OREGON STATE
UNIVERSITY
COMPUTER
SCIENCE
DEPARTMENT

SIMPLIFYING DETERMINISTIC PARSING

Alan W. Carter
Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1W5

Michael J. Freiling
Department of Computer Science
Oregon State University
Corvallis, OR 97331

84-30-1

CS-TR-84-30-1

SIMPLIFYING DETERMINISTIC PARSING

by

Alan W. Carter
Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1W5

Michael J. Freiling
Department of Computer Science
Oregon State University
Corvallis, OR 97331

To appear in the 10th International Conference on Computational Linguistics
July 2-6, 1984, Stanford University, Stanford, CA.

Simplifying Deterministic Parsing

Alan W. Carter[1]
Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1W5

Michael J. Freiling[2]
Department of Computer Science
Oregon State University
Corvallis, OR 97331

ABSTRACT

This paper presents a model for deterministic parsing which was designed to simplify the task of writing and understanding a deterministic grammar. While retaining structures and operations similar to those of Marcus's PARSIFAL parser [Marcus 80] the grammar language incorporates the following changes. (1) The use of productions operating in parallel has essentially been eliminated and instead the productions are organized into sequences. Not only does this improve the understandability of the grammar, it is felt that this organization corresponds more closely to the task of performing the sequence of buffer transformations and attachments required to parse the most common constituent types. (2) A general method for interfacing between the parser and a semantic representation system is introduced. This interface is independent of the particular semantic representation used and hides all details of the semantic processing from the grammar writer. (3) The interface also provides a general method for dealing with syntactic ambiguities which arise from the attachment of optional modifiers such as prepositional phrases. This frees the grammar writer from determining each point at which such ambiguities can occur.

1. INTRODUCTION

Marcus has effectively described the advantages of a deterministic parsing model as is embodied in his PARSIFAL system. Unfortunately a hindrance to the usability of PARSIFAL is the complexity of its grammar. The popularity of Woods' ATN parsing model [Woods 70] demonstrates that the ease with which a grammar can be written and understood is one of the greatest factors contributing to its usability. This paper describes DPARSER (Deterministic PARSER) which is an implementation of an alternate deterministic parsing model intended to reduce the complexity of deterministic grammars.

DPARSER has been implemented and a small grammar written. In developing the grammar the focus has been on dealing with the syntactic ambiguities between the attachment of phrases and thus it can currently handle only simple noun and verb phrases.

2. CONSTITUENT BUFFER

DPARSER maintains a constituent buffer which is manipulated by the grammar to derive the constituent structure of the input sentence. Each node of the buffer contains a constituent consisting of a set of feature-type, feature-value pairs, and a set of subconstituents. When parsing begins the constituent buffer contains a single node with an associated subgrammar for parsing sentence constituents. As the subgrammar of the sentence node examines the buffer positions to its right, words are brought in from the input sentence to fill the empty positions. When the grammar discovers a subconstituent phrase to be parsed, it performs a PUSH operation specifying a subgrammar for parsing the constituent and the position of the rightmost word in the constituent phrase. The PUSH operation inserts a new node into the buffer immediately preceding the constituent phrase and begins executing the specified subgrammar. This subgrammar may of course perform its own PUSH operations and the same process will be repeated. Once the subconstituent is complete control returns to the sentence node and the buffer will contain the parsed constituent in place of those which made up the constituent phrase. The sentence node can now attach the parsed constituent removing it from the buffer. When all the subconstituents of the sentence node have been attached

[1] supported in part by an I.W. Killam Predoctoral Fellowship

[2] supported in part by the Blum-Kovler Foundation, Chicago, Ill.

the parsing is complete.

To familiarize the reader with the form of the constituent buffer we consider the processing of the sentence *Jones teaches the course.* as the final NP is about to be parsed. Figure 1 shows the current state of each buffer node giving its position, state of execution, essential syntactic features, and the phrase which it dominates so far. Following the terminology of Marcus we refer to the nodes which have associated subgrammars as *active nodes* and the one currently executing is called *the current active node.* All buffer positions are given relative to the current active node whose position is labeled "*" .

The buffer in its current state contains two active nodes: the original sentence node and a new node which was created to parse the sentence predicate (i.e. verb phrase and its complements). The next modification of the buffer takes place when the subgrammar for the predicate node examines its first position causing the word *the* to be inserted in that position. At this point a bottom-up parsing mechanism recognizes that this is the beginning of a noun phrase and a PUSH is performed to parse it; this leaves the buffer in the state shown in Figure 2 .

The subgrammar for the noun phrase now executes and attaches the words *the* and *course.* It then examines the buffer for modifiers of the simple NP which causes the final punctuation, ".", to be inserted into the buffer. Since the period can not be part of a noun phrase, the subgrammar ends its execution, the PUSH is completed, and the predicate node again becomes the current active node. The resulting state of the buffer is shown in Figure 3; the words *the* and *course* have been replaced by the noun phrase constituent which dominates them.

Aside from PUSH and ATTACH, the following three operations are commonly used by the grammar to manipulate the constituent buffer.

LABEL   label a constituent with a syntactic feature
MOVE   move a constituent from one position to another
INSERT   insert a word into a specified position

Examples of these actions are presented in the following section.

The differences between the data structures maintained by PARSIFAL and DPARSER are for the most part conceptual. PARSIFAL's active nodes are stored in an *active node stack* which is separate from the constituent buffer. To allow active nodes to parse constituent phrases which are not at the front of the

```
POSITION -1   active
    SYNCLASS S   SENT-TYPE DECL
    (Jones)
POSITION *   current active
    SYNCLASS PRED   VTYPE ONE-OBJ
    (teaches)
UNSEEN WORDS: the course .
```

Figure 1. before pushing to parse the NP

```
POSITION -2   active
    SYNCLASS S   SENT-TYPE DECL
    (Jones)
POSITION -1   active
    SYNCLASS PRED   VTYPE ONE-OBJ
    (teaches)
POSITION *   current active
    SYNCLASS NP
    ()
POSITION 1   not active
    SYNCLASS DET   WORD THE   EXT DEF
    (the)
UNSEEN WORDS: course .
```

Figure 2. parsing the noun phrase

```
POSITION -1   active
     SYNCLASS S   SENT-TYPE DECL
     (Jones)
POSITION *   current active
     SYNCLASS PRED   VTYPE ONE-OBJ
     (teaches)
POSITION 1   not active
     SYNCLASS NP   NVFORM N3PS
     (the course)
POSITION 2   not active
     SYNCLASS FINAL-PUNCT   WORD .
     (.)
```

Figure 3. after the push is completed

buffer an offset into the buffer can be associated with an active node. The control of which active node is currently executing is affected through operations which explicitly manipulate the active node stack.

Church's deterministic parser, YAP [Church 80], uses a constituent buffer consisting of two halfs: an *upper buffer* and a *lower buffer*. The grammar rules try to attach nodes from the lower buffer to those in the upper buffer. While this structure is similar to PARSIFAL's, it does not draw such a rigid distinction between active and inactive nodes. There are no separate subgrammars associated with the nodes which constituents are being attached to, and nodes may be moved freely from one buffer to the other allowing them to be attached before they are complete. While our constituent structure does maintain active nodes with separate subgrammars, the control of the parsing process is similar to that used by Church in that it is possible for incomplete nodes to be attached. As will be seen in a latter section this is an essential feature of DPARSER's constituent buffer.

3. SEQUENCES

In DPARSER each constituent is assigned a *sequence*. Each sequence consists of a list of steps which are applied to the buffer in the order specified by the sequence. A *step operator* indicates how many times each step can apply: steps marked with "+" need never apply, those marked by "=" must apply once, and those marked by "*" can apply any number of times. A step may call another sequence which has the effect of inserting immediately following that step, the steps of the named sequence.

Each step consists of a list of rules where the priority of the rules are made explicit by their ordering in the list. Each rule is of the form

$$[p_1]\ [p_2] \dots [p_n] \rightarrow (a_1)\,(a_2) \dots (a_n) \quad .$$

Each precondition, p_i, tests a buffer node for the presence or absence of specified feature-type, feature-value pairs. When a rule is applied each action, a_i, is evaluated in the specified order. In attempting to apply a step each of the step's rules is tested in order, the first one whose preconditions match the current buffer state is performed.

In order to recognize certain constituent types bottom-up, sequences may be associated with a bottom-up precondition. When the parser encounters a node which matches such a precondition, a PUSH to the sequence is performed. This mechanism is equivalent to PARSIFAL's attention shifting rules and is used primarily for parsing noun phrases.

In order to clarify the form of a sequence, the example sequence TRANS-MAJOR-S shown in Figure 4 is discussed in detail. This sequence is associated with the initial sentence node of every input sentence. It performs the operations necessary to reduce the task of parsing an input sentence to that of parsing a normal sentence constituent as would occur in a relative clause or a sentence complement. While this sequence will misanalyze certain sentences it does handle a large number through a small set of rules.

STEP 1 handles the words *which* and *who* which behave differently when they appear at the beginning of a sentence. The first rule determines if *which* is the first word; if it is then it labels it as a determiner. The second rule handles *who* which is labels as a NP.

```
STEP: 1  +
      [1 WORD WHICH] -> (LABEL 1 {SYNCLASS DET} {EXT WH})
      [1 WORD WHO] -> (LABEL 1 {SYNCLASS NP} {EXT WH})
STEP: 2  =
      [1 EXT WH] ->
                (LABEL * {SENT-TYPE QUEST} {QUEST-TYPE NP})
      [1 SYNCLASS NP] -> (LABEL * {SENT-TYPE DECL})
      [1 ROOT HAVE][2 SYNCLASS NP][3 TENSE TENSELESS] ->
                (LABEL * {SENT-TYPE IMPER})
      [1 VTYPE AUXVERB] ->
                (LABEL * {SENT-TYPE QUEST} {QUEST-TYPE YN})
      [1 TENSE TENSELESS] -> (LABEL * {SENT-TYPE IMPER})
STEP: 3  +
      [1 EXT WH][2 VTYPE AUXVERB][3 SYNCLASS NP]
                [4 NOT PTYPE FINAL] -> (MOVE 1 WH-COMP)
STEP: 4  +
      [* QUEST-TYPE (YN NP-QUEST)] -> (MOVE 2 1)
      [* STYPE IMPER] -> (INSERT 1 you)
```

Figure 4. SEQUENCE TRANS-MAJOR-S

STEP 2 examines the initial constituents of the sentence to determine whether the sentence is imperative, interrogative, declarative, etc. . Since each sentence must be analyzed as one of these types the step is modified by the "=" operator indicating that one of the step's rules must apply. The first rule tests whether the initial constituent of the sentence is a WH type NP; NP's like *who, which professor, what time,* etc. fall into this category. If this precondition succeeds then the sentence is labeled as a question whose focus is a noun phrase. The second rule tests for a leading NP and, if it is found, the sentence is labeled as declarative. Note that this rule will not be tested if the first rule is successful and the step depends on this feature of step evaluation. The following rule tries to determine if *have*, appearing as the first word in a sentence, is a displaced auxiliary or is the main verb in an imperative sentence. If the rule succeeds then the sentence is labeled as imperative, otherwise the following rule will label any sentence beginning with an auxiliary as a yes/no type question. The final rule of the step labels sentences which begin with a tenseless verb as imperatives.

STEP 3 picks up a constituent which has been displaced to the front of the sentence and places it in the special WH-COMP register. Generally a constituent must have been displaced if it is a WH type NP followed by an auxiliary followed by another NP; however, an exception to this is sentences like *Who is the professor?* in which the entire sentence consists of these three constituents.

STEP 4 undoes any interrogative or imperative transformations. The first rule moves a displaced auxiliary around the NP in sentences like *Has Jones taught Lisp ?* and *When did Jones teach Lisp ?*. Note that for the latter sentence the previous step would have picked up *when* and hence *did* would be at the front of the buffer. The second rule of this step inserts *you* into the buffer in front of imperative sentences.

Like DPARSER, PARSIFAL's grammar language is composed of a large set of production rules. The major difference between the two languages is how the rules are organized. PARSIFAL's rules are divided into packets several of which may be active at once. At any point in the parsing each of the rules in each active packet may execute if its precondition is matched. In contrast to this organization, DPARSER's sequences impose a much stronger control on the order of execution of the productions.

Aside from the bottom up parsing mechanism the only competition between rules is between those in the individual steps. The purpose of constraining the order of execution of the productions is to reflect the fact that the parsing of a particular constituent type is essentially a sequential process. Most of the rules involved in the parsing of a constituent can only apply at a particular point in the parsing process. This is particularly true of transformational rules and rules which attach constituents. Those rules which can apply at various points in the parsing may be repeated within the sequence so that they will only be tested when it is possible for them to apply and they will not be allowed to apply at points where they should not. Clearly the necessity to repeat rules at different points in a sequence can increase the size of the grammar; however, it is felt that a grammar which clearly specifies the possible set of actions at each point can be more easily understood and modified.

4. SEMANTIC PROCESSING

While semantic processing was outside Marcus's central concern a semantic system was developed which operates in parallel with PARSIFAL , constructing the semantic representation as its subconstituents were attached. In order to deal with syntactic ambiguities the action part of rules can contain semantic tests which compare the semantic well-formedness of interpretations resulting from a set of possible attachments. Such comparative tests can choose between one or more constituents to attach in a particular syntactic role; for example a rule for attaching a direct object can use such a test to choose whether to attach a displaced constituent or the next constituent in the buffer. Comparative tests can also be used to decide whether to attach an optional modifier (such as a prepositional phrase) or leave it because it better modifies a higher level node. Unfortunately this latter class of tests requires each rule which attaches an optional modifier to determine each node which it is syntactically possible to attach the node to. Once this set of syntactically possible nodes is found, semantics must be called to determine which is the best semantic choice. Such tests complicate the grammar by destroying the modularity between the subgrammars which parse different constituent types.

For the LUNAR system [Woods 73] Woods added an experimental facility to the basic ATN framework which allowed an ATN to perform such comparative tests without requiring them to be explicitly coded in the grammar. The Selective Modifier Placement mechanism was invoked upon completion of an optional modifier such as a PP. It then collected all the constituents which could attach the modifier and performed the attachment it determined to be the best semantic fit. A mechanism similar to this is incorporated as a central part of DPARSER and is intended to be used whenever an attachment is locally optional. Before giving the details of this mechanism we discuss the semantic interface in general.

In DPARSER a narrow interface is maintained between syntax and semantics which alleviates the grammar writer of any responsibility for semantic processing. The interface consists of the ATTACH action which immediately performs the specified attachment and the IF-ATTACH test which only succeeds if the attachment can be performed in light of the other constituents which may want to attach it.

Both ATTACH and IF-ATTACH have the same parameters: the buffer position of the constituent to be attached and a label identifying the syntactic relationship between the constituent and its parent. Such a label is equivalent to a "functional label" of the RUS system [Bobrow & Webber 80]. When an attachment is performed the semantic system is passed the parameters of the attachment which it then uses to recompute the interpretation of the current active node.

IF-ATTACH tests are included as the final precondition of those grammar rules which wish to attach a trailing modifier; the test returns true if it is syntactically possible for the modifier to be attached and the modifier best semantically modifies that node. If the test is true then the attachment is performed as a side effect of the test.

To the grammar writer the IF-ATTACH test has the prescient capability to foresee which active node should be allowed to attach the modifier and immediately returns true or false. However, the implementation requires that when an IF-ATTACH test is performed, the current active node must be suspended and the node which pushed to it restarted. This node can then execute normally with the suspended active node appearing like any other node in the buffer. The node continues executing until it either completes, in which case the process continues with the next higher active node, or it encounters the IF-ATTACHed node. If, at this point, the active node issues another IF-ATTACH then this new request is recorded with the previous ones and the process continues with the next higher active node. This sequence of suspensions will end if an active node becomes blocked because it expects a different constituent type than the one in the position of the IF-ATTACHed node. When this occurs the interpretations which would result from each of the pending IF-ATTACH tests are computed and the attachment whose interpretation the semantic system considers to be the most plausible is performed. Alternately, a sequence of suspensions may be terminated when an active node ATTACHes the node that the suspended active nodes had tried to IF-ATTACH. Such a situation, an example of which occurs in the parsing of the sentence *Is the block in the box?*, indicates that the pending IF-ATTACH requests are syntactically impossible and so must fail.

The following example shows how the IF-ATTACH mechanism is used to handle sentences where the attachment of a prepositional phrase is in question. We consider the parsing of the sentence *Jones teaches* *the course in Lisp.* We start the example immediately following the parsing of the PP (Figure 5). At this point the sequence for the noun phrase is about to apply the rule shown in Figure 6 which tries to attach PP modifiers. Since the precondition preceding the IF-ATTACH test is true the IF-ATTACH test is made. This causes the current active node to be suspended until it can be decided whether the attachment can be performed (Figure 7).

Control now returns to the predicate node which attaches the suspended NP as the object of the verb. As normally occurs after an attachment, the NP node is removed from the buffer; however, because the node will eventually be restarted it retains a virtual buffer position. The sequence for parsing the predicate now applies the same IF-ATTACH rule (Figure 6) to attach any prepositional phrase modifiers. Again since the PP is the first constituent in the buffer the IF-ATTACH test is performed and the predicate node is suspended returning control to the sentence active node (Figure 8).

When the sentence node restarts it execution, it attaches the predicate of the sentence leaving the buffer as shown in Figure 9. Having found a complete sentence the sentence node executes a final step which expects to find the final punctuation; since there is none the step fails. This failure triggers the arbitration of the set of pending IF-ATTACH requests for the attachment of the PP. In this case the semantic system determines that the PP should modify the NP. The parser then restarts the NP node at the point where it issued the IF-ATTACH and allows it to make the attachment (Figure 10). The NP node then tries again to attach a PP but seeing only the period it realizes that its constituent is complete and terminates.

```
POSITION -2   active
     SYNCLASS S   SENT-TYPE DECL
     (Jones)
POSITION -1   active
     SYNCLASS PRED   VTYPE ONE-OBJ
     (teaches)
POSITION *   current active
     SYNCLASS NP   NVFORM N3PS
     (the course)
POSITION 1   not active
     SYNCLASS PP
     (in Lisp)
UNSEEN WORDS: .
```

Figure 5. after the completion of 'in Lisp'

```
     [1 SYNCLASS PP][IF-ATTACH 1 PP-MOD] ->
```

Figure 6. rule for attaching prepositional phrases

```
POSITION -1   active
     SYNCLASS S   SENT-TYPE DECL
     (Jones)
POSITION *   current active
     SYNCLASS PRED   VTYPE ONE-OBJ
     (teaches)
POSITION 1   suspended active
     SYNCLASS NP   NVFORM N3PS
     (the course)
POSITION 2   not active
     SYNCLASS PP
     (in Lisp)
POSITION 3   not active
     SYNCLASS FINAL-PUNCT   WORD .
     (.)
```

Figure 7. after the NP has tried to attach the PP

```
POSITION *   active
     SYNCLASS S   SENT-TYPE DECL
     (Jones)
POSITION 1   suspended active
     SYNCLASS PRED   VTYPE ONE-OBJ
     (teaches)
DELETED   suspended active
     SYNCLASS NP   NVFORM N3PS
     (the course)
POSITION 2   not active
     SYNCLASS PP
     (in Lisp)
POSITION 3   not active
     SYNCLASS FINAL-PUNCT   WORD .
     (.)
```

Figure 8. after the PRED node has tried to attach the PP

```
POSITION *   current active
     SYNCLASS S   SENT-TYPE DECL
     (Jones teaches the course)
DELETED   suspended active
     SYNCLASS PRED   VTYPE ONE-OBJ
     (teaches the course)
DELETED   suspended active
     SYNCLASS NP   NVFORM N3PS
     (the course)
POSITION 1   not active
     SYNCLASS PP
     (in Lisp)
POSITION 2   not active
     SYNCLASS FINAL-PUNCT   WORD .
     (.)
```

Figure 9. after the subject and predicate have been attached

```
POSITION -1   active
     SYNCLASS S   SENT-TYPE DECL
     (Jones teaches the course in lisp)
DELETED   suspended active
     SYNCLASS PRED   VTYPE ONE-OBJ
     (teaches the course in Lisp)
DELETED *   current active
     SYNCLASS NP   NVFORM N3PS
     (the course in Lisp)
POSITION 1   not active
     SYNCLASS FINAL-PUNCT   WORD .
     (.)
```

Figure 10. after the PP is attached

Next the monitor restarts the predicate active node but does not allow it to make the attachment. This results in the node eventually terminating without performing any more actions. At this point each of the IF-ATTACH requests have been processed and the step whose failure caused the processing of the requests is retried. This time it is successful in finding the final punctuation and attaches it. The parse is now complete (Figure 11).

Aside from prepositional phrase attachment there are many other situations where optional modifiers can arise. For example in the sentence *I saw the boy using the telescope* the phrase *using the telescope* may modify *boy* as a relative clause where the relative pronoun has been deleted, or it may modify *saw* where

```
POSITION *    current active
      SYNCLASS S   SENT-TYPE DECL
      (Jones teaches the course in Lisp .)
```

Figure 11.

the preposition *by* has been deleted. Another example is the sentence *Is the block in the box?* . In this sentence the PP *in the box* must, for syntactic reasons, complement the verb; however, in the local context of parsing the NP *the block*, it is possible for the PP to modify it. IF-ATTACH can easily be extended to attach optional pre-modifiers; it could then be used to derive the internal structure of such complex noun phrases as *the Lisp course programming assignment.*

The IF-ATTACH test is proposed as a mechanism to solve this general class of problems without requiring the grammar writer to explicitly list all constituents to which an unattached constituent can be attached. Instead, it is sufficient to indicate that a trailing modifier is optional and the monitor does the work in determining whether the attachment should be made.

5. CONCLUSION

A grammar language for deterministic parsing has been outlined which is designed to improve the understandability of the grammar. Instead of allowing a large set of rules to be active at once, the grammar language requires that rules be organized into sequences of steps where each step contains only a small number of rules. Such an organization corresponds to the essentially sequential nature of language processing and greatly improves the perspicuity of the grammar. The grammar is further simplified by means of a general method of interfacing between syntactic and semantic processing. This interface provides a general mechanism for dealing with syntactic ambiguities which arise from optional post-modifiers.

REFERENCES

Bobrow, R.J. and B.L. Webber [1980] "PSI-KLONE: Parsing and Semantic Interpretation in the BBN Natural Language Understanding System", in Proceedings of the CSCSI/SCEIO Conference 1980.

Carter, A.W. [1983] "DPARSER -- A Deterministic Parser", Masters Thesis, Oregon State University.

Church, K.W. [1980] *On Memory Limitations in Natural Language Processing,* MIT/LCS Technical Report #245., Cambridge, Mass.

Marcus, M.P. [1976] "A Design for a Parser for English", in Proceedings of the ACM Conference 1976.

Marcus, M.P. [1980] *A Theory of Syntactic Recognition for Natural Language,* The MIT Press, Cambridge, Mass.

Rustin R. [1973] *Natural Language Processing,* Algorithmics Press, New York.

Woods, W.A. [1970] "Transition Network Grammars for Natural Language Analysis", in Communications of the ACM 13:591.

Woods, W.A. [1973] "An Experimental Parsing System for Transition Network Grammars", in [Rustin 73].