Expert Systems With Applications, Vol. 2, pp. 47–58, 1991
Printed in the USA.

0957-4174/91 $3.00 + .00

Rule-Based Training of Neural Networks

STAN C. KWASNY

Center for Intelligent Computer Systems,* Department of Computer Science, Washington University, St. Louis, MO

KANAAN A. FAISAL

King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

Abstract—*Rule-based expert systems either develop out of the direct involvement of a concerned expert or through the enormous efforts of intermediaries called knowledge engineers. In either case, knowledge engineering tools are inadequate in many ways to support the complex problem of expert system building. This article describes a set of experiments with adaptive neural networks which explore two types of learning, deductive and inductive, in the context of a rule-based, deterministic parser of Natural Language. Rule-based processing of Language is an important and complex domain. Experiences gained in this domain generalize to other rule-based domains. We report on those experiences and draw some general conclusions that are relevant to knowledge engineering activities and maintenance of rule-based systems.*

1. INTRODUCTION

IN THE FIELD of Artificial Intelligence (AI), 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 some knowledge is acquired, it becomes the basis for system organization and rules that mimic the expertise. However, acquisition of such knowledge is an ongoing concern even as refinements are made during maintenance.

The task is not a simple one. 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. Certainty factors, for example, 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). The task of Natural Language Understanding, in fact, has been perceived as a task of knowledge engineering (Shapiro & Neal, 1982).

Once formulated, rules are difficult to debug and maintain. In XSEL/XCON, for example, it is reported (Barker & O'Connor, 1989) that 40% of the rules require changes each year to adapt to new products and for other reasons. Systems like TEIRESIAS (Davis, 1982) have been specifically designed to aid in the cri-

Requests for reprints should be sent to Stan C. Kwasny, Center for Intelligent Computer Systems, Department of Computer Science, Washington University, Campus Box 1045, 1 Brookings Drive, St. Louis, MO 63130.

S.C.W. gratefully acknowledges the support of King Fahd University of Petroleum and Minerals. We thank the reviewers for their comments which have improved aspects of our presentation of this work. The authors also express gratitude to William E. Ball, Georg Dorffner, Marios Fourakis, David Harker, Dan Kimura, Barry Kalman, Ron Loui, John Merrill, Gadi Pinkas, and Robert Port for thoughtful discussions and comments concerning this work.

* The sponsors of the Center are McDonnell Douglas Corporation, Southwestern Bell Telephone Company, and Mitsubishi Electronics America, Inc.

tiquing and reformulation of rule-based systems by the experts themselves. However, the expert is still required to operate at the level of rules, which may be very unnatural in many domains. Rules, no matter how sophisticated, do not always make the best programming language.

Even with better tools, once performance reaches a certain point it may prove difficult to achieve an improved level of performance. Upon analysis, some of the difficulty may be attributable to design decisions made early in the construction of the system which lose their validity as the system evolves. Such problems often require changes to the basic design and complete retooling of the system.

Traditionally, rules also play an important role in the processing of Natural Language. In fact, the comparison between rule-based expert systems and rule-based Natural Language systems is close indeed, especially in examining some of the difficulties 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 both types of systems.

The goal for expert systems is to symbolically describe a domain in such a manner that relevant decisionmaking can occur and that a computer system can perform at the level of expert in this regard. But, is this always a realistic goal? While progress in Natural Language Processing (NLP), for example, continues to be made, why does this continue to be only a partially solved problem? Or, more specifically, why is there no operationally complete set of symbolic rules for English (i.e., no set of rules that perfectly describes all understandable English utterances)? Certainly, it is neither for lack of experts nor lack of effort. Perhaps, if we agree with Winograd and Flores (1987, p. 107), the answer is that ". . . computers cannot understand language." We do not believe computer systems are so impaired.

The problems raised for rule-based expert systems as well as those mentioned for rule-based NLP could be attacked through learning. Learning would seem to be a solution, but traditional learning approaches too often require even deeper insights and understanding of the problem domain than the expert himself may possess. While strictly symbolic learning may be possible, such systems typically need to invent new symbols to extend their capabilities symbolically and this imposes limitations on the approach.

If learning could be conducted in a sufficiently flexible manner, a trainable system could 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. This distinction between easily articulated rules and the more difficult ones is analogous to the distinction between textbook and experiential learning. The new intern, for example, is filled with book learning, but lacks the wisdom and knowledge of the domain which only comes through experience. Connectionist (neural) networks hold promise for providing such a flexible learning scheme.

This article presents results from experimentation with a connectionist parsing system. The system is simply viewed as an example of a complex rule-based system. Conclusions reached in these experiments, therefore, have consequences for many other types of rule-based systems.

2. RULE-BASED CONNECTIONIST PARSING

Natural Language processing is both symbolic and subsymbolic. It is symbolic in the role symbols play in writing systems and in the chunking of concepts, while it is subsymbolic because of the fuzziness of concepts, and the apparent high degree of parallelism in the activity. In building parsers, the subsymbolic aspect of language is usually lost, although there have been attempts to construct parsers that are totally subsymbolic (see Fanty, 1985; Selman & Hirst, 1985; Waltz & Pollack, 1985).

Rules usually play a sacred role in parsing systems and, as mentioned earlier, are often executed as if following instructions in a program. Whether we are only interested in building robust expert systems, or are interested in modeling human expertise, this method is incorrect. Rules should be permitted to play an advisory role only—that is, for guidance in typical situations and not as prescriptions for precise processing.

In the case of English, if a complete set of rules for all meaningful English forms existed, then it might be satisfactory to rely on a rule-based approach. But no such set of rules exists, nor does it seem desirable or even possible to construct such a set. Any rule-based system that is based literally on rules tends to be brittle since there is no direct way to process inputs that are not anticipated to some extent. Furthermore, the acquisition of new rules often requires tedious retuning of existing rules. The only solution to these problems in a practical and realistic manner is through learning.

We have developed a connectionist deterministic parsing system called CDP which offers solutions to these problems (Kwasny & Faisal, 1989). Training can be conducted either from rules or from examples of processing. The resultant network is then tested on a variety of novel sentence patterns and its generalization capabilities studied.

A set of experiments are presented 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 set of deterministic grammar rules and tested with sentences which are grammatical and ones that are not. Tests are also conducted with sentences containing lexically ambiguous items.

2.1. Deterministic Parsing

For a complete understanding of the motivation and architecture of CDP, it is necessary to describe the work on which it is based. For a readable description of deterministic, "wait-and-see" parsing, see Winston (1984), or for a more thorough discussion, see Marcus (1980).

The determinism hypothesis restricts Natural Language Processing to a deterministic mechanism. It states that

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

It follows from this hypothesis that NLP need not depend on backtracking, nor are any partial structures produced during parsing which fail to become part of the final structure. This is equivalent to prohibiting chains of reasoning in a rule-based expert system which ultimately do not contribute to the final answer. Obviously, processing is restricted in a major way under this assumption.

PARSIFAL (Marcus, 1980) is a demonstration of a a deterministic, rule-based parser of Natural Language. Extensions to this system have been proposed for processing ungrammatical sentences [PARAGRAM (Charniak, 1983)], for resolving lexical ambiguities [ROBIE (Milne, 1986)], and for acquiring syntactic rules from examples [LPARSIFAL (Berwick, 1985)].

We have found it beneficial to combine these four tasks into one implementation which is partly symbolic and partly connectionist. The connectionist approach has particular advantageous in unifying these four systems (Kwasny, Faisal, & Ball, in press). The system architecture of PARSIFAL has been reconfigured 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 discovered independently by Werbos (1974) and Rumelhart, Hinton, and Williams (1986).

As illustrated in Figure 1, the primary components of a deterministic parser are a buffer for lookahead in the sentence, a stack for processing embedded structures, and a collection of rules for controlling the building and movement of constituents of the sentence being processed. Processing occurs essentially left-to-right. The absence of backtracking is an important advantage in developing a connectionist-based parser since structures, once built, are never discarded.

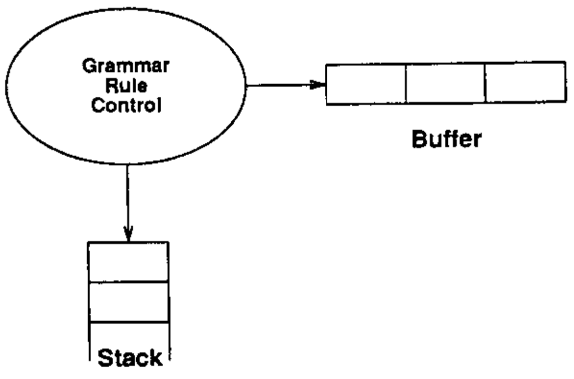

FIGURE 1. Deterministic Parsing in Schematic Form.

Rules are partitioned into rule sets. A standard recognize-act cycle is used to achieve a parse. A consequent of a rule may be the activation or deactivation of a rule set thereby providing a simple conflict resolution strategy. Conflicts within rule sets are resolved from the static ordering (i.e., numeric priority) of the rules. Actions can effect changes to both the stack and the buffer. As buffer positions are vacated at the far end, new sentence components flow in sequentially. If a successful parse is found, a termination rule will fire leaving the final structure on top of the stack.

2.2. Connectionist Deterministic Parsing

CDP provides a setting for experimentation with a variety of grammars and network designs. It combines the basic mechanism of deterministic parsing with an adaptive, feed-forward neural network to enable the generalization and robustness of connectionism to be evaluated in this domain. A backpropagation 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 construct a mechanism capable of learning to deal syntactically with language in a robust manner.

As Gallant (1988) points out, there are important advantages to constructing rule-based systems using neural networks. 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 & Kwasny, in press).

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 rules that are general (in the sense that they must be applicable in a wide variety of processing situations), but specific sentence forms must be processed. The "inductive" strategy derives its training sequence from coded examples of sentence processing. It is inductive in the sense that it is based on specific sentence examples, although a potentially wider variety of sentence forms must be processed. 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 entire set of grammar rules is contained in Appendix A. 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 representation 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 comparison to the other deterministic parsing systems, CDP performs favorably. Virtually all of our examples have been drawn from previous work. 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.

3. 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 saved 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 average strength of responses during processing. The closer the sentence matches the training patterns, the lower the error and the greater the strength. Strengths are used for comparison purposes only.

Strengths are computed as the reciprocal of the error. The strength for each individual step is simply the reciprocal of the error for that step. The step 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.

As mentioned earlier, there are two distinct approaches to training a network to parse sentences. Each of these training strategies results in a slightly different version of CDP. The differences in the derivation of the two types of training patterns are illustrated in Figure 2. Deductive training begins with deterministic grammar rules which are coded into rule templates, one rule template representing one grammar rule. Instantiation of a rule template leads to a training pattern which is presented during learning. Coding and instantiation are discussed below. Inductive training is based on traces of sentence processing itself. The coded training patterns derived in this way have in some sense

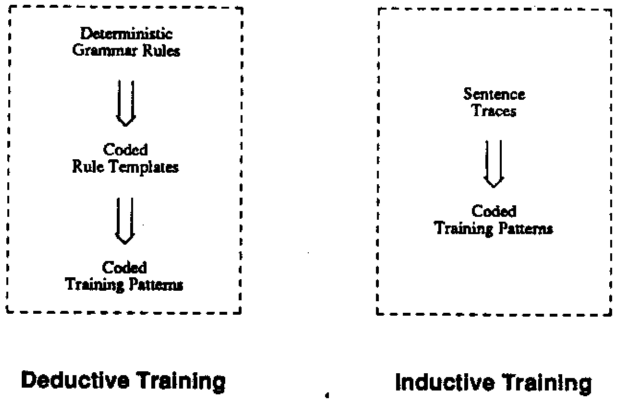

FIGURE 2. Extraction of Deductive and Inductive Training Patterns.

already been instantiated and, therefore, are suitable for learning with no further translation.

3.1. 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 as in PARSIFAL. Each constituent in the rule is represented by an ordered feature vector of +1 (on), −1 (off), or ? (do not care). Instantiation of the vector occurs by randomly changing ? to +1 or −1. Each template, therefore, can be instantiated into many patterns making each epoch of training slightly different. During training, the network learns the inputs which are highly correlated with expected outputs and those that are not. Training is arranged so that ? values are uncorrelated with the outputs from those training patterns.

Each rule template containing *n* ?'s can generate up to 2^n unique training cases. Some rule templates have up to 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:

$$\{\langle\text{Stack}\rangle\langle\text{1st Item}\rangle\langle\text{2nd Item}\rangle\langle\text{3rd Item}\rangle \rightarrow \text{Action}\}$$

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

$$\{\langle S\ node\rangle\langle\text{“}have\text{”}\rangle\langle NP\rangle\langle VERB, -en\rangle \rightarrow \textit{Switch 1st and 2nd items}\}$$

while a rule for imperative sentences would be written:

$$\{\langle S\ node\rangle\langle\text{“}have\text{”}\rangle\langle NP\rangle\langle VERB, inf\rangle \rightarrow \textit{Insert YOU}\}$$

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.

Appendix A contains the grammar rules used as a basis for all deductive training experiments in this study. Our rules were derived from the grammar contained in Appendix C of Marcus (1980) which includes those rules specifically discussed in building a case for deterministic parsing. They 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 correctly parses those sentences that the original rules could parse.

3.2. Inductive Learning

Inductive training depends on training patterns derived from traces of processing of actual sentences. This processing is guided by application of the rules of a deterministic grammar as before. This form of training requires that the network demonstrate the correct rule-following behavior after training runs with a comparatively small sample of sentence traces. Although no symbolic rules are learned, the behavior of the rules is captured within the weights 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.

The primary difference between the two forms of learning is seen if we consider the space of possible patterns. With deductive training, that space is systematically presented during learning in such a way that each major distinction to be made during processing is represented in each epoch. Inductive training happens in a less systematic way with no guarantee of appropriate representation of cases. Thus, 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 processing the actual samples of sentences.

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 study.

3.3. Architecture of CDP

As Figure 3 illustrates, CDP is organized into a symbolic component and a subsymbolic component. Actions in CDP are performed symbolically on traditional data structures which are also maintained symbolically. The subsymbolic 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 & Ko-

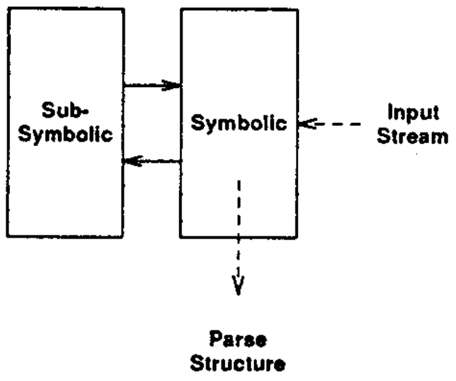

FIGURE 3. CDP System Organization.

walik, 1987). For CDP, the advantages are performance and robustness.

It is the responsibility of the symbolic component to handle the input sentence coding it for presentation to the network. In the subsymbolic component, the network produces a designation of an action to be performed by producing an activation pattern across output units. Activation of output units is interpreted in a winner-take-all manner, with the highest activated unit determining the action to be taken. Actions themselves are performed symbolically on conventional data structures. The whole process is very efficient in time and space, although learning itself occurs off-line and is a time-consuming process.

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.

What the model actually sees as input during sentence processing 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. Iteration over an input stream is performed by moving unprocessed sentence forms into the buffer as vacancies are created. Iteration ends when the buffer becomes empty and a stop action is requested by the network.

At this point, it is instructive to follow an example with more details revealed as shown in Figure 5. When a sentence form like

**John have should scheduled the meeting.*

appears in the input stream, the first three constituents fill the buffer. Note that in reality this is an ungrammatical sentence form. Later, it will be shown that CDP can correctly produce the structure shown. The contents of the three elements along with the contents of the top of the stack are coded into a feature vector and presented to the network producing a single action. The action is executed potentially producing changes to the buffer and stack. When processing stops, the final structure can be seen on the top of the stack, just as for PARSIFAL.

4. PERFORMANCE

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

4.1. Parsing Grammatical Sentences

Experimentation with grammatical sentences demonstrates the ability of CDP to perform as PARSIFAL. Earlier we mentioned that convergence testing from the rule templates is possible by changing each *?* to a zero value. Here we examine the performance of CDP with actual sentences.

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.

Strengths are computed as the reciprocal of the average error per processing step for each sentence. The error on each step is determined by taking the Euclidean distance between the actual vector of output unit activation values and an idealized vector with only the

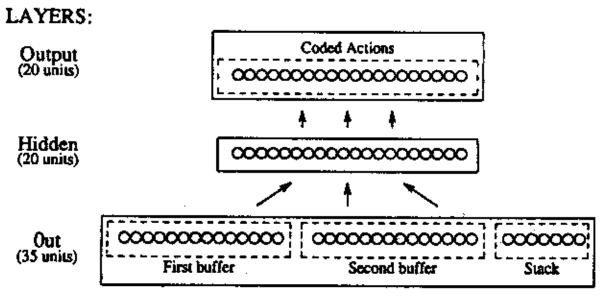

FIGURE 4. Subsymbolic Component.

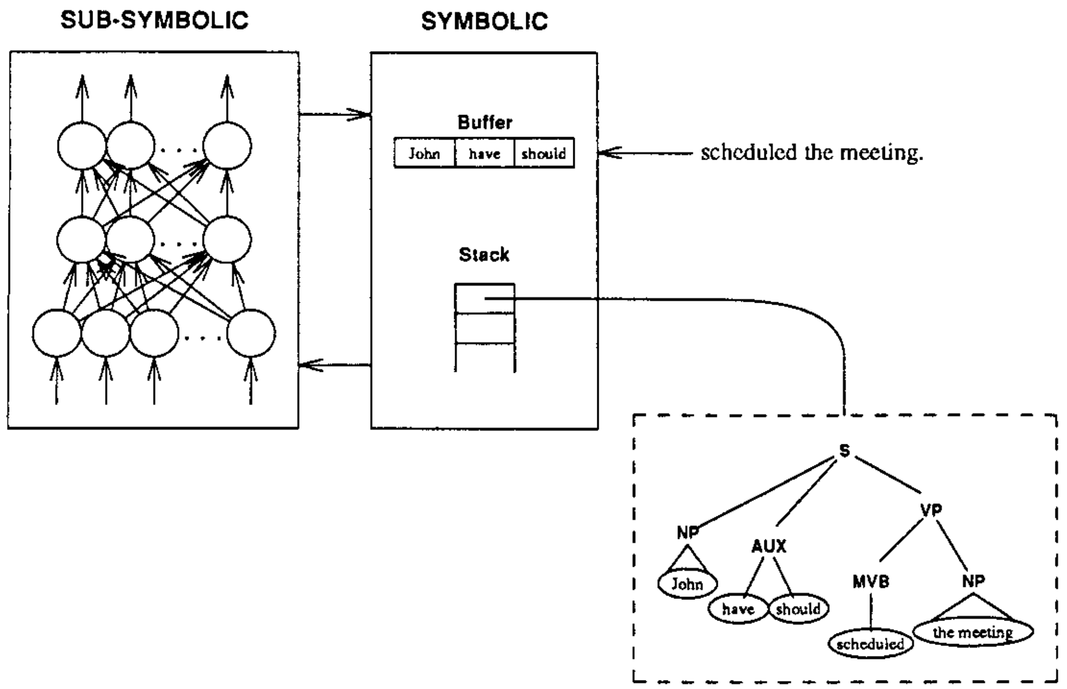

FIGURE 5. CDP System Overview.

winning unit turned on (one corner of the hypercube in error space). Strengths reflect the certainty with which actions for building structures are being selected, but should be used for relative comparisons only.

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.

Most of these sentences come from examples used in 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 presented to CDP unambiguously, but the lexical choices are provided in parentheses.

Capabilities described thus far have only duplicated what is easily done symbolically. Of course, the feed-forward network does support very fast decisionmaking due to the feed-forward nature of the model. But what other features does the model possess? Importantly, how robust is the processing?

4.2. Lexical Ambiguity

ROBIE extends PARSIFAL to address issues of lexical ambiguity. It requires additional rules and lexical fea-

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 sceduling 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

tures to handle these properly. In the deterministic approach, it is essential that lexical items be properly disambiguated to permit processing to proceed without backtracking.

In a set of experiments with CDP, the parser is tested for this. Normal sentences are 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. Selected sentences are shown in Table 2. The numbers again indicate the average strength for each sentence. In the cases shown, the lexically ambiguous words are correctly interpreted and reasonable structures result.

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.

For example, sentence (12) 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 (13), 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 (14) contains the word *will* coded ambiguously as an NP and an auxiliary modal verb. In the context of the sentence, it is clearly used as a modal auxiliary and the parser treats it that way. A similar result is obtained for sentence (15). In sentence (16), *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.

TABLE 2
Lexically Ambiguous Sentences Used in Testing

Sentence Form	Deductive Avg. Strength	Inductive Avg. Strength
(12) They ⟨can⟩ fish.	4.5	2.6
(13) They can ⟨fish⟩.	172.2	4.9
(14) ⟨Will⟩ he go?	83.6	14.3
(15) Tom ⟨will⟩ hit Mary.	118.7	19.9
(16) Tom ⟨hit⟩ Mary.	39.0	2.5

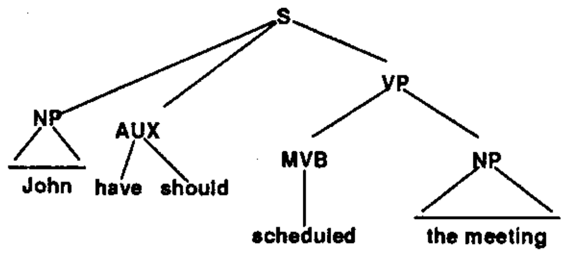

FIGURE 6. Parse of "*John have should scheduled the meeting."

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

4.3. 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 always has some rule to fire. Reported experimentation with PARAGRAM shows this to be an effective method for stretching the inherent capabilities of the grammar.

CDP attempts to accomplish this through generalization. An important test of these capabilities is its responses to novel sentences. These are strictly dependent upon its training experiences since in deductive training no relaxation rules (Kwasny & Sondheimer, 1983), meta-rules (Weischedel & Sondheimer, 1983), or other special mechanisms are added to the original grammar rules to handle ungrammatical cases. Likewise, in inductive training no ungrammatical sentences are 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. For our purposes, most of the examples are derived from PARAGRAM and the parsing systems mentioned earlier.

Selected ungrammatical sentences are properly processed as a direct result of the generalization properties

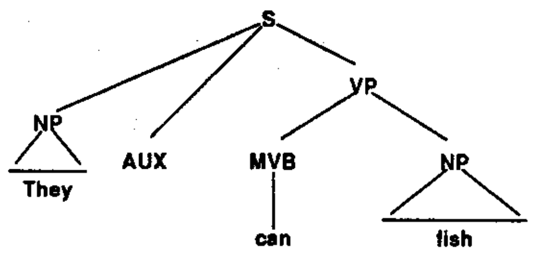

FIGURE 7. Parse of "They can(v) fish(np)."

TABLE 3
Ungrammatical Sentences Used in Testing

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

of learning. When presented an ungrammatical sentence, (i.e., 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.

In Table 3, selected ungrammatical sentences are shown. These examples produce 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 (in Table 3) when compared to similar grammatical ones (in Table 1).

Sentence (17) is similar to sentence (1), except that the word order has been changed in a simple way. Since sequencing of items in the buffer is a seemingly crucial part of the model, it was felt that this would test the model significantly. Surprisingly, the structure produced is identical to that produced while parsing sentence (1), but with lower strengths. The only difference is that the two auxiliary verbs, *have* and *should*, are reversed as shown in Figure 6.

Sentence (18) contains a disagreement between the auxiliary *has* and the main verb *schedule*. This example investigates whether predictions made while processing the auxiliary can be only partially fulfilled and still lead to a successful parse. In this case, the comparable grammatical sentence (3) parses identically. Strengths are again lower in both deductive and inductive cases.

Sentence (19) 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 past tense. Although this is incorrect for this sentence, it is not an unreasonable choice. On closer examination, our inductive training patterns seem to reflect passives more thoroughly than competing alternatives. This is the only case in which inductive exceeds deductive strength. In this case, two dissimilar processing sequences are followed and, therefore, the two results are not comparable.

Sentence (20) 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 (20) 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 (20), as reported by Charniak (1983).

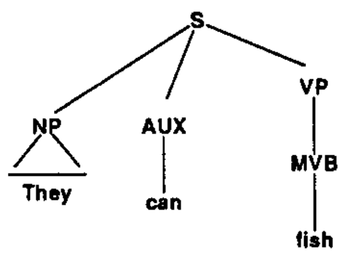

FIGURE 8. Parse of "*They can(aux) fish(v).*"

4.4. Discussion of Results

Using syntax-based approaches to resolving ungrammatical sentences has limitations as seen above due to the ambiguities of such sentences. These problems are well-known as discussed in Kwasny (1980). Importantly, CDP is easily influenced to make one choice or another as a matter of competition. This is a fundamental property of CDP as argued elsewhere (Kwasny & Faisal, 1989).

While deductive training exhibits better performance than inductive training for all tasks, there are trade-offs 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.

5. CONCLUSIONS

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 retrained. Contrast this with the typical rule-based expert system in which each new rule may require rethinking 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.

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).

The potential for exploiting the notion of extensional programming is also very good. The dependency on inductive learning will be greatest in those domains which are difficult to analyze. These are exactly the domains in which standard expert system techniques break down. We have begun to examine an approach to NLP based almost exclusively on inductive training sequences (Kwasny & Faisal, 1989). This looks promising, but is a topic for further research.

REFERENCES

Barker, V.E. & O'Connor, D.E. (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.* Cambridge, MA: MIT Press.

Charniak, E. (1983). A parser with something for everyone. In M. King (ed.), *Parsing natural language* (pp. 117–150). New York: Academic Press.

Davis, R. (1982). Teiresias: Applications of meta-level knowledge. In R. Davis and D.B. Lenat (eds.), *Knowledge-based systems in artificial intelligence.* New York: McGraw-Hill.

Faisal, K.A. (1989, August). CDP: Connectionist deterministic parser, a dissertation proposal. *Technical Report WUCS-89-33,* Department of Computer Science, Washington University, St. Louis, MO.

Faisal, K.A. & Kwasny, S.C. (1990, in press). Deductive and inductive learning in a connectionist deterministic parser. In *Proceedings of the International Joint Conference on Neural Networks,* Washington, DC.

Fanty, M. (1985). Context-free parsing in connectionist networks, (*Technical Report 174*). Computer Science Department, University of Rochester, Rochester, NY, 1985.

Gallant, S.I. (1988). Connectionist expert systems. *Communications of the ACM,* **31**(2), 152–169.

Kitzmiller, C.T. & Kowalik, J.S. (1987). Coupling symbolic and numeric computing in knowledge-based systems, *AI Magazine,* **8**(2), 85–90.

Kwasny, S.C. (1980, November). Treatment of ungrammatical and extra-grammatical phenomena in natural language understanding systems, Indiana University Linguistics Club, Bloomington, IN.

Kwasny, S.C. & Faisal, K.A. (1989, November). *Connectionism and determinism in a rule-based natural language system.* (*Technical Report WUCS-89-31*). Department of Computer Science, Washington University, St. Louis, MO.

Kwasny, S.C. & Faisal, K.A. (1989, August) Competition and learning in a connectionist deterministic parser. In *Proceedings of the 11th Annual Conference of the Cognitive Science Society* (pp. 690–697). Hillsdale, NJ: Erlbaum.

Kwasny, S.C., Faisal, K.A., & Ball, W.E. (1990, in press). Unifying several natural language systems in a connectionist deterministic parser. In *Proceedings of the Western Multi Conference,* San Diego, CA.

Kwasny, S.C. & Sondheimer, N.K. (1981). Relaxation techniques for parsing ill-formed input. *American Journal of Computational Linguistics,* 7(2), 99–108.

Marcus, M.P. (1980). A theory of syntactic recognition for natural language. Cambridge, MA: MIT Press.

Milne, R. (1986). Resolving lexical ambiguity in a deterministic parser, *Computational Linguistics,* **12**(1), 1–12.

Rumelhart, D.E., Hinton, G., & Williams, R.J. (1986). Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland (eds.), *Parallel distributed processing* (pp. 318–364) Cambridge, MA: MIT Press.

Selman, B. & Hirst, G. (1985). A rule-based connectionist parsing system, In *Proceedings of the 7th Annual Conference of The Cognitive Science Society* (pp. 212–221). Hillsdale, NJ: Erlbaum.

Shapiro, S.C. & Neal, J.G. (1982, June). A knowledge engineering approach to natural language understanding. In *Proceedings of the 20th Annual Meeting of the Association for Computational Linguistics,* 136–144.

Waltz, D.L. & Pollack, J.B. (1985). Massively parallel parsing: A strongly interactive model of natural language interpretation. *Cognitive Science,* **9**, 51–74.

Weischedel, R.M. & Sondheimer, N.K. (1983). Meta-rules as a basis for processing ill-formed input, *American Journal of Computational Linguistics,* 9(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. & McGraw, K.L. (1989, April). SIGART—Special issue on knowledge acquisition, New York: ACM Press.

Winograd, T. & Flores, F. (1987). *Understanding computers and cognition* (p. 107). Reading, MA: Addison-Wesley.

Winston, P.H. (1984). *Artificial intelligence.* Reading, MA: Addison-Wesley, 1984.

APPENDIX A

Grammar Rules

1. **Rule Initial**	If	stack is empty
	then	Creat 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 Final Punct.
	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	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–
AU-Perf	–	–	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
AUX-Prog	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–
AUX-Pass	–	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–
AUX-Modal	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–
AUX-DO	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–
Create																						
S	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
NP	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–
VP	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–	–	–
AUX	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Insert-You	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Switch	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Drop	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	+	–	–	+	–	–	–
Stop	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+