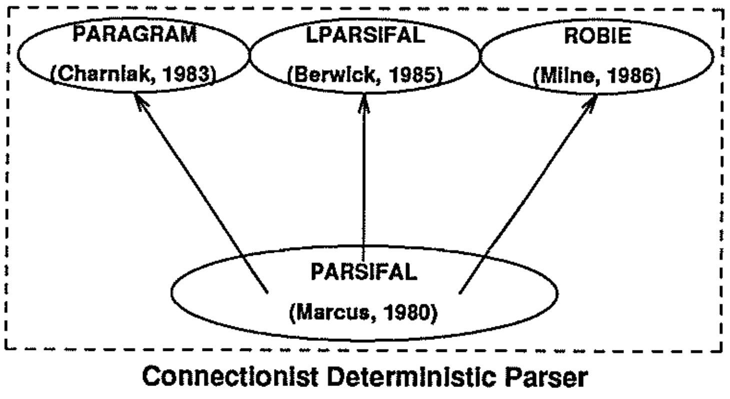

Figure 1:
Deterministic Parsing Systems

As Figure 1 illustrates, these systems share their common roots in deterministic parsing, but represent independent solutions in specific problem areas. The integration of their processing capabilities is a specific goal of our work (Kwasny 1988). The system architecture of PARSIFAL has been re-configured and the behavior of the rules and other mechanisms from these systems are being simulated using a neural network simulator. Learning in the network is achieved through backward propagation (Werbos 1974; Rumelhart et al. 1986) and an implementation of a Connectionist Deterministic Parser (CDP) has been constructed (Kwasny and Faisal 1989). Capabilities from each of the systems shown have been demonstrated.

2.1. Simulation of Deterministic NLP Systems

Each of the three NLP systems shown in Figure 1 provide improvements and enhancements to the PARSIFAL system. Since these improvements are complementary, it is desirable to combine their features into one system. Although we have chosen to pursue this goal through connectionism, could this be accomplished symbolically?

Attempting to combine these systems symbolically would require that several issues be addressed. PARAGRAM handles ungrammatical sentences by changing some of the rules and introducing a scoring mechanism. Since ROBIE adds rules for dealing with lexical ambiguity, these rules would have to be examined and made compatible with the scoring facility of PARAGRAM. Furthermore, rule-learning in LPARSIFAL would have to be extended to give it the capability of learning these new rules even though, at present, it is only capable of learning a percentage of the rules in PARSIFAL. Thus, for the symbolic approach to yield one integrated system the combinatorial interplay among the parts must be addressed. While not impossible, the building of such a system would certainly involve solving some difficult problems.

The connectionist approach has several important advantages in unifying these four systems (Kwasny et al, 1990). Much of the capability is derived directly from properties of connectionist networks and therefore no special mechanisms are required for each. Learning is a fundamental feature of the approach. Ungrammatical sentences, comparable to those of PARAGRAM, are processed as a consequence of the generalization properties of the network. These same properties also aid in the disambiguation of lexical items.

2.2. Deterministic ("Wait-and-See") Parsing

Deterministic ("Wait-and-See") Parsers (WASP) process input sentences primarily left-to-right. Determinism is accomplished by permitting a lookahead of up to three constituents with a constituent buffer designated for that purpose. To permit embedded structures, a stack is also part of the architecture. This is illustrated in Figure 2. The absence of backtracking is an important advantage in developing a connectionist-based parser since structures, once built, need never to be thrown away.

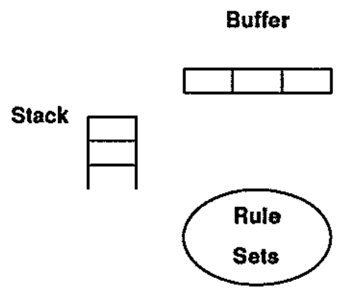

Figure 2:
WASP Components

Rules are partitioned into rule sets and a single processing step consists of selecting a rule that can fire from an active rule set. Once selected, the rule is fired and its action is performed. Rule sets are usually associated with the current (top-level) node of the structure being built. Conflicts are resolved from the static ordering (priority) of rules within the rule set. The action effects changes to the stack and buffer. After a series of processing steps, a termination rule fires and processing is terminated. The final structure is left on top of the stack.

2.3. Connectionist Deterministic Parsing

CDP combines the concepts and ideas from deterministic parsing together with the generalization and robustness of connectionist, adaptive (neural) networks. The combination of these ideas was first suggested by McClelland and Kawamoto (1986, p.317), but our approach differs somewhat from the case-based approach they advocate. A back-propagation neural network simulator, which features a logistic function that computes values in the range of −1 to +1, is being used in this work. The ultimate goal is to produce a parser that is capable of learning some reasonable facility with language, but does not fail on inputs that are only slightly different from the inputs it is designed to process.

There are important advantages to constructing rule-based systems using neural networks (Gallant, 1988). Our focus is on building a connectionist parser, but with more general issues in mind. How successfully can a connectionist parser be constructed and what are the advantages? Success clearly hinges on the careful selection of training sequences. Our experiments have examined two different approaches and compared them (Faisal and Kwasny, 1990).

The ''deductive'' strategy uses rule ''templates'' derived from the rules of a deterministic grammar. It is deductive in the sense that it is based on general knowledge in the form of rules although the resultant network is required to process specific sentences. The ''inductive'' strategy derives its training sequence from coded examples of sentence processing. It is inductive in the sense that it is based on traces of the processing of specific sentences but is required to generalize to a wider range of examples. The goal of both deductive and inductive learning is to produce a network capable of mimicking the rules or sentences on which its training is based and to do so in a way that generalizes to many additional cases. Once initial learning has been accomplished, simulation experiments can be performed to examine the generalization capabilities of the resulting networks.

In our implementation, a moderate-sized grammar developed from PARSIFAL is used for training. The grammar used in CDP is capable of processing a variety of sentence forms which end with a final punctuation mark. Simple declarative sentences, yes-no questions, imperative sentences, and passives are permitted by the grammar. The model actually receives as input a canonical representations of each word in the sentence in a form that could be produced by a simple lexicon. Such a lexicon is not part of the model in its present form.

Experiments are conducted to determine the effectiveness of training and to investigate whether the connectionist network generalizes properly to ungrammatical and lexically ambiguous cases. In comparisons with the other deterministic parsing systems, CDP performs favorably. Much of the performance depends on the extent and nature of the training, of course, but our results show that through proper training a connectionist network can indeed exhibit the same behavioral effect as the rules. Furthermore, once trained, the network is efficient, both in terms of representation and execution.

Deductive training generally performs well on all generalization tasks and outperforms inductive training by scoring generally higher on all experiments. Reasons for this include the specificity of the inductive training data as well as the lack of a large amount of training data in the inductive case required to provide sufficient variety. These training differences will be discussed later in this report.

3. Connectionism and NLP

Any plausible model of language processing should permit alternative linguistic structures to compete while inputs are processed left-to-right. Computer models based on backtracking (e.g., Augmented Transition Networks (ATNs) or Definite Clause Grammars (DCGs)) do not adequately capture the competitive nature of sentence processing. Furthermore, there is no evidence from human experiments that any conscious reprocessing of inputs is routinely performed. The lone exception is perhaps for ''garden path'' sentences.

A good example of competition can be found in the TRACE model of speech perception (McClelland and Elman, 1986). In that work, competing interpretations of the pseudo-speech feature vectors are proposed and activation levels rise or fall as each potential interpretation is supported or contradicted. Parsers should permit syntax and other levels of processing to aid in resolving lexical ambiguities just as ambiguous phonemes were resolved in TRACE.

In most neural network or connectionist parsers, grammar rules are processed into a network of units connected with excitatory and inhibitory links. The number of units required to realize a given grammar is a function of the maximum input sentence length and the complexity of the grammar. Hence, a limitation is introduced on the number of elements that can be present in the input. Sentences are processed within such a framework by presenting them, possibly in a simulated left-to-right fashion, at the input side of the network and activations are permitted to spread through the network (see Cottrell, 1985a; Fanty, 1985; Waltz and Pollack, 1985). Alternatively, a stochastic method, such as simulated annealing, is used (see Selman and Hirst, 1985).

Classically, symbolic parsers process inputs iteratively from an unbounded stream of input. Neural network parsers typically do not work iteratively and have limits imposed artificially on the length of the sentence. There is, however, work underway on neural network iteration mechanisms that could be used in parsers of natural language. For examples of work on the mechanisms of iteration in neural networks, see Williams and Zipser, (1988) or Servan-Schreiber, Cleeremans, and McClelland (1988).

In classic approaches, natural language processing by computer is performed under the direction of a set of grammar rules. These are often executed as if following instructions in a program. If we are modeling human sentence processing, then this method is incorrect. Rules should be permitted to play an advisory role only — that is, as descriptions of typical situations and not as prescriptions for precise processing. Control in the application of a rule or variant of a rule should be determined jointly as a data-driven and expectation-driven process. An alternate view can be seen in recent work aimed at using fixed resources for NLP by exchanging bounded lookahead for bounded backtracking and using local parallel processing in matching fast register vectors (Blank, 1989). While this work is done symbolically, there may be potential for developing a connectionist model from it.

Symbolic rules are an essential part of most linguistic accounts at virtually all levels of processing, from speech signal to semantics. But systems based literally on rules tend to be brittle since there is no direct way to process linguistic forms that do not strictly adhere to the pre-conceived rules. If a complete set of rules for all meaningful English forms existed, then this might be satisfactory. But no such set of rules exists, nor does it seem desirable or even possible to construct such a set. Furthermore, the rules would have a difficult time capturing "degrees of grammaticalness" as described by Chomsky (1965)[3].

Another consequence of a rule-based grammar is a pragmatic one. The acquisition of new grammar rules often require tedious re-tuning of existing rules. Rarely can a rule be added to the grammar without it affecting and being affected by other rules in the grammar. To the credit of their creators, some grammars have been continually refined over a period of years, even decades, in an attempt to more accurately depict the processing requirements of English. The only solution to this problem in a practical and realistic manner is through learning.

3.1. Learning a Rule-Based Grammar

A deterministic parser applies rules to a stack and buffer of constituents to generate and perform actions on those structures. One of its primary features, as mentioned earlier, is that it does not backtrack, but proceeds forward in its processing never building structures which are later discarded.

Training of CDP proceeds by presenting patterns to the network and teaching it to respond with an appropriate action using backpropagation. The input patterns represent encodings of the buffer positions and the top of the stack from the deterministic parser. The output of the network contains a series of units representing actions to be performed during processing and judged in a winner-take-all fashion. Network convergence is observed once the network can achieve a perfect score on the training patterns themselves and the error measure has decreased to an acceptable level (set as a parameter).

All weights in the network are initialized to random values between -0.3 and +0.3. Once the network is trained, the weights are stored in a file so that various experiments can be performed. A sentence is parsed by iteratively presenting the network with coded inputs and performing the action specified by the network. Each sentence receives a score representing the overall average strength of responses during processing. The score for each processing step is computed as the reciprocal of the error for that step. The error is computed as the Euclidean distance between the actual output and an idealized output consisting of a **-1** value for every output unit except the winning unit which has a **+1** value. The errors for each step are summed and averaged over the number of steps. The average strength is the reciprocal of the average error per step.

[3] There have been several expressions of this idea in the literature. Several psycholinguistic studies have attempted to measure the reality of this notion, both from a use as well as an interpretation perspective. We select Chomsky as an important reference and one that illustrates a classic viewpoint.

There are two distinct approaches to training a network to parse sentences. Each of these training strategies result in a slightly different version of CDP. The difference lies in the nature of the training patterns presented. One approach uses rule templates, training patterns derived from the rules. This type of learning is deductive in the sense that a very general form of each rule is learned from which the parser must derive actions specific to individual cases. The second approach uses training data derived from sentence processing traces. This form of training is inductive in the sense that the parser must arrive at general patterns of performance from the specific instances presented.

3.2. Deductive Learning

Each grammar rule is coded as a training template which is a list of feature values, but templates are not grouped into rule packets. In general, each constituent is represented by an ordered feature vector in which one or more values is ON(**+1**) for features of the form and all other values are either OFF(**-1**) or DO NOT CARE (**?**). A rule template is instantiated by randomly changing **?** to **+1** or **-1**. Thus, each template represents many training patterns and each training epoch is slightly different. During training, the network learns the inputs upon which it can rely in constructing its answers and which inputs it can ignore.

The probability of a **?** becoming a **+1** or **-1** is equal and set at 0.5. Each rule template containing n **?**'s can generate up to 2^n unique training cases. Some rule templates have over 30 **?**'s which means they represent approximately 10^9 training cases. It is obviously impossible to test the performance of all these cases, so a zero is substituted for each **?** in the rule template to provide testing patterns. While in actual processing a zero activation level for a feature will never be encountered, zero is a good test since it represents the mean of the range of values seen during training.

Grammar rules are coded into rule templates by concatenating the feature vectors of the component constituents from the stack and buffer. Each grammar rule takes the following form:

{ <Stack> <1st Item> <2nd Item> <3rd Item> → Action }

For example, a rule for Yes/No questions would be written:

{ *< S node > < "have" > < NP > < VERB, -en > → Switch 1st and 2nd items* }

while a rule for imperative sentences would be written:

{ *< S node > < "have" > < NP > < VERB, inf > → Insert YOU* }

By replacing each constituent with its coding, a rule template is created. In the two rules above, rule templates are created with a **?** value for many of the specific verb features of the initial form "have" in each rule, but are carefully coded for the differences in the third buffer position where the primary differences lie. Because different actions are required, these are also coded to have different teaching values during training.

A slightly modified version of the grammar from appendix C of Marcus (1980) was used as a basis for all deductive training experiments in this paper. This appendix includes the rules specifically discussed by Marcus in building his case for deterministic parsing and can be taken as representative of the mechanisms involved. To assure good performance by the network, training has ranged from 50,000 to 200,000 presentations cycling through training cases generated from the rule templates. Once training is complete, the parser that uses the network should correctly parse exactly those sentences that the original rules can parse. In this way, the rules of PARSIFAL can be simulated in a deductively-trained CDP.

3.3. Inductive Learning

A second type of training for the network uses training patterns derived from traces of the situations encountered and actions performed during the processing of actual sentences. This processing is guided by application of the rules of a deterministic grammar as before. PARSIFAL is simulated in this way and the task of the inductively-trained CDP parallels that of LPARSIFAL. In LPARSIFAL the object is to learn (symbolic) grammar rules from examples of correct sentences. The success of this task is gauged by directly comparing the rules learned to those of PARSIFAL.

In CDP inductive training requires the network to exhibit the correct rule-following behavior after being trained with a sample of sentence traces. Training occurs through the mechanism of backward propagation. No symbolic rules are learned as such, but the behavioral characteristics of the rules are captured within the parameters of the network. Furthermore, the behavior is guaranteed to approximate the behavior required in the sample training sentences as closely as desired, depending on the convergence rate and the quantity of training employed.

CDP, therefore, can exhibit different properties depending on the patterns used in training. Inductive learning is a much more tedious process since much more data is required as compared to that required for deductive training. Also, the range of sentence types handled depends greatly on the completeness of the examples presented. Deductive training imposes an ordering on the training patterns that assures a completeness which is difficult to achieve with inductive training, but inductive training patterns reflect the frequency of rule occurrences seen in actual sentence processing.

Inductive learning begins with training data derived as ''sentence traces'' of deterministic parsing steps. Training proceeds by presenting patterns of these steps to the network and teaching it to respond with an appropriate action. A small set of positive sentence examples were traced which resulted in 64 unique training patterns. These were used for all inductive experiments in this paper.

This approach can be compared with LPARSIFAL which attempts to learn PARSIFAL rules from examples of *positive evidence* (i.e., grammatical sentences). LPARSIFAL starts with a small set of rules and gradually builds up new rules. In effect, the system is inductively learning the grammar rules from sentence examples. The target for learning in LPARSIFAL is the PARSIFAL grammar. LPARSIFAL requires several hundred sentences to acquire approximately 70% of the parsing rules originally hand-written for the Marcus parser. In our experiments, the network exhibited better than 70% coverage of our rule-based grammar after training on a small number of traces.

3.4. Architecture of CDP

CDP, therefore, is composed of a connectionist network trained using backward propagation from rule templates which are derived from a deterministic grammar. As shown above, rule templates are intermediate between symbolic rules and the training patterns required by the network. Each rule template typically represents a large number of patterns. They serve to relate situations that occur during parsing with the action deemed appropriate for that situation. Actions in CDP are performed symbolically on traditional data structures which are also maintained symbolically.

As Figure 3 illustrates, CDP is organized into a symbolic component and a sub-symbolic component. The latter component is implemented as a numeric simulation of an adaptive neural network. The symbolic and numeric components cooperate in a tightly coupled manner since there are proven advantages to this type of organization (Kitzmiller and Kowalik 1987). For CDP, the advantages are performance and robustness.

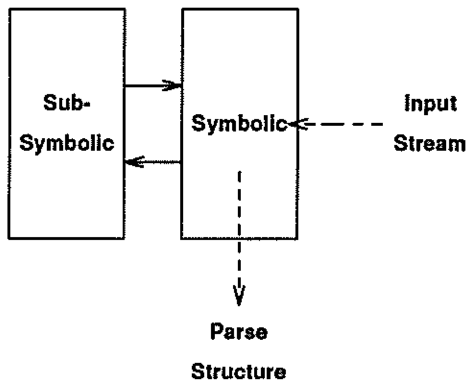

Figure 3:
CDP System Organization

The symbolic component manages the input sentence and the flow of constituents into the lookahead buffer, coding them as required for the input level of the network in the sub-symbolic component. On the return side, it evaluates the activations of the output units, decides which action to perform, and performs that action, potentially modifying the stack and buffer in the process.

The responsibility of the sub-symbolic component, therefore, is to examine the contents of the buffer and stack and yield a preference for a specific action. These preferences are garnered from many iterations of back-propagation learning with instances of the rule templates. Learning itself occurs off-line and is a time-consuming process, but once learned the processing times for the system are excellent. Computations need only flow in one direction in the network. The feed-forward multiplication of weights and computation of activation levels for individual units produce the pattern of activation on the output level. Activation of output units is interpreted in a winner-take-all manner, with the highest activated unit determining the action to be taken.

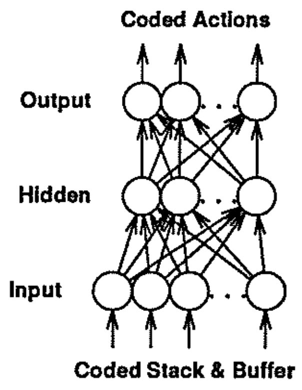

Figure 4:
Sub-Symbolic Component

In the set of experiments described here, the network has a three-layer architecture as illustrated in Figure 4, with 35 input units, 20 hidden units, and 20 output units. Each input pattern consists of three feature vectors from the buffer items and one stack vector. Each vector activates 14 input units in a pattern vector representing a word or constituent of the sentence. The stack vector activates seven units representing the current node on the stack. In our simplified version of the grammar, only two items are coded from the buffer and thus 35 input units are sufficient. One hidden layer has proven sufficient in all of these experiments. The output layer represents the 20 possible actions that can be performed on each iteration of processing.

During sentence processing, the network is presented with encodings of the buffer and the top of the stack. What the model actually sees as input is not the raw sentence but a canonical representation of each word in the sentence in a form that could be produced by a simple lexicon, although such a lexicon is not part of the model in its present form. The network produces the action to be taken which is then performed. If the action creates a vacancy in the buffer and if more of the sentence is left to be processed then the next sentence component is moved into the buffer. The process then repeats until a stop action is performed, usually when the buffer becomes empty. Iteration over the input stream is achieved in this fashion.

Figure 5 shows the nature of the processing. When a sentence form like ''John have should scheduled the meeting'' appears in the input stream, the first three constituents fill the buffer. These contents along with the contents of the top of the stack are encoded and presented to the network. The network, in turn, produces a single action which is then executed symbolically, yielding changes in the buffer and stack. This process repeats until a stop action is performed at which time the resultant parse structure is left on top of the stack.

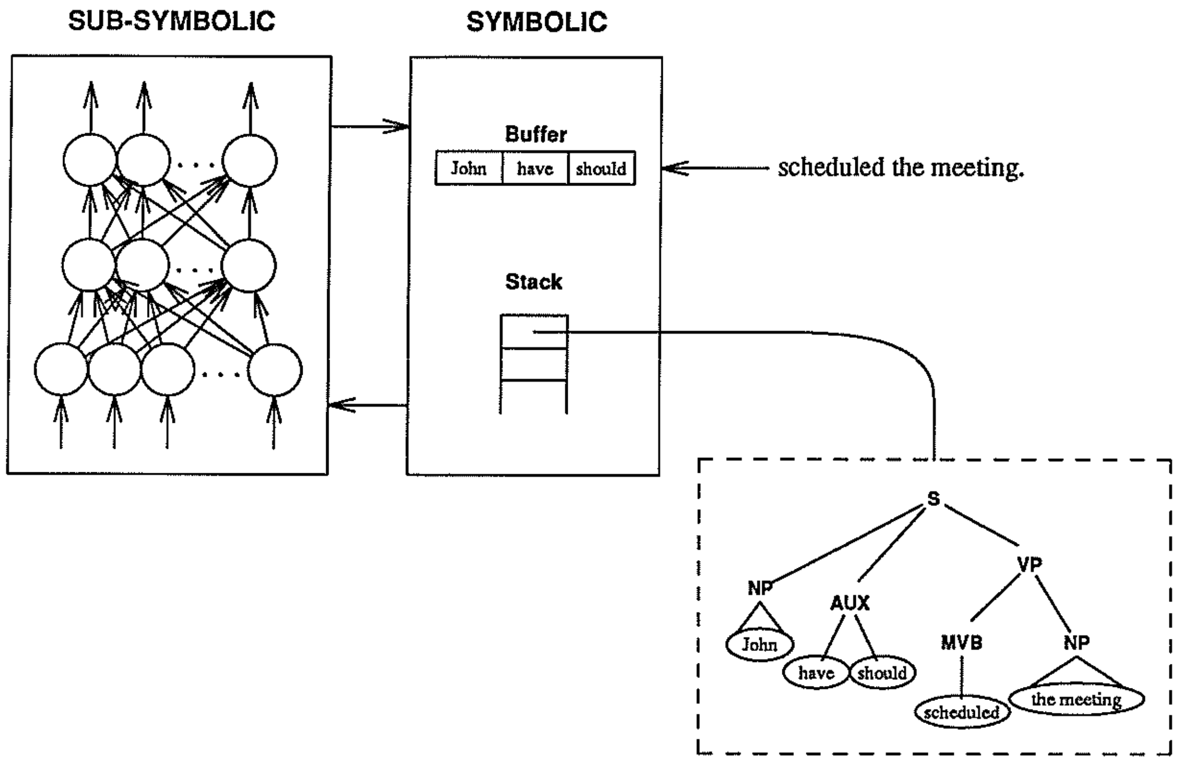

Figure 5:
CDP System Overview

4. Performance

CDP is capable of successfully processing a variety of simple sentence forms such as simple declarative sentence, simple passive, and imperative sentences as well as yes-no questions. For test and comparison between the inductive and deductive CDPs, several sentences were coded that would parse correctly by the rules of the deterministic parser. Also, several mildly ungrammatical and lexical ambiguous sentences were coded to determine if the network would generalize in any useful way. The objective was to test if syntactic context could aid in resolving such problems.

4.1. Parsing Grammatical Sentences

Experimentation with grammatical sentences demonstrates the ability of CDP to perform as PARSIFAL. Earlier we mentioned that testing from the rule templates is possible by changing the **?** to a zero value. Here we examine the performance of CDP with actual sentences. Thus, the claim that CDP simulates both PARSIFAL and LPARSIFAL is substantiated.

Grammatical sentences, by our definition, are those which parse correctly in the rule-based grammar from which we derived the training set. Table 1 shows several examples of grammatical sentences which are parsed successfully along with their response strengths in both deductive and inductive learning. These strengths are

TABLE 1
Grammatical Sentences Used in Testing

	Sentence Form	Deductive Avg. Strength	Inductive Avg. Strength
(1)	John should have scheduled the meeting.	283.3	84.7
(2)	John has scheduled the meeting for Monday.	179.3	84.2
(3)	Has John scheduled the meeting?	132.2	64.4
(4)	John is scheduling the meeting.	294.4	83.5
(5)	The boy did hit Jack.	298.2	76.2
(6)	Schedule the meeting.	236.2	67.8
(7)	Mary is kissed.	276.1	84.9
(8)	Tom hit(v) Mary.	485.0	80.3
(9)	Tom will(aux) hit(v) Mary.	547.5	78.7
(10)	They can(v) fish(np).	485.0	80.3
(11)	They can(aux) fish(v).	598.2	76.8

computed as the reciprocal of the average error per processing step for each sentence and reflect the certainty with which individual actions for building structures are being selected.

Each example shows a relatively high average strength value, indicating that the training data has been learned well. Also, the deductive average strength value is higher, in almost all cases, than the corresponding inductive average strength. Although comparisons are difficult to make due to variations in the number of unique training patterns and other factors, the deductively-trained network exhibits uniformly more definitive decisions than the inductively-trained network.

Many of these sentences were used in testing the deterministic parsing systems described earlier. Parse trees are developed which are identical with ones produced by those systems. Sentences (8)-(11), which contain ambiguous words, are given unambiguously with lexical choices provided in parentheses.

Capabilities described thus far have only duplicated what can be done symbolically rather comfortably. Of course, the feedforward network allows very fast decision-making due to the nature of the model. But what other features does the model possess? Importantly, how robust is the processing?

4.2. Parsing Ungrammatical Sentences

PARAGRAM extends PARSIFAL to handle ungrammatical sentences. This is accomplished by considering all rules in parallel and scoring each test performed on the left-hand side of a rule according to predefined weights. The rule with the best score fires. In this way, processing will always have some rule to fire. Reported experimentation with PARAGRAM shows this to be an effective method of stretching the inherent capabilities of the grammar.

For CDP, an important test of its generalization capabilities is its response to novel sentences. This is strictly dependent upon its training experiences since in deductive training no relaxation rules (Kwasny and Sondheimer 1981), meta-rules (Weischedel and Sondheimer 1983), or other special mechanisms were added to the original grammar rules to handle ungrammatical cases and in inductive training no ungrammatical sentences were used. Experiments consist of testing a few ungrammatical, but meaningful, sentences that are close to the training data and within the scope of our encoding.

Selected ungrammatical sentences are properly processed as a direct result of the generalization properties of learning. When presented an ungrammatical sentence, that is one for which the rules of the grammar would fail to find a parse, the network automatically relates the situations arising during parsing to similar situations on which it has been trained. The tendency is for it to select the closest situation. Very often, this generates precisely the response required to relate the deviant form to one that is not.

TABLE 2
Ungrammatical Sentences Used in Testing

	Sentence Form	Deductive Avg. Strength	Inductive Avg. Strength
(12)	*John have should scheduled the meeting.	25.1	6.6
(13)	*Has John schedule the meeting?	38.1	18.2
(14)	*John is schedule the meeting.	4.7	4.9†
(15)	*The boy did hitting Jack.	26.6	7.5‡

In Table 2, a few ungrammatical sentences are shown that were tested. These were chosen because they are similar to the training data and within the scope of our encoding. These examples have produced reasonable structures when presented to our system. They are shown in the table with their response strengths. Note that overall average strength is lower for ungrammatical sentences when compared to similar grammatical ones.

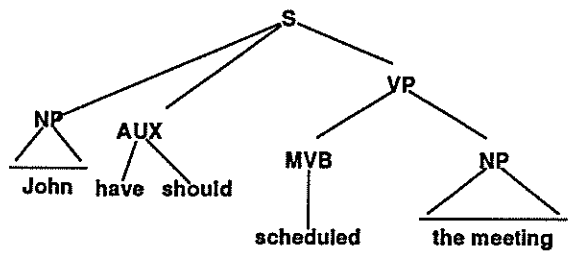

Figure 6:
Parse of "*John have should scheduled the meeting."

In sentence (12), for example, the structure produced was identical to that produced while parsing sentence (1), but with lower strength in the inductive case. The only difference is that the two auxiliary verbs, *have* and *should*, were reversed as shown in Figure 6. Sentence (13) contains a disagreement between the auxiliary *has* and the main verb *schedule* and yet the comparable grammatical sentence (3) parsed identically in both approaches, but with lower strength again in the inductive approach.

Sentence (14) can be compared with sentence (4). In the deductive case, a structure similar to that built for sentence (4) is indeed constructed. However, in the inductive case (marked with †), the network attempts to process 'is' as if it were indicating the passive tense. Although this is incorrect for this sentence, it is not an unreasonable choice.

Sentence (15) can be compared with sentence (5), but there is not one clear choice in how the sentence should appear if grammatical. The deductive-trained network processes sentence (15) as sentence (5), while the inductive result (marked with ‡) shows the sentence processed as if it were progressive tense ('The boy is hitting Jack'). In PARAGRAM, a nonsensical parse structure is produced for sentence (15), as reported by Charniak (p. 137). The problems with using a syntax-based approach to handling ungrammatical sentences are well-known (see, for example, Kwasny, 1980).

TABLE 3
Lexically Ambiguous Sentences Used in Testing

	Sentence Form	Deductive Avg. Strength	Inductive Avg. Strength
(16)	They <can> fish.	4.5	2.6
(17)	They can <fish>.	172.2	4.9
(18)	<Will> he go?	83.6	14.3
(19)	Tom <will> hit Mary.	118.7	19.9
(20)	Tom <hit> Mary.	39.0	2.5

4.3. Lexical Ambiguity

ROBIE extends PARSIFAL to address issues of lexical ambiguity. ROBIE requires additional rules and lexical features to handle these items properly. In the deterministic approach, it is essential that lexical items be properly disambiguated to permit processing to proceed without backtracking.

Thus, in another set of experiments with CDP, the parser was tested for this. Normal sentences were presented, except that selected words were coded ambiguously (here indicated by angle brackets < > around the word) to represent an ambiguously stored word from the lexicon. Some of these sentences are shown in Table 3. The numbers again indicate the average strength for each sentence. In the cases shown, the lexically ambiguous words were correctly interpreted and reasonable structures resulted.

CDP utilizes syntactic context to resolve these ambiguities. Again, the generalization capability of the network automatically works to relate novel situations to its training cases. For lexically ambiguous situations, some inputs may contain features which confuse its identity as expected by the parser. The context provided by the buffer and stack of the deterministic parser has proven to be sufficient to aid in resolving many ambiguities. An important fact is that, as before, no additional rules or mechanisms are required to provide this capability.

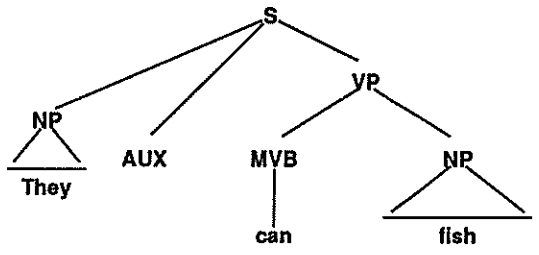

Figure 7:
Parse of "They can(v) fish(np)"

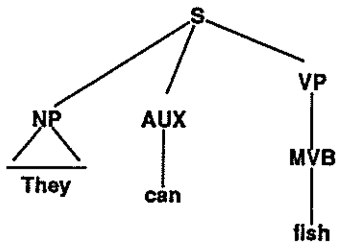

Figure 8:
Parse of "They can(aux) fish(v)"

For example, sentence (16) presents *can* ambiguously as an auxiliary, modal, and main verb, while *fish* is presented uniquely as an NP. *Can* is processed as the main verb of the sentence and resulted in the same structure as sentence (10) of Table 1. This is shown in Figure 7. Here, each word is presented unambiguously with *can* coded as a verb and *fish* coded as an NP. The same structure results in each case, with the average strength level much higher in the unambiguous case. Likewise, sentence (17), by coding *fish* ambiguously as a verb/NP and coding *can* uniquely as an auxiliary verb, produced the same structure as sentence (11). This is the structure in Figure 8.

Sentence (18) contains the word *will* coded ambiguously as an NP and an auxiliary, modal verb. In the context of the sentence, it is clearly being used as a modal auxiliary and the parser treats it that way. A similar result was obtained for sentence (19). In sentence (20), *hit* is coded to be ambiguous between an NP (as in playing cards) and a verb. The network correctly identifies it as the main verb of the sentence.

In the cases shown, the lexically ambiguous words were disambiguated and reasonable structures resulted. Note that the overall average strengths were lower than comparable grammatical sentences discussed, as expected. Also, the deductive average strength value is higher than inductive average strength.

4.4. Deductive vs. Inductive Performance

While deductive training exhibits better performance than inductive training for all tasks, there are tradeoffs in the two approaches. Deductive training requires rules as the basis for rule templates while inductive training requires a large amount of data to be successful. Fortunately there is a middle ground which allows mixtures of the two training strategies. Training can be performed using rule templates as well as patterns based on sentence traces. In a recent set of experiments in which the two types of training data were combined, the network was capable of generalizing in ways similar to deductive learning, but also showed particularly good performance on the specific cases reflected in the inductive data.

What does this mean for rule-based expert systems? Where knowledge naturally exists in rule form and such rules can be reliably stated, rule templates can be formed which generate appropriate training cases. However, where knowledge only exists in the form of anecdotal cases, it can be expressed in the form of inductive training patterns. As new cases are discovered for which the rules do not apply, inductive data can be easily constructed and the network re-trained. Contrast this with the typical rule-based expert system in which each new rule may require re-thinking an entire set of existing rules.

Our work has shown some of the trade-offs between deductive and inductive learning. Both have a place in the construction of a neural network designed to perform complex rule-based tasks such as parsing. Deductive learning can be compared to the idealized form of learning one would typically get from textbooks, while the analogy to inductive learning is that which comes through practical experiences.

4.5. Scaling Up

How well do these ideas for a connectionist parsing system scale up to a larger grammar? The prospects for successful scaling up are very good. A grammar with more than 70 rules has been constructed and network convergence has been observed. The rules represent some important new capabilities, including the processing of embedded sentences and processing performed by attention shifting rules in PARSIFAL. These new results are being evaluated and will be presented in a forthcoming PhD thesis (Faisal, 1989).

5. Toward Fully Connectionistic Parsing

While CDP demonstrates the ease with which flexible, robust parsing can be achieved, it does not represent the ultimate goal of this work. One weakness in the CDP approach is the requirement that a set of grammar rules exist for deductive training or that a large set of examples exist for inductive training. In reality, one would like to train

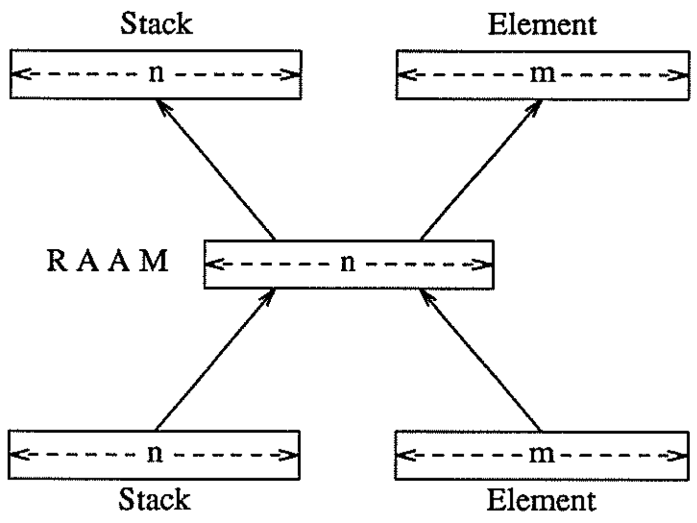

Figure 9a: Stack Recursive Auto-Associative Memory (RAAM) Training

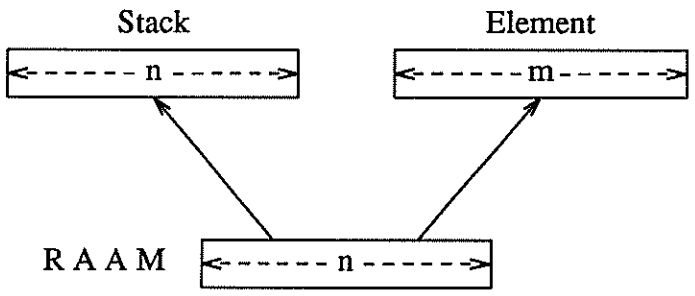

Figure 9b: Stack RAAM ''POP''

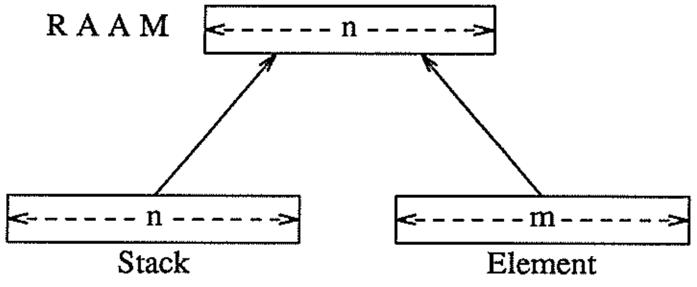

Figure 9c: Stack RAAM ''PUSH''

the network extensionally with examples of inputs and their corresponding structures. This approach is consistent with what has been called "extensional programming."

This plan requires that the network itself be capable of representing both a stack and a tree, since these structures would no longer be maintained symbolically as in CDP. It must represent a stack because stacks are essential to processing embedded structures. It must represent a tree if indeed the process of parsing is to produce a parse tree or hierarchical structure. At first, such requirements seem difficult to meet without severely restricting the parser produced. Indeed, as mentioned earlier, others who have attempted to build connectionistic parsers have accepted limitations such as bounded sentence length or the requirement that an entire sentence be presented at one time as input. These restrictions contradict the iterative nature of the process. Fortunately, some recent work has provided the necessary insight.

5.1. Recursive Auto-Associative Memory

Pollack (1989) describes a method for implementing arbitrary stacks and trees in a recursive auto-associative memory (RAAM). The idea is to provide neural network circuitry to perform the basic stack operations of push and pop. The pop operation necessarily guarantees to extract the last element pushed onto the stack. The push operation guarantees to compress the stack and new element into a stack that pop can operate upon.

The trick is in the training. As Figure 9a shows, both push and pop are trained in concert so that they can respond as required. Training develops a RAAM at the hidden layer of a back-prop network which stores elements like a stack and can be presented along with a new element to construct yet another generation of RAAM. This can continue to an arbitrary depth. With each round of training, both the push and the pop are guaranteed to reciprocate in adding or removing elements. As shown in Figures 9b and 9c, the individual push or pop circuitry can be used independently to create or destroy the stack. The illustration assumes that m units are required for each element and n units are sufficient for the stack RAAM.

Similarly, binary trees can be represented to an arbitrary depth. A slight generalization of the stack RAAM shows that if both input objects represent subtrees, then the combination will effectively be a binary tree, provided it can again be decomposed into its parts. Through proper training sequences, this can be accomplished, as report by Pollack. Figures 10a-10c illustrate this, showing m units for each tree or subtree.

One might imagine an additional variation in which n-ary trees are represented for known n. Generalization along these lines is not absolutely necessary, of course, as binary trees have a natural correspondence with more general tree structures. What should be clear from this discussion is that a network, properly configured and properly trained, can be used to represent these traditional data structures.

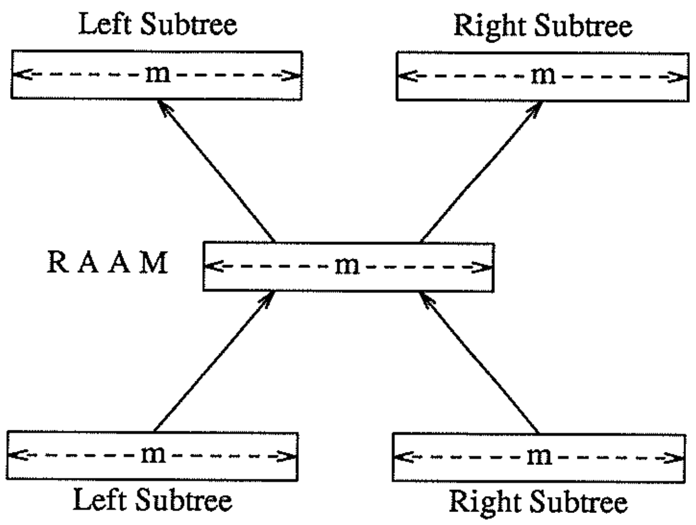

Figure 10a: Tree RAAM Training

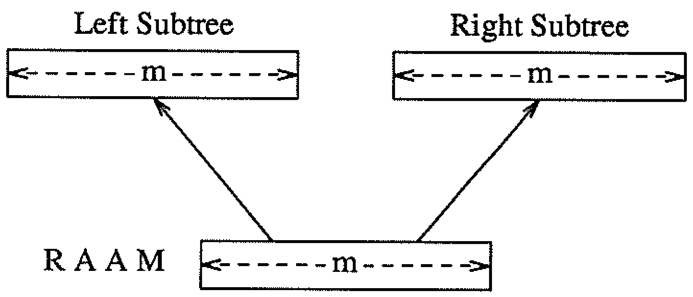

Figure 10b: Tree RAAM Decomposition

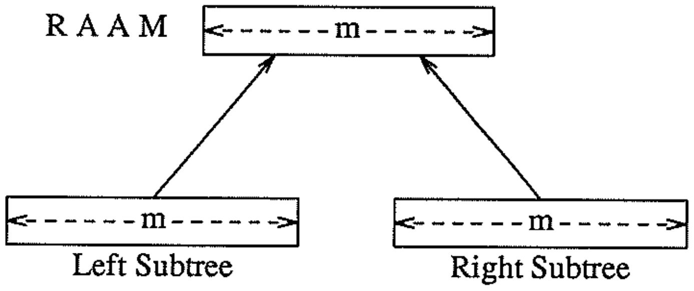

Figure 10c: Tree RAAM Construction

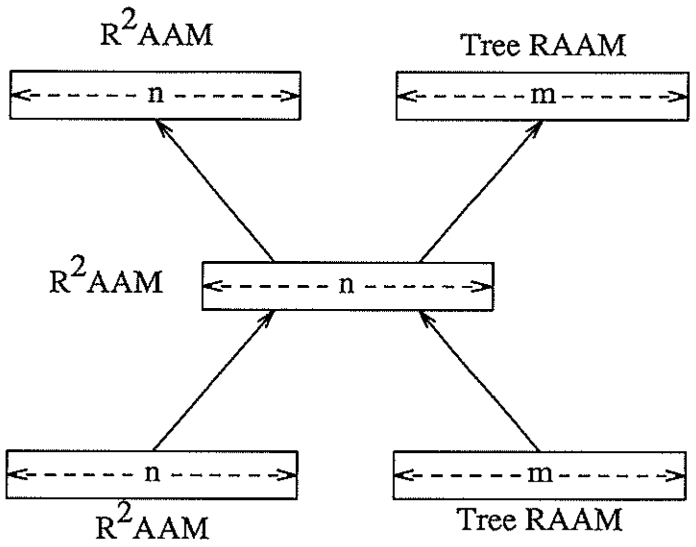

Figure 11a: Training of R^2AAM

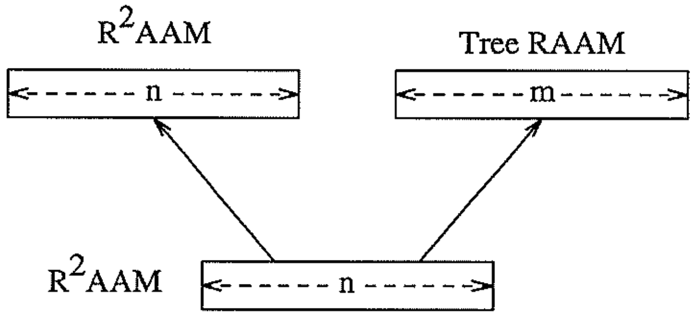

Figure 11b: R^2AAM POP

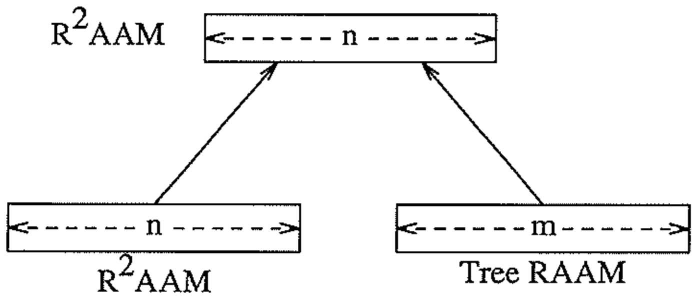

Figure 11c: R^2AAM PUSH

5.1. Recursive Recursive Auto-Associative Memory

For parsing, at least for deterministic parsing, a stack whose elements are trees is required. This leads us to a further generalization of the RAAM idea. We propose a Recursive Recursive Auto-Associative Memory (R^2AAM) which permits the composition of elements which are themselves tree RAAMs into a stack. Figure 11a shows how, as with stacks, the network could be trained to construct circuitry for both push and pop. Figures 11b and 11c show how both pop and push would evolve out of training the memory. Such a memory is recursive in its elements as well as in how the elements combine.

At this point in our research, no experiments have been performed to confirm how well these memory structures will perform. Our speculation is that they will work well and lead to a viable way of thinking about such processing. Possibly, the training of such structures will pose a problem since the complexity of the structure could overwhelm. Also, the resolution of the decomposed structure could prove to be unacceptable. However, the extension of Pollack's ideas to what we require seems very direct.

Remaining is the question of how to know if proper structures are being built in the R^2AAM. For this, we use an idea from St. John and McClelland (1988). A collection of probe units are included such that when a given probe is activated, an encoding of the corresponding piece of structure should appear in the output. It can then be observed if proper encodements are being constructed.

5.2. Design for Fully Connectionistic Parsing

Putting these pieces together, a design emerges for deterministic, fully connectionistic parsing (FCP). Figure 12 shows that design. It is based on the simple idea from deterministic parsing that processing is to occur left-to-right with no backtracking. The input stream fills the input buffer as constituents are processed. The primary task of the network is to shift the input buffer left by one element while internalizing the first element. The element shifted off the end must be represented in the R^2AAM in an appropriate and final way. Note that if processing follows the steps of PARSIFAL, this could require some extremely complicated processing, including the dropping and attaching of several structures. If this proves to be problematic, several small variations are possible to permit ''worst case'' scenarios.

Tests can be conducted with the probes to verify that the element has indeed been properly represented. In fact, the probes must be part of the training to work correctly. The shifted buffer is copied back at the input buffer, with the input stream filling in the missing element. In fact, due to the timing constraints of real-time language processing, the input stream might not have the next token ready for processing requiring a wait for the next token. The R^2AAM from the previous iteration becomes part of the new input. The dashed boxes represent possible additional layers of network. When an empty buffer occurs, parsing is finished and the final structure is accessible through the probes.

This design is an evolution which still subscribes to the determinism hypothesis, but does not demand a complete set of grammar rules for training. Instead, training is based

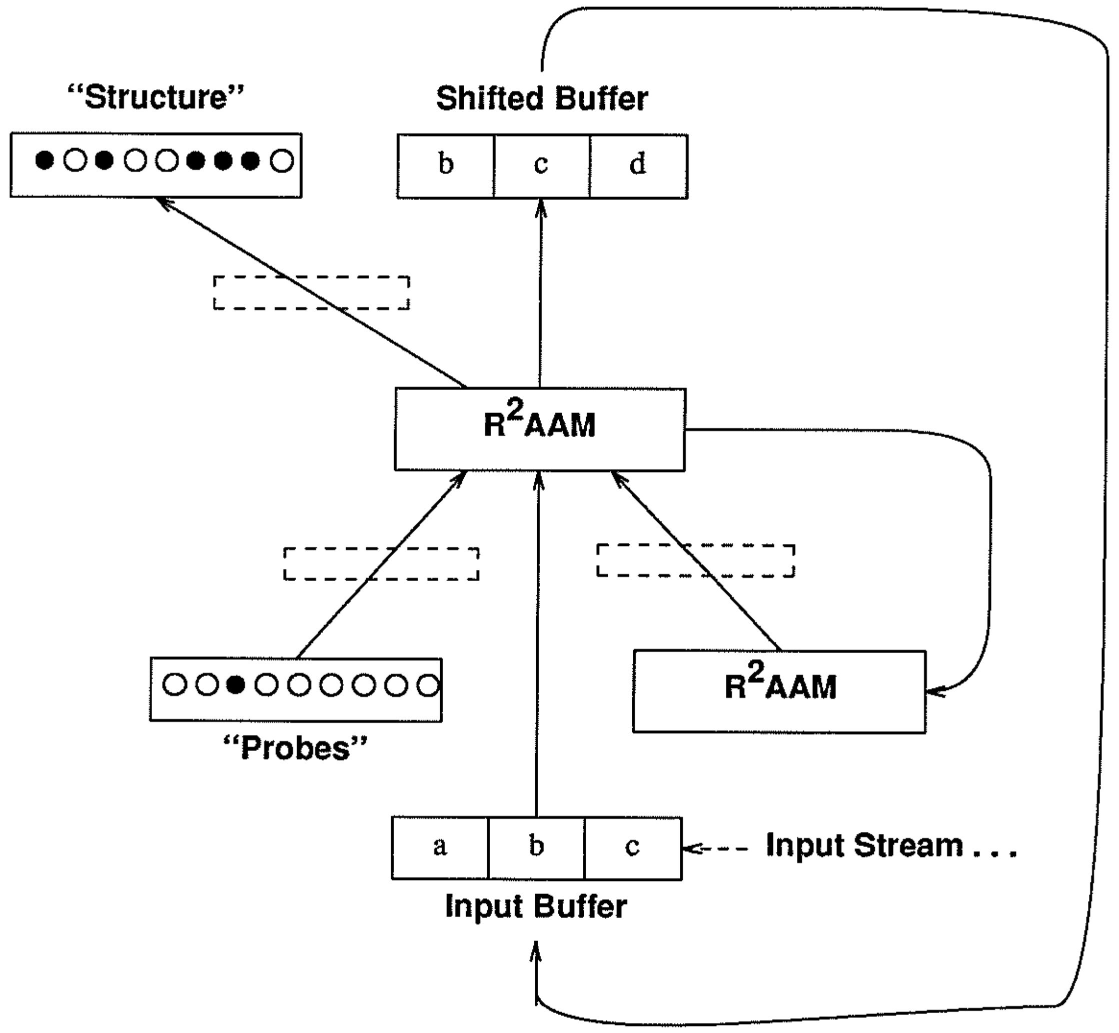

Figure 12:
Towards Fully Connectionistic Parsing

on the resultant structures we require and the step-by-step process for getting there. It is loosely based on deterministic parsing in that we know from those experiences that each input element at the left end of the buffer must be processed into at least a partial structure. Of course, the choice of three buffer positions is also based on assumptions made in deterministic parsing.

Experimentation is just underway to substantiate these claims. Our hope is to move a few steps closer to a fully connectionistic parser, but without sacrificing any desirable properties from conventional parsers. The design in Figure 12 is being evaluated for that purpose.

6. Acknowledgments

We are grateful to many people for helping in this effort. The authors express gratitude to William Ball, Georg Dorffner, Marios Fourakis, David Harker, Dan Kimura, Ron Loui, and John Merrill for thoughtful discussions and comments concerning this work.

References

Barker, V.E. and D.E. O'Connor. 1989. "Expert Systems for Configuration at Digital: XCON and Beyond." *Communications of the ACM 32*, 3, 298-318.

Berwick, R.C. 1985. *The Acquisition of Syntactic Knowledge.* MIT Press, Cambridge, MA.

Blank, G.D. 1989. "A Finite and Real-Time Processor for Natural Language." *Communications of the ACM 32*, 10, 1174-1189.

Charniak, E. 1983. "A Parser with Something for Everyone." In *Parsing Natural Language*, M. King, ed. Academic Press, New York, NY, 117-150.

Chomsky, N. 1965. *Aspects of the Theory of Syntax.* Cambridge: MIT Press.

Cottrell, G.W. 1985. "Connectionist Parsing." *Proceedings of the 7th Annual Conference of the Cognitive Science Society*, Irvine, CA, 201-211.

Davis, R. 1982. "Teiresias: Applications of Meta-Level Knowledge." In Davis, R. and Lenat, D.B., *Knowledge-Based Systems in Artificial Intelligence*, New York: McGraw-Hill.

Faisal, K.A. 1989. "CDP: Connectionist Deterministic Parser, A Dissertation Proposal," Technical Report WUCS-89-33. Department of Computer Science, Washington University, St. Louis, Mo. (Aug.).

Faisal, K.A. and S.C. Kwasny. 1990. "Deductive and Inductive Learning in a Connectionist Deterministic Parser." *Proceedings of the International Joint Conference on Neural Networks, forthcoming*, Washington, DC, (Jan 15-19).

Fanty, M. 1985. "Context-Free Parsing in Connectionist Networks." Technical Report 174, Computer Science Department, University of Rochester, Rochester, NY.

Gallant, Stephen I. 1988. "Connectionist Expert Systems," *Communications of the ACM 31*, 2, 152-169.

Kitzmiller, C.T., and J.S. Kowalik. 1987. "Coupling Symbolic and Numeric Computing in Knowledge-Based Systems." *AI Magazine 8*, no. 2, 85-90.

Kwasny, S.C., K.A. Faisal, and W.E. Ball. 1990. "Unifying Several Natural Language Systems in a Connectionist Deterministic Parser." *Proceedings of the Western Multi Conference, forthcoming*, San Diego, CA, (Jan 17-19).

Kwasny, S.C. and K.A. Faisal. 1989. "Competition and Learning in a Connectionist Deterministic Parser." In *Proceedings of the 11th Annual Conference of The Cognitive Science Society*, (Ann Arbor, MI, Aug. 16-19). Lawrence Erlbaum Associates, Hillsdale, NJ, 690-697.

Kwasny, S.C. 1988. "A Parallel Distributed Approach to Parsing Natural Language Deterministically." Technical Report WUCS-88-21. Department of Computer Science, Washington University, St. Louis, Mo. (Aug.).

Kwasny, S.C. and N.K. Sondheimer. 1981. "Relaxation Techniques for Parsing Ill-Formed Input." *American Journal of Computational Linguistics 7*, no. 2, 99-108.

Kwasny, S.C. 1980. "Treatment of Ungrammatical and Extra-Grammatical Phenomena in Natural Language Understanding Systems." Indiana University Linguistics Club, Bloomington, Indiana. (Nov.).

Marcus, M. P. 1980. *A Theory of Syntactic Recognition for Natural Language.* MIT Press, Cambridge, MA.

McClelland, J. L., & D.E. Rumelhart. 1988. *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*, Cambridge: MIT Press.

McClelland, J. L., & A. H. Kawamoto. 1986. "Mechanisms of Sentence Processing: Assigning Roles to Constituents of Sentences." In *Parallel Distributed Processing*, D.E. Rumelhart and J.L. McClelland, MIT Press, Cambridge, MA, 272-325.

McClelland, J.L., & J.L. Elman. (1986). "The TRACE Model of Speech Perception." *Cognitive Psychology 18*, 1-86.

Milne, R. 1986. "Resolving Lexical Ambiguity in a Deterministic Parser." *Computational Linguistics 12*, no. 1 (Jan-Mar): 1-12.

Pollack, Jordan B. 1989. "Recursive Distributed Representations," Report 89-JP-RECURSIVE, Laboratory for Artificial Intelligence Research, The Ohio State University, Columbus, Oh. (Aug.)

Rumelhart, D. E., G. Hinton, and R.J. Williams. 1986. "Learning Internal Representations by Error Propagation." In *Parallel Distributed Processing*, D.E. Rumelhart and J.L. McClelland, MIT Press, Cambridge, MA, 318-364.

St. John, M.F., and J.L. McClelland. 1988. "Learning and Applying Contextual Constraints in Sentence Comprehension." Technical Report AIP-39, Carnegie-Mellon University, Department of Psychology, Pittsburgh, Pa. (June 8).

Saussure, F. de. 1955. *Cours de Linguistique Générale.* 5th edition. Payot, Paris.

Selman, B. & G. Hirst. 1985. "A Rule-Based Connectionist Parsing System." *Proceedings of the 7th Annual Conference of the Cognitive Science Society*, Irvine, CA, 212-221.

Servan-Schreiber, D., A. Cleeremans, & J.L. McClelland. 1988. "Encoding Sequential Structure in Simple Recurrent Networks." Report CMU-CS-88-183, Carnegie Mellon University, Pittsburgh, PA.

Waltz, D.L. & J.B. Pollack. 1985. "Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation." *Cognitive Science 9*, 51-74.

Weischedel, R.M. and N.K. Sondheimer. 1983. "Meta-Rules as a Basis for Processing Ill-Formed Input." *American Journal of Computational Linguistics 9*, no. 3-4, 161-177.

Werbos, P. 1974. "Beyond Regression: New Tools for Prediction and Analysis in Behavioral Science." PhD Thesis. Harvard University, Cambridge, Ma.

Westphal, C.R., & K.L. McGraw. 1989. SIGART — Special Issue on Knowledge Acquisition. Number 108, April. New York: ACM Press.

Williams, R.J., & D. Zipser. 1988. ''A Learning Algorithm for Continually Running Fully Recurrent Neural Networks.'' ICS Report 8805, University of California, San Diego, CA.

Winograd, T. and F. Flores. 1987. *Understanding Computers and Cognition*. Addison-Wesley, Reading, Ma.

Appendix A: Grammar Rules

1.	Rule Initial	If	stack is empty
		then	Create a new S node
2.	Rule Yes-No-Q	If	current node is S
			no attachments
			first item is an auxiliary verb
			second item is NP node
		then	Switch
3.	Rule Imperative	If	current node is S
			no attachments
			first item is a infinitive verb
		then	Insert YOU
4.	Rule Parse-Subj	If	current node is S
			no attachments
			first item is NP node
			second item is a verb
		then	Attach the first item as NP
5.	Rule Start-Aux	If	current node is S
			first item is a verb
			AUX node is not attached
		then	Create AUX node
6.	Rule Aux-Attach	If	current node is S
			first item is AUX node
		then	Attach the first item as AUX
7.	Rule Aux-Perfective	If	current node is AUX
			first item is auxiliary of root HAVE
			second is past participle verb
		then	Attach the first item as PERF.
8.	Rule Aux-Progressive	If	current node is AUX
			first item is auxiliary of root BE
			second is present participle verb
		then	Attach the first item as PROG.
9.	Rule Aux-Passive	If	current node is AUX
			first item is auxiliary of root BE
			second is past participle verb
		then	Attach the first item PASSIVE
10.	Rule Aux-Modal	If	current node is AUX
			first item is auxiliary of type Modal
			second is infinitive verb
		then	Attach the first item as MODAL

11.	Rule Aux-Do	If	current node is AUX
			first item is auxiliary of root DO
			second is infinitive verb
		then	Attach the first item as DO
12.	Rule Aux-Complete	If	current node is AUX
			first item is not auxiliary
		then	Drop the current node into the buffer
13.	Rule MVB 1	If	current node is S
			first item is a verb
			AUX node is attached
		then	Create VP node
14.	Rule MVB 2	If	current node is VP
			first item is a verb
		then	Attach the first item as MVB
15.	Rule Passive 1	If	current node is VP
			auxiliary of the S node is passive
		then	Create NP
16.	Rule Passive 2	If	the current node is NP
		then	Drop the current node into the buffer
17.	Rule Objects	If	current node is VP
			first item is NP node
		then	Attach the first item as NP
18.	Rule PP-Under-VP	If	current node is VP
			first item is PP node
		then	Attach the first item as PP
19.	Rule VP-Done 1	If	current node is VP
			first item is FinalPunct.
		then	Drop the current node into the buffer
20.	Rule VP-Done 2	If	current node is S
			first item is VP node
		then	Attach the first item VP
21.	Rule S-Done 1	If	current node is S
			first item is FinalPunct.
		then	Attach the first item as FIN.
22.	Rule S-Done 2	If	current node is S
			buffer is empty
		then	Stop, reporting a successful parse

Appendix B: Grammar Rule Encoding

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Stack Nodes																						
S	–	+	+	+	+	+	–	–	–	–	–	–	+	–	–	–	–	–	–	+	+	+
NP	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–	–
VP	–	–	–	–	–	–	–	–	–	–	–	–	–	+	+	–	+	+	+	–	–	–
AUX	–	–	–	–	–	–	+	+	+	+	+	+	–	–	–	–	–	–	–	–	–	–
No-attach	+	+	+	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
AUX-PASSIVE	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	+	?	?	?	?	?	?
AUX-ATTACHED	–	–	–	–	–	–	–	–	–	–	–	–	+	+	+	+	+	+	+	+	+	+
1st BUFFER																						
NP	?	–	–	+	–	–	–	–	–	–	–	?	–	–	?	?	+	–	–	–	–	–
VP	?	–	–	–	–	–	–	–	–	–	–	?	–	–	?	?	–	–	–	+	–	–
PP	?	–	–	–	–	–	–	–	–	–	–	?	–	–	?	?	–	+	–	–	–	–
AUX	?	–	–	–	–	+	–	–	–	–	–	?	–	–	?	?	–	–	–	–	–	–
VERB	?	+	+	–	+	–	+	+	+	+	+	?	+	+	?	?	–	–	–	–	–	–
Inf	?	–	+	–	?	–	–	–	–	–	–	?	?	?	?	?	–	–	–	–	–	–
Ing	?	–	–	–	?	–	–	–	–	–	–	?	?	?	?	?	–	–	–	–	–	–
En	?	–	–	–	?	–	–	–	–	–	–	?	?	?	?	?	–	–	–	–	–	–
auxVerb	?	+	?	–	?	–	+	+	+	+	+	–	?	?	?	?	–	–	–	–	–	–
Be	?	?	?	–	?	–	–	+	+	–	–	?	?	?	?	?	–	–	–	–	–	–
Have	?	?	?	–	?	–	+	–	–	–	–	?	?	?	?	?	–	–	–	–	–	–
Do	?	?	?	–	?	–	–	–	–	–	+	?	?	?	?	?	–	–	–	–	–	–
Modal	?	?	–	–	?	–	–	–	–	+	–	?	–	–	?	?	–	–	–	–	–	–
FinalPunct	?	–	–	–	–	–	–	–	–	–	–	?	–	–	?	?	–	–	+	–	+	–
2nd BUFFER																						
NP	?	+	?	–	?	?	–	–	–	–	–	?	?	?	?	?	?	?	–	–	–	–
VP	?	–	?	–	?	?	–	–	–	–	–	?	?	?	?	?	?	?	–	–	–	–
PP	?	–	?	–	?	?	–	–	–	–	–	?	?	?	?	?	?	?	–	–	–	–
AUX	?	–	?	–	?	?	–	–	–	–	–	?	?	?	?	?	?	?	–	–	–	–
VERB	?	–	?	+	?	?	+	+	+	+	+	?	?	?	?	?	?	?	–	–	–	–
Inf	?	–	?	?	?	?	–	–	–	+	+	?	?	?	?	?	?	?	–	–	–	–
Ing	?	–	?	?	?	?	–	+	–	–	–	?	?	?	?	?	?	?	–	–	–	–
En	?	–	?	?	?	?	+	–	+	–	–	?	?	?	?	?	?	?	–	–	–	–
auxVerb	?	–	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	–	–	–	–
Be	?	–	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	–	–	–	–
Have	?	–	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	–	–	–	–
Do	?	–	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	–	–	–	–
Modal	?	–	?	?	?	?	–	–	–	–	–	?	?	?	?	?	?	?	–	–	–	–
FinalPunct	?	–	?	–	?	?	–	–	–	–	–	?	?	?	?	?	?	?	–	+	–	–
Actions																						
Attach																						
Subj-NP	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
VP	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–
AUX	–	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
FinalPunct	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–
MVB	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–	–
Obj-NP	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–
PP	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–
AUX-Perf	–	–	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
AUX-Prog	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–
AUX-Pass	–	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–
AUX-Modal	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–
AUX-DO	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–
Create																						
S	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
NP	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–
VP	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–	–	–
AUX	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Insert-You	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Switch	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Drop	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	+	–	–	+	–	–	–
Stop	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+