Slot Grammars

Michael C. McCord

Computer Science Department
University of Kentucky
Lexington, Kentucky 40506

This paper presents an approach to natural language grammars and parsing in which slots and rules for filling them play a major role. The system described provides a natural way of handling a wide variety of grammatical phenomena, such as WH-movement, verb dependencies, and agreement.

1. Introduction

This paper presents a formalism for natural language grammars, with accompanying parser. The grammars are called *slot grammars* because they are organized around slots (grammatical relations) and rules for filling them. The parser works bottom-up and maintains, for each phrase being built up, a list called the *available slots* list, ASLOTS. A phrase can grow by having one of the slots in its ASLOTS list filled by a suitable adjoining phrase.

As a phrase grows, its ASLOTS list generally shrinks, because slots are ordinarily removed from ASLOTS as they get filled. However, a slot can be marked as *multiple* and then receive more than one filler. A more interesting exception to the shrinking of ASLOTS is that the procedure for filling a slot may operate on ASLOTS itself and add new slots to it. The operation of *raising* builds such new slots as "copies" of slots in the ASLOTS list of a filler phrase. Certain standard grammatical constructions, such as WH-movement, can be handled with this raising operation.

The parser processes the words of a sentence from left to right, at each stage working out all the slot-fillings that develop when the new word is thrown in with the phrases that have already been built up. However, a given phrase grows *middle-out.* Its history begins with a word which is its *head,* and its slot-fillers may be adjoined on the left or the right. A left-adjunction, if appropriate, is made immediately, because the filler already exists; but a right-adjunction waits till more words have been processed. Middle-out construction allows more data-directed control. For instance, the initial value of the ASLOTS list of a phrase is determined partially by the lexical entry for its head word.

In computational linguistic background, the system is most closely related to the augmented phrase structure grammars (APSG's) of George Heidorn (1972,1975). In APSG's, syntactic and semantic slots (relation attributes) are heavily used, though not as systematically as in slot grammars, because the APSG system does not maintain an ASLOTS list. The APSG parsing algorithms are bottom-up; and in the sample grammars, phrases are usually built up in a middle-out fashion, starting with a head word and adjoining items on the left or the right.

Although slot grammars are organized mainly around slots, they also make use of *states,* and thus have a relationship to the augmented transition networks (ATN's) of Woods (1970,1973). But the use of states in slot grammars is much more constrained than in ATN's, and, in general, slot grammars are contrasted with ATN's in the paper.

On the linguistic side, the theory proposed is most closely related to work in the systemic grammar tradition (Hudson, 1971,1976; McCord, 1975, 1977), especially to Hudson's theory of *daughter-dependency* grammar (Hudson, 1976).[1] The work of Kac (1978) is also related; and there are some connections to the tradition of Kenneth Pike and Charles Fries (Cook, 1969), at least in the basic notion of *slot* and *filler.*

The paper is intended as a contribution to natural language syntax and parsing. Very little is said about semantics. However, the system could readily

[1] I wish to thank Richard Hudson for many useful discussions pertinent to the present work.

0362-613X/80/010031-13$01.00

be augmented with procedures that build up semantic interpretations along with syntactic analyses. In such a "complete" system, semantic and pragmatic knowledge would be applied concurrently with syntactic knowledge; but syntax would still play a guiding role in the processing.

Section 2 of the paper, *The centrality of slots,* argues for the advantages of an ASLOTS list, mainly in connection with verb dependencies, unbounded movement rules, and conjunctions. Section 3, *States and slots,* explains how states are used and basically how slot-filling takes place. A simple diagrammatic notation for slot grammars is introduced. Section 4, *Formal representation of syntax,* describes the form of the input of syntax to the program (which is written in LISP). Section 5, *Representation of frames by the system,* gives details of the data structures used by the system. Section 6, *The lexicon,* describes the formal representation of the lexicon, and argues for some of the advantages of data-directed control. Section 7 is an *Outline of the parsing algorithm.* Section 8 gives *A sample grammar* and discusses some of the linguistic choices made in it. Section 9 is a *Summary* of the characteristics of the system.

2. The centrality of slots

In natural language parsing, common control devices are the use of states (as in transition networks) and the examination of individual slots and flags. These devices are used in slot grammars, but in a restrained way. The most central control device is the maintenance of the *available slots* list, ASLOTS. The claim of this section is that this is linguistically and computationally natural, especially in conjunction with bottom-up parsing and middle-out construction of phrases.

The ideas will be illustrated with the formation of verb phrases (VP's). Following Heidorn (1972, 1975), I use this term to include a verb with *any* of its sisters, even the subject. The data structure used by the slot grammar system for analyzing a VP, during parsing, is called the VP *frame.* This is an association list of *registers* and their *values,* much as is used in ATN parsing (Woods, 1973). The values of registers can be procedures as well as "declarative" structures. There is some parallel of characteristics of these frames with the frames of Minsky (1975) and Winograd (1975). Complete details will be given in Section 5.

The main register of concern now is ASLOTS. The initial ASLOTS register for a VP frame might contain the list (SUBJ IOBJ OBJ ADVL). If the SUBJ slot can be filled, then the system forms a new VP frame showing SUBJ filled and having its ASLOTS reduced to (IOBJ OBJ ADVL). Some slots, such as ADVL (adverbial), may be marked as *multiple* slots in the grammar, and these are not removed from ASLOTS when they are filled. The members of ASLOTS are in general *optionally* filled. Any checking for obligatory slots must be done explicitly in the grammar. Although ASLOTS is stored as a list, it is treated as an unordered set; the position of a slot in ASLOTS has no effect on whether it can be filled.

One advantage of this approach is that one can express verb-dependencies in an immediate and simple way. Instead of classifying verbs by features like *transitive,* one can just initialize the ASLOTS register of the VP frame so that it contains the slot OBJ. The initialization information that is special to a given verb is stored in the lexical entry for the verb, in a list of slots called the *sister-dependency* list of the verb. (These slots correspond roughly to *sister-dependency* rules in the theory of Hudson, 1976.) For example, the sister-dependency list stored with the verb *give* might be (IOBJ OBJ). When a VP frame is formed with *give* as its head, its initial ASLOTS will include (IOBJ OBJ). Certain other slots, such as SUBJ, AUXL (auxiliary), and ADVL, are common to all verbs, so it would be redundant to list them in the lexicon. These are *default* slots and are listed in the general syntax of the VP. (These slots correspond roughly to *daughter-dependency* rules in Hudson, 1976.) In setting up the initial value of ASLOTS, the parser automatically combines the default slots with the sister-dependency slots of the particular verb, so that the initial VP frame for *give* would have ASLOTS = (SUBJ AUXL ADVL IOBJ OBJ).

This treatment of verb-dependencies is more direct than the use of transitivity features or encoding in transition network states, because this initial ASLOTS list expresses more directly what the verb "needs" to be the head of the VP. The semantic interpretation of the VP should be built (partially) from these slots and their fillers, and the syntax of the VP is guided by the filling of these particular slots. Furthermore, this method ties in nicely with the middle-out construction of the VP; search proceeds outward from the item that sets the goals.

Not only does the slot grammar system initialize ASLOTS appropriately, but it also updates ASLOTS as parsing proceeds. At any point, ASLOTS provides a natural expression of what remains to be adjoined to the VP. Most parsers (e.g. ATN and APSG parsers) keep track of what slots *have* been filled, but it seems reasonable also to keep track of what slots may *yet* be filled, and use these in the control mechanism. Then rules that might be applied to fill a slot like OBJ never become activated if OBJ is not available.

For instance, Heidorn (1972) has a rule roughly like the following:

VP(TRANS,¬OBJ) NP --> VP(OBJ=NP).

This says that when a transitive VP with OBJ slot unfilled is followed by an NP, then a new VP is formed with OBJ filled by the NP. The rule will be tested every time a VP is formed, and this will be fruitless if the verb is not transitive (cannot take an OBJ) or if it already has an OBJ. Notice that OBJ is (implicitly) mentioned *three* times (counting the TRANS) in the rule, whereas one feels somehow that OBJ should be mentioned only once, since the rule is about filling the OBJ slot. Furthermore, if one had a slot that could be filled by more than one kind of filler (not just an NP) then this sort of rule would have to be duplicated for each type of filler.

The appropriateness of basing search on an available-slots list seems especially clear in a language like Japanese with a rather free order of VP constituents. Suppose a grammar is to be written which captures the simple idea that the verb comes at the end of the VP, and the preceding NP's have case markings and can come in any order. In a slot grammar, the verb can activate a VP frame which has an ASLOTS list appropriate for that verb. Then the VP frame "looks to the left", filling slots in ASLOTS, and removing non-multiple slots from ASLOTS as it goes. In a situation that starts with, say, four slots and removes all but one, only this one slot will be relevant for further expectations in looking to the left, and rules will not be attempted needlessly.

Still another reason for basing expectations on ASLOTS has to do with the way *raising* constructions can be treated in bottom-up, middle-out analysis. Many languages allow unbounded raising of items, as in

(1) Which chair does Mary believe John said he was sitting in?

Here the question arises as to what syntactic role the initial NP *which chair* fills. Two VP levels and a PP down, there is a slot OBJ which is the object of the preposition *in*. Does *which chair* fill OBJ directly? If we try to write rules which accomplish this, we have to make them search down VP chains of arbitrary length and be aware of possible branching due to conjunctions, as in

(2) Which chair does Mary believe that Al bought and John was sitting in?

It seems that the rule for filling the object of the preposition should not have to "know about" these complications. The complications are created by VP complementation of verbs like *believe* and by conjunctions like *and*. The constructions that *create* the complications should take responsibility and should smooth the way for the placing of *which chair*.

In slot grammars this is handled by the operation of *raising* slots. Every slot has a procedure attached to it called its *slot-rule*, which can test for the sorts of fillers the slot might have and can perform actions. RAISE is a possible action, and is illustrated as follows. Consider a sentence like

(3) Which chair does Mary believe that Al bought?

The VP frame for *believe* has a slot COMP (verb-complement) which can be filled by another VP. To the right of *believe* is a VP *that Al bought*. This VP is "incomplete" in the sense that its ASLOTS register still contains a slot OBJ. In the slot-rule for COMP there is an instruction to RAISE all members of the filler's ASLOTS that belong to a specified list. (Some slots, such as verb auxiliaries, are not raised by COMP.) Raising a slot means creating a *new* member of the matrix VP's ASLOTS which is a sort of "image" of the lower slot. It has the same slot-rule and it is marked as being associated with the lower slot. A slot may be raised through several levels, but a path showing its origin is maintained for the purpose of semantic interpretation.

In sentence (3) when the COMP slot for *believe* raises the lower OBJ to a new slot OBJ1, this is available to be filled by *which chair* at a certain stage when the top VP is looking to the left.

The WH-movement that appears in sentences (1),(2), and (3) is a special kind of *unbounded left movement* (the left-dislocated item can be moved out of an unbounded number of embedded VP's). Another kind is *topicalization*, as in

(4) This chair, she said you could put in the room.

Raising applies to unbounded left movement in general, and in fact the same RAISE operation invoked by the VP COMP slot is used for handling both (3) and (4).

In ATN grammars, unbounded left movement is handled by the HOLD facility (Woods, 1970, 1973). The ATN puts the left-dislocated item (like *this chair* in (4)) on a special stack by the HOLD action, and then at a later opportune time removes it from the stack while traversing a *virtual* arc --- in the case of (4), an arc parallel to the verb-object-NP arc --- so that *this chair* becomes the object of *put*.

The HOLD method does not mix well with bottom-up parsing, however, because it depends on using the complete left context at each point. (The item retrieved on a virtual arc could have been held anywhere from the beginning of the sentence.) Since bottom-up, middle-out analysis appears to be best for natural language (as this paper attempts to show), and since RAISE is a viable alternative to HOLD, we have an argument against HOLD.

Furthermore, raising appears to be more generally applicable than HOLD. As hinted at in the discussion of (2) above, conjunction constructions should also involve raising. In that sentence, the

and frame should be responsible for creating the conjoined VP frame spanning *that Al bought and John was sitting in,* whose ASLOTS contains a slot OBJ1 which is related to both the object of *bought* and the object of *in,* by raising. This OBJ1 is further raised by the COMP slot of *believe* to a slot which is finally filled by *which chair.*

The details for raising by conjunctions have not been completely worked out, but the general situation seems to be roughly as follows. When a conjunction frame sees two frames of the same category on either side (the two conjuncts), it should construct raised slots corresponding to the *intersection* of the ASLOTS lists of the conjuncts. (In calculating the intersection, two slots that are already raised are considered equal if they originated from the same slot.) For example, in the sentence

(5) John ate and slept.

we could consider the *ate* frame to have ASLOTS = (SUBJ AUXL ADVL OBJ), but the *slept* frame would have ASLOTS = (SUBJ AUXL ADVL). The intersection would be (SUBJ AUXL ADVL), and these slots would be raised to slots (SUBJ1 AUXL1 ADVL1) in the conjoined VP *ate and slept.* Then *John* fills SUBJ1, to form the complete VP (5). There is no object slot available in the conjoined VP. On the other hand, the conjoined VP *cooked and ate* would have both a subject and an object slot available, and we could get

(6) John cooked and ate the pizza.

In Woods (1973) conjunctions were handled by a *system* facility designed specially for conjunctions --- meaning that the rules for conjunctions are not input by the grammar writer. The bottom-up, middle-out analysis with raising outlined above seems more straightforward and more controllable by the grammar writer. Consider a raising treatment possible for the following example discussed in Woods (1973):

(7) John drove his car through and
completely demolished a plate glass window.

The *and* frame has on its left the VP *drove his car through* with ASLOTS = (SUBJ AUXL ADVL OBJ1), where OBJ1 is raised from the OBJ slot in the incomplete PP by ADVL. To the right is the VP *completely demolished* having ASLOTS = (SUBJ AUXL ADVL OBJ). The *and* frame creates the conjoined VP *drove his car through and completely demolished,* having raised ASLOTS = (SUBJ1 AUXL1 ADVL1 OBJ2) corresponding to the essentially identical ASLOTS lists of the two conjuncts. Then SUBJ1 is filled by *John* and OBJ2 is filled by *a plate glass window,* for the analysis of the complete sentence.

3. States and slots

If all phrases had their heads at the beginning or end, and their other slots could be filled in any order, then all searching could be controlled by the unordered set ASLOTS. Many languages (including English) have an intricate combination of free placement of some slot-fillers with ordering restrictions on others. One conceivable method of controlling order would be to include tests in slot-rules for the position of the filler relative to other slot-fillers; but this seems to result in an unreasonable amount of testing, especially in languages in which there is a good deal of fixed order. It appears to be advisable to use some notion of "state" or "stage" in building phrases. In middle-out construction, another reason for using states is to control the *direction* in which the construction is proceeding; adjunctions might be made on the left, then the right, then switch directions again.

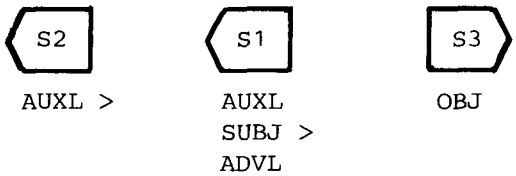
In a slot grammar, each phrase frame has a register STATE, which contains an atom somewhat like an ATN state. Each state has a *direction,* LEFT or RIGHT, associated (permanently) with it, the idea being roughly that if a phrase is in state S, then it is looking for fillers in the direction associated with S.

A restriction placed on states in slot grammars which makes their use much more constrained than in ATN's is that the set of states for a given phrase type (like VP) is *linearly ordered.* As a phrase gets built up, it can move ahead, but can never move back, in this ordering of states. Because of the linear order, the term *stage* might be more suggestive than *state.*

In the grammar, slots are related to states in the following way. Each slot is specified to be *attached* to one or more states. To fill a given slot with a proposed filler, one must be able to advance (or not move back) from the current state of the matrix phrase (along the linear order of states) to a state to which the slot is attached, with the direction of the state corresponding to the direction of the proposed filler.

The following diagram for a small VP grammar illustrates the use of states and slot attachment.

(8)



The states are S1, S2, and S3. Here, and in future examples, the integers in the state names indicate their linear order. States S1 and S2 have direction LEFT and S3 has RIGHT. Slots are written under the states to which they are attached. Note that AUXL is attached to both S1 and S2. The sign >

after a slot indicates that it is attached as a *state-advancer.* This means that if the slot is filled while the frame is in the given state, then the frame will advance to the next state (otherwise it stays in the given state). AUXL is attached to S2 as a state-advancer, but to S1 as a non-state-advancer. Slots ADVL and AUXL are multiple slots, although that is not shown in the diagram.

Here is an example of VP construction using VP grammar (8). The successive VP's constructed are underlined, and to the side of each underline is shown the slot *just* filled and the state the VP is in *after* the slot-filling.

```
Could Al have already left the bus?   (9)
                      ----          HEAD,S1
                ------------        ADVL,S1
          ------------------        AUXL,S1
      ----------------------        SUBJ,S2
--------------------------          AUXL,S3
-------------------------------- OBJ, S3
```

When SUBJ is filled at S1, the frame is advanced to S2, where it may get an AUXL in a question sentence. Several AUXL's may appear in state S1, but once the SUBJ has been filled, there is a chance for only one more AUXL, because an AUXL at S2 will advance the frame to S3. Also note that there is no chance for an ADVL between the SUBJ and the preposed question AUXL, as in

```
(10) *Could already Al have left the bus?
```

Consider another example:

```
(11)  Al has left the bus.
            ----          HEAD,S1
         --------         AUXL,S1
      -----------         SUBJ,S2
      ------------------ OBJ, S3
```

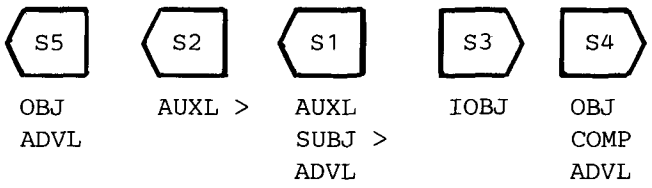
This illustrates, in the filling of OBJ, that a slot can be filled even when the frame is not yet in a state to which the slot is attached; it just has to be possible to *advance* to such a state S (only the first such is used). After the filling, if the slot is attached to S as a state-advancer, then the frame will be advanced to the next state after S; otherwise it stays in state S.

The use of states in slot grammars can be considered a generalization of some techniques used by Heidorn in APSG's. In the grammar of Heidorn (1972), a VP first works to the right getting all postmodifiers of the main verb, then works to the left getting all premodifiers. To control this, Heidorn used a register PRM (standing for "premodified") as follows. PRM is preset to *off.* Every rule that picks up a postmodifier checks that PRM is still *off,* and every rule that picks up a premodifier sets PRM to *on.* The slot grammar register STATE can be considered a generalization of PRM, in that its values are atoms that control direction of search.

In a recent APSG grammar for NP's, Heidorn[2] uses a technique which is even closer to our use of states.[3] He uses a register ML (standing for "modification level") which takes on integer values, and the numerical ordering is used in controlling the stages of building up an NP, allowing multiple direction changes. The left-hand sides of production rules often check that ML is less than or equal to a certain value, and the right-hand sides set ML to a certain value. This is similar to our requirement for *advancing* states in slot filling.

Now let us extend the VP grammar (8) to one which accepts a wider range of constructions.

(12)



Note that there are two direction switches in this grammar. First S1 and S2 go left; then there is a switch to the right with S3 and S4, and then a switch back to the left with S5. Reasons for this complication will be given below. The additional slots in this diagram are IOBJ and COMP. IOBJ (indirect object) accepts only NP's; the semantically equivalent to-form is accepted by ADVL at S4. (ADVL accepts, say, adverbs and PP's.) COMP (complement) has VP fillers.

This VP grammar is intended to capture the following intuitive description of a way of building up a VP. Starting at the head verb, we work left getting possible auxiliaries and adverbials. At some point, we may get a subject. If so, then there is a chance for one more auxiliary (in the case of a question sentence). Then we work to the right and may pick up an indirect object (with no other items intervening between it and the head verb). Then, still to the right, we pick up OBJ, COMP, or any number of ADVL's, in any order. Then, back to the left, we might find an OBJ or any number of ADVL's. Of course if OBJ has already been filled at S4, it will have been removed from ASLOTS and will not be available at S5. An example in which OBJ is filled at S5 is

```
(13)  Which chair   did   John   buy ?
      ----------    ---   ----   ---
         OBJ        AUXL  SUBJ   HEAD
```

[2] Private communication to the author.

[3] These two techniques were developed independently of each other.

Why are there two direction switches? Accepting for the moment the reasonableness of starting to the left with S1 and S2, why not continue left and make S5 the third state? The answer involves raising. In sentences like (1), (2), and (3), *which chair* fills an object slot raised from a VP found by COMP at S4. So S4 has to be visited before S5.

It still might seem that one could make only one direction switch by starting immediately to the right after the head verb, as was done in Heidorn (1972). One reason for going left initially has to do again with raising. The relative clause slot in the subject NP can be raised to the right of the head verb, as in:

(14) The man is here that I was telling you about.

Even if this right extraposition were not handled by the precise mechanism of raising, it seems reasonable that the subject should already be present in the VP before "placing" the extraposed modifier correctly.

Also, it seems plausible psychologically to go left first, because the auxiliaries and the subject are so closely related to the verb and their position usually identifies their role. But the role of a fronted item like *which chair* in sentences (1), (2), and (3) cannot be identified until a good deal of the rest of the sentence has been processed.

4. Formal representation of syntax

The interpreter-parser is written in LISP 1.6 running on a DEC-10. There are two functions, SYNTAX and LEXICON, which accept the grammar and preprocess it. They are both FEXPR functions (receiving their arguments unevaluated). The form of a call to SYNTAX will be described in this section.

SYNTAX is called for each phrase-type, such as VP, NP, and PP. The top-level form of a call is

```
(SYNTAX phrase-type
   STATES:
      state-specification ...
   SLOTS:
      slot-specification ...
   DEFAULTS:
      slot ... )
```

Before going into more details, let us look at an example, the formal specification of the grammar shown earlier in diagram (8).

```
(SYNTAX VP
  STATES:
    (S1 L) (S2 L) (S3 R)
  SLOTS:
    SUBJ
      (FLR NP) (S1 >)
    AUXL *
      (FLR AUX) (S1 S2 >)
    ADVL *
      (OR (FLR ADV) (FLR PP)) (S1)
    OBJ
      (FLR NP) (S3)
  DEFAULTS:
    SUBJ AUXL ADVL )
```

The general rules are as follows. The state-specifications are given in the order to be assigned to the states. The form of a state-specification is a list:

(name direction [test-action ...])

where the square brackets are metasymbols indicating optionality. The *name* is the name of the state and can be any LISP atom. The *direction* is L or R. A *test-action,* if given, is a LISP form which will be evaluated, and must give a non-NIL result, for a slot-filling to succeed, whenever the frame is advanced to the given state by the slot-filling. For example, suppose given the state-specification

```
(S5 L (IS SUBJ))
```

in a VP syntax. If an attempted slot-filling advances the frame to state S5, then the test (IS SUBJ) will have to succeed (meaning that the SUBJ slot is already filled) in order for the slot-filling to succeed.

The general form of a slot-specification is as follows:

name [*] slot-rule state-attachments

The optional star indicates that the slot is multiple. During parsing, the system takes care of removing non-multiple slots from ASLOTS as they get filled.

The slot-rule is a LISP form which can test for the sorts of fillers the slot can have, and perform actions. In the sample grammar above, the slot-rules use the test (FLR cat), which requires that the filler be of the category *cat*. No actions are shown in this grammar; but possible actions are calls to the RAISE function and the setting of registers, and these are exhibited in the grammar of Section 8.

The last part of the slot-specification is the state-attachments. The required form is

({state-name [>]} ...)

In other words, one writes a list of state names, each optionally followed by the sign >. If the sign > does follow the state, then the slot is attached as an advancing slot, otherwise as a non-advancing slot. The meaning of this for state transitions was discussed in the preceding section.

The last part of the call to SYNTAX is the sequence of default slots. These are collected by SYNTAX into a list and stored on the property list of the phrase-type, to be used as described in Section 2.

There are a few "primitive" functions (like FLR and RAISE) supplied for writing slot-rules and state test-actions. These will be described as they appear in examples below.

5. Representation of frames by the system

As mentioned earlier in Section 2, frames are stored as association lists:

({register value} ...)

Because of the non-determinism in the processing, I follow Woods (1973) in setting registers by just tacking on the new register/value pair onto the front of the frame.

There are several special registers known to the system. Two that have already been discussed extensively are ASLOTS and STATE. The others are as follows. CAT contains the atom which is the phrase-type, such as VP, or, in the case of words, the basic part of speech, such as V or N. WORD, in the case of lexical frames, contains the actual (inflected) word, and ROOT contains the root form. FEATURES contains the list of atoms treated as features. For example, a VP might have FEATURES = (QUESTION PROGRESSIVE).

LB and RB contain, respectively, the left boundary and right boundary of the phrase or word. A boundary is an atom representing the space between two words in the input sentence, or the start or end. (A phrase always represents an analysis of a connected segment of words in the sentence --- all the words between its left and right boundaries.)

FTEST stands for *filler-test* and contains a form which is evaluated (as a test-action) by the parser when the frame is tried as a filler. More details on this will be given in the next two sections.

The final system register is FSLOTS, which is used to hold the results of already filled slots. The value of FSLOTS is another association list, of the form

({slot filler} ...)

where each filler is of course another frame. The slot/filler pairs in FSLOTS are placed in accordance the actual positions of the fillers in the sentence. For instance, in the VP

Probably John left yesterday

FSLOTS would be of the form

(ADVL x SUBJ x HEAD x ADVL x).

Notice that in this sort of association list, the same register can occur more than once, and an earlier occurrence does not "hide" a later one. There is a system function

(SLOTSET slot filler direction)

which takes care of updating FSLOTS during slot-filling, putting the new pair on the correct side of FSLOTS. Maintaining FSLOTS as a reflection of surface order is useful for outputting parse trees, and it is also probably important for semantic interpretation.

Notice that the terms *register* and *slot* are being used in distinct ways. *Register* is the general term for one of the variables in our association lists. *Slots* are specific to the linguistic theory. Besides the special slot HEAD, they must be mentioned as slots in calls to SYNTAX; and any slot relevant to a given phrase frame will appear somewhere in its ASLOTS or FSLOTS.

Although slot/filler associations are all stored in the register FSLOTS, each slot is also used as a register in the phrase frame. As a register, a slot contains its slot-rule. SYNTAX stores the slot-rule of a slot on the property list of the slot (under the property RULE). But this is basically a default rule, and the system allows the lexicon to make exceptions, by information in the sister-dependency list for the head item. Thus, the slot-rule for COMP in the initial VP frame for a verb like *help* can be special to that verb. To allow this flexibility, the slot-rule for COMP is stored in the *register* COMP. Furthermore, it appears that the slot-rule for a given slot in a given phrase frame should actually be allowed to change while the phrase is being built up. Reasons for this will be given in the next section.

6. The lexicon

The lexicon is accepted and preprocessed by the LISP function LEXICON. Each member of the argument list is a lexical entry, of the form:

(word category [feature] ... [form] ...)

Examples are

```
(JOHN N SG PROPER)
(GIVE V (VM GIVES GIVING GAVE GIVEN)
        (SD (IOBJ) (OBJ)) )
```

Here VM and SD stand for "verb morphology" and "sister-dependencies", and are actually LISP functions.

What LEXICON accomplishes for each lexical entry is to produce frames associated with the words involved in the entry, and put them on the property lists of the words under the property LEX. These are frames for the word as filler, as well as initial frames for phrases in which the word is HEAD. For instance, the LEX list for HAS in the trial grammars consists of a word frame which might become a filler for the AUXL slot in some VP, as well as a VP frame in which HAS is the main verb.

The forms that appear at the end of a lexical entry (such as the VM and SD forms above) are evaluated by LEXICON and can add to the collection of frames being constructed. If no forms are

given, LEXICON will only construct a single word frame (for the word at the beginning of the entry).

Forms like VM add inflected words to the root word at the head of the entry, so that frames get constructed for all these words. I have not gone into spelling rules for regular inflections, but these could easily be added.

The SD form implements the ideas on sister-dependency slot lists discussed in Section 2. A call to SD has the form:

(SD {(slot [slot-rule])} ...)

An example is

```
(SD (OBJ) (COMP (FLR ADJ))).
```

The slots listed are of course the sister-dependency slots for the verb. The optional slot-rule after a slot will replace the slot-rule given for that slot in syntax; thus the latter should be considered a *default* slot-rule. The function SD constructs the initialized phrase frame in which the verb is HEAD. The (initial) ASLOTS list consists of the default slots from the VP syntax plus the slots specified in SD. Also, any test-actions associated with the first state of the VP are evaluated --- as if the HEAD advances the frame to the first state.

It was argued in Hudson (1977) that subject-verb agreement rules belong to morphology and not to syntax. The main point of the argument is that some verbs make more distinctions than others. Considering the standard six combinations of person and number, one notes that nearly all English verbs make a distinction only between the third person singular and the other combinations --- and this is only in the present tense. The exceptions are that the modals make *no* distinctions (in present or past), and the verb *be* makes three distinctions in the present and two in the past.

If we put subject-verb agreement in English syntax, we would presumably have to carry along enough distinctions of person and number to satisfy the fastidious verb *be.* On the other hand, if the finite verb is *gave* or *can,* there is no need for subject-verb agreement to come up at all. Another example is that some determiners require number agreement with the head noun in English, but for the most common one of all, *the,* there is no need for number agreement to enter the picture.

As with sister-dependency slots, this is a case where data-driven processing is called for, and all agreement rules are put in the lexicon. It was mentioned in the preceding section that the system knows about a frame register FTEST containing a test which must be satisfied when the frame is used as a filler. This is where we place the agreement check, and the lexicon can adapt it uniquely to the particular type of verb involved.

The FTEST employed for agreement uses a (FEXPR) function CHECK, which is called as follows:

(CHECK slot test)

For example, the filler frame for the verb *has* has in the FTEST register:

```
(CHECK SUBJ (NEGF IT PL))
```

Here the NEGF test requires that the subject does *not* have the feature PL (plural). It seems better to express it negatively, instead of requiring the SUBJ to have the feature SG (singular), so that for VP subjects as in

The boys' being there causes trouble

we will not have to say that the VP subject is SG.

When the finite verb is tried as a filler (either of AUXL or the VP HEAD) and (CHECK SUBJ test) gets evaluated, what happens? A problem is that the SUBJ may or may not have already been filled at this point, depending on whether we have certain question sentences or not. *If* SUBJ is already present, CHECK applies the test to the SUBJ filler on the spot. Otherwise, it adds the test to the slot-rule of SUBJ, by making a new SUBJ slot-rule of

(COND (test original-SUBJ-slot-rule)).

Being able to change slot-rules in this way is another reason for storing slot-rules in the slot as register, as was discussed at the end of the preceding section.

The lexical function VM actually takes responsibility for creating these CHECK's as necessary for all verbs besides *be* and the modals. For instance, VM will create a CHECK for GIVES, but none for GAVE.

Another example of data-driven processing which has been put into the lexicon is the set of requirements that English auxiliaries have on other auxiliaries and the main verb. In the VP syntax, there is simply a multiple slot AUXL, with no distinction between kinds of auxiliaries, their ordering, or their inflectional requirements. But there is the well-known sequence:

modal perfect-have prog-be passive-be main-verb

with the inflectional requirement that each auxiliary has on whatever verb follows it.

One alternative would be to have four slots MODAL PERF, PROG, and PASS. But a problem is that this clutters up ASLOTS quite a bit, so that a lot of slots would keep getting tried uselessly. It seems better to go more bottom-up and proceed from whatever verbs actually appear. The AUXL filler *be, if* it appears, can check whether the next verb to its right is an ing-form or en-form, and can declare that the VP is progressive or passive accordingly. This test-action is put into the lexical entry

for BE, and LEXICON makes it part of the FTEST for the *be* filler-frames.

One thing that *is* done in syntax to facilitate this testing is to keep a VP frame register VERB1 set to the current left-most verb. Each auxiliary has to check the features of VERB1. This will appear in the sample syntax given in Section 8.

The ordering of the auxiliaries is strict, and checks on this are also made in their filler-tests. Perhaps it is not even computationally necessary or psychologically real to do this in parsing; perhaps one could leave it to generation.

The multiple slot AUXL collects what could be thought of as *premodifiers* of the main verb. An analog in NP's is the multiple slot ADJC which collects premodifiers of the head noun, filled by certain types of adjectives, adjective phrases, and NP's. Here too, there are ordering restrictions as in *big red house* vs. **red big house,* although it would seem foolish to enshrine this in syntax by making lots of slots for different types of noun premodifiers. An example that makes AUXL look a little more free is that in some American dialects, more than one modal can be used, as in *might ought to do that,* or even *might should do that.*

7. Outline of the parsing algorithm

The parsing algorithm takes advantage of some preprocessing done by the function SYNTAX. The input to SYNTAX shows a linear order on the states and shows each slot attached to certain states. Recall (from Section 3) the conditions necessary for filling a slot SL when the matrix frame is in state ST, and the proposed filler is on, say, the left. There must be a state $s \geq ST$ such that SL is attached to s and the direction of s is LEFT. Suppose such an s exists. Let ST1 be the first such. If SL is attached to ST1 as a state-advancer, let STRANS be the successor state of ST1; otherwise let STRANS = ST1. If no s exists, let STRANS be NIL. Let us call STRANS the *left-transform* of state ST by slot SL. The *right-transform* is defined similarly. These state transforms are precalculated by SYNTAX, and stored on the property lists of the slots, thus saving on search time.

The heart of the parsing algorithm is a function

(MODIFY IT MATRIX DIR)

It constructs all frames which result when the frame IT modifies (fills a slot in) the frame MATRIX from the direction DIR. (DIR=LEFT means that IT is on the immediate left of MATRIX.)

MODIFY proceeds as follows. Let us assume that DIR = LEFT (the case DIR = RIGHT is entirely symmetric). Let ST be the current state of MATRIX. Then for each slot SL in the ASLOTS list of MATRIX, MODIFY determines whether IT can fill SL by making the following five tests, in the order given:

(a) The left-transform STRANS of ST by SL must be non-NIL.

(b) The slot-rule of SL is evaluated, and the result must be non-NIL. This result is called ACTION and is saved for use in test (d).

(c) The filler-test (the value of the FTEST register in IT) must evaluate to non-NIL.

(d) The ACTION must evaluate to non-NIL. (The reasons for this double evaluation of the slot-rule will be given below.)

(e) If STRANS is not equal to ST, then the test-action associated with STRANS is evaluated and must give a non-NIL result.

If these five tests are satisfied, then the frame MATRIX is updated as follows. SL is set to IT using SLOTSET, as in Section 5. ASLOTS is modified by the deletion of SL if SL is non-multiple. STATE is set to the left-transform STRANS. Finally, the left boundary of MATRIX is set to the left boundary of IT. The presence of this new version of MATRIX is recorded by a function INSERT, described below. Of course the old version of MATRIX stays around, for possible use in other modifications.

Note that tests (b) and (d) perform a *double* evaluation of the slot-rule: The value obtained in (b) should be another LISP form (ACTION), and this is further evaluated in (d). The reason for this is that the action performed by a slot-rule may disturb registers that must be examined by the filler-test, used in (c). This situation does not come up in the sample grammar of the preceding section, but it will be illustrated in the next section. (In the grammar of the preceding section, all slot-rules just evaluate to T if they do not give NIL, so the action, T, is trivial, and (d) will be satisfied if (b) is.)

The top level function, PARSE, of the parser takes a sentence, and processes its words left to right as follows. It creates boundary markers for the words (as it goes), and, for each boundary marker B, it stores on the property list of B, under the indicator RESULTS, the list of all frames produced so far whose right boundary is B.

For each new word W, PARSE looks on the LEX list of frames associated with W (produced by the lexicon). If this list is empty, W is not in the lexicon and parsing is halted with an error message. Otherwise, PARSE calls the function INSERT on each frame in the LEX list.

The goal of the function INSERT, when it is given a frame FR, is to work out all ways that FR can modify, or be modified by, the frames that al-

ready exist, as well as to record the existence of FR for future modifications (after more words have been processed). For the latter purpose, INSERT simply puts FR on the RESULTS list of its right boundary. For the former purpose, INSERT does the following. For each frame FR1 in the RESULTS list of the left boundary of FR, INSERT calls

(MODIFY FR FR1 'RIGHT)

and

(MODIFY FR1 FR 'LEFT).

Note the recursion that exists because INSERT calls MODIFY and MODIFY can call INSERT. The recursion stops because MODIFY does not call INSERT if no modifications are possible.

When PARSE has processed the last word, it looks for those VP frames that span the whole sentence, and it prints these out in an indented tree format, as will be described and illustrated in the next section.

8. A sample grammar

The syntax diagrams are shown in Figure 1, and the input to LISP is shown in Figures 2 and 3. A portion of the lexicon is given later.

Let us first look at the NP syntax. An NP frame begins with the head noun in state N1. The test-actions associated with this first state involve RAISEF, which raises features from the most recent filler (in this case, the head noun) to the matrix frame. The result is that the number of the head noun is made a feature of the NP itself. From the head noun, one can work left getting any number of adjectives (ADJC is multiple). If a determiner is selected (filling DETR) then the NP is advanced to state N2, so that no more premodifying adjectives can be picked up. Then one is ready for postmodifiers (in this case, PP's), filling the multiple slot REL. But the frame can get into state N2 and receive REL fillers, as in *tea with cream,* without being advanced there by DETR, just because of the fact that N2 follows N1.

The PP syntax is trivial, just having a preposition as head, followed by an NP.

The VP syntax is an extension of the grammar shown earlier in diagram (12). The current grammar has a fairly complete treatment of the verb system. As outlined in the section on the lexicon, the requirements of the verb auxiliaries are managed by keeping a VP register VERB1 set to the currently left-most verb. This is initialized by the state test-action attached to state S1 (see Figure 2). This is executed as soon as the HEAD verb is filled in (actually in the lexicon), setting the register VERB1 to the value of the slot HEAD (i.e., to the frame for the head verb).

Updating of VERB1 is handled by the slot-rule for AUXL:

```
(==> (FLR V AUX) (= VERB1 IT))
```

This rule is involved in a non-trivial application of the double evaluation scheme for slot-rules described in the preceding section. When

(==> test action)

is evaluated, the test will first be evaluated. In the above example, this asks whether the filler is a verb with the feature AUX. If the test gives NIL, then the function ==> returns NIL. Otherwise, ==> returns the action, unevaluated. The parser saves this form and evaluates the filler-test for the current filler auxiliary, which needs to examine VERB1 *before it gets changed.* If this test succeeds, *then* the parser evaluates the action, (= VERB1 IT), which updates VERB1 to the new filler auxiliary.

One addition appearing in the VP syntax above is the BINDER slot attached to state S8. This gets subjunctions like *that, although, if,* and *whether* at the front of the VP.

The other additions of states have to do with the auxiliaries and the subject in question sentences. State S3 has no slots attached, but is just there to hold the test-action (ADDF QUESTION), as shown in Figure 2, which is executed for preposed auxiliaries. This adds the feature QUESTION to the matrix VP. Note that the state test-action is placed on the state that the preposed auxiliary *advances* the frame to, in accordance with the rules described in Section 4. And the preposed AUXL is attached to S2 as a state-advancer so that no more AUXL's can appear to its left. The extra state S3 does not "get in the way" of other state transitions because of the pre-processing done by SYNTAX (described in Section 7).

State S4 is added in order to handle question sentences in which the head verb is the only verb and is an auxiliary, as in

Is John Happy? May I? Does he?

This is reflected in the state test-actions for S4 shown in Figure 2. The function (IS slot) tests that that slot is filled; (ISF frame feature) tests whether the given frame has the given feature; ($ register) gets the value of the register.

In our grammar, the head of a VP is just the *last* verb in the verb group, and in elliptical VP's will be treated like a main verb. In an elliptical sentence like *Could he be?* the verb *be* is the HEAD of the VP and is just a verb which happens to be marked with the feature AUX. We leave it for other (non-syntactic) rules to decide whether this VP is elliptical for something like *Could he be happy there?* or *Could he be going there?*

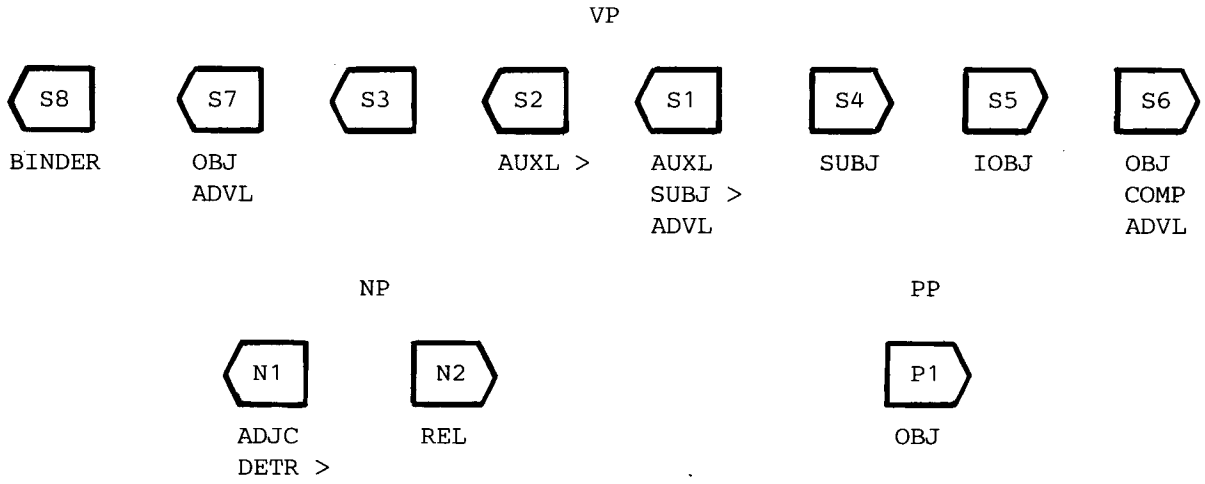


Figure 1. Syntax diagrams.

```
(SYNTAX VP
  STATES:
    (S1 L (= VERB1 (SL$ HEAD)))
    (S2 L)
    (S3 L (ADDF QUESTION))
    (S4 R
      (NOT (IS AUXL))
      (ISF ($ VERB1) AUX)
      (ADDF QUESTION) )
    (S5 R)
    (S6 R)
    (S7 L (IS SUBJ) (CLOSE))
    (S8 L)
  SLOTS:
    BINDER
      (FLR SUBJUNCTION) (S8)
    SUBJ
      (FLR NP) (S1 > S4)
    AUXL *
      (==> (FLR V AUX) (= VERB1 IT))
      (S1 S2 >)
    IOBJ
      (FLR NP) (S5)
    OBJ
      (FLR NP) (S6 S7)
    COMP
      (==>(FLR VP) (RAISE (OBJ ADVL) S7))
      (S6)
    ADVL *
      (OR (FLR ADV) (FLR PP))
      (S1 S6 S7)
  DEFAULTS:
    BINDER SUBJ AUXL ADVL )
```

Figure 2. VP syntax.

```
(SYNTAX NP
  STATES:
    (N1 L (RAISEF SG) (RAISEF PL))
    (N2 R)
  SLOTS:
    DETR
      (FLR DET) (N1 >)
    ADJC *
      (FLR ADJ) (N1)
    REL *
      (FLR PP) (N2)
  DEFAULTS:
    ADJC DETR REL )

(SYNTAX PP
  STATES:
    (P1 R)
  SLOTS:
    OBJ
      (FLR PP) (P1)
  DEFAULTS:
    OBJ )
```

Figure 3. NP and PP syntax.

The slot-rule for COMP in Figure 2 contains a call to the RAISE function:

```
(RAISE (OBJ ADVL) S7).
```

The first argument to RAISE is the list of slot types to be raised. Any slot in the filler's ASLOTS will be raised if it is actually OBJ or ADVL *or* if it originally came from one of these slots (by previous raisings). Each raised slot is given a new and unique name, but a record is kept of where it came from. It is given the same slot-rule and multiple property as the slot it was just raised from. The remaining arguments to RAISE form a state-attachments list, showing where the raised slots are to be attached.

The state S7 to which slots of type OBJ and ADVL (raised or not) are attached is the position for fronted items. As an example, the parser gives two analyses for

When did Mary say John had left?

according as *when* modifies *say* or *left*. In the first case, *when* just fills the ADVL slot in the top VP. In the second, it fills a raised slot in the top VP which was raised by COMP from the ADVL in the embedded VP.

We do not want to raise out of just any VP. It appears that we should not raise out of VP's with fronting. Compare

What do you think that those cost in France?

*What do you think that in France those cost?

This is prevented in the grammar of Figure 2 by the state action (CLOSE) attached to state S7 (the position for fronted items), which sets a flag that RAISE recognizes. When RAISE sees a CLOSEd filler frame, it just returns T and does not raise anything. This would happen in the second example above, where *in France* fills ADVL at S7 in the embedded VP and closes it. VP's with fronting *can* be accepted as fillers, as in

I think that in France those cost quite a bit.

I think that this vacation we'll enjoy a lot.

However, it is probably not right to block raising solely by internal properties of the filler VP. In a relative clause like *whom John saw,* raising would certainly be blocked, as above, by the fronting. But in the relative clause *who saw John, who* just fills the SUBJ slot, so that no closing is done. Even more clearly, in the relative clause *John saw* in *Fred is the man John saw,* there is not even a relative pronoun.

The simple answer here is that some *slots* call RAISE and others do not. Our slot COMP calls RAISE; but REL, the noun postmodifier (which would get relative clauses in an extended grammar), just does not call RAISE.

The nature of the lexicon for the sample grammar should be fairly clear from the discussions in Section 6 and the present section. Figure 4 shows part of the trial lexicon, with a sample for each part of speech. Enough samples are included to cover the types of words appearing in an example parse given below.

The function NM ("noun morphology") is similar to VM. The function TEST causes its argument to be the filler-test in all the word frames constructed for the lexical entry. Note that the word A has such a test (for number agreement in the NP), but THE does not. The verbs THINK, GIVE, and SEEM illustrate different SD lists. The SD form for SEEM causes the default slot-rule for COMP to be replaced with (FLR ADJ), so that sentences like *John seems happy* are accepted.

```
(LEXICON
  (JOHN N SG (SD))
  (HE N PRON SG (SD))
  (CHAIR N (NM CHAIRS))
  (WHAT N WH (SD))
  (LARGE ADJ)
  (THE DET)
  (A DET (TEST (NEGF FRAME PL)))
  (WHICH DET WH)
  (THAT DET (TEST (NEGF FRAME PL)))
  (THAT SUBJUNCTION)
  (IN PREP (SD))
  (ALMOST ADV)
  (THINK V (VM THINKS THINKING THOUGHT)
           (SD (COMP)) )
  (GIVE V  (VM GIVES GIVING GAVE GIVEN)
           (SD (IOBJ) (OBJ)) )
  (SEEM V  (VM SEEMS SEEMING SEEMED)
           (SD (COMP (FLR ADJ))) )
  (HAVE V AUX (VM HAS HAVING HAD)
    (SD (OBJ))
    (TEST (AND
      (ISF ($ VERB1) EN)
      (NEGF FRAME DO-AUX MODAL)
      (ADDF PERF) )) )
  (DO V AUX (VM DOES DOING DID DONE)
    (SD (OBJ))
    (TEST (AND
      (NEGF ($ VERB1) SG ING EN ED)
      (NEGF FRAME MODAL PERF PROG PASS)
      (ADDF DO-AUX) )) )   )
```

Figure 4. Sample from the lexicon.

The most complicated entries are for verbs that can be auxiliaries. Examples for HAVE and DO are shown. These entries include the main verb use as well as the auxiliary verb use. The SD form is pertinent for the former, and the TEST for the latter. For example, the filler-test for the auxiliary HAVE requires that the next verb to the right (VERB1) be a past participle.

Figure 5 shows a sample parse tree, for the sentence *Which chair did Mary think John said he almost bought?* In the tree, subordination is shown by indentation. The root node for each frame is labeled by its category and features. For lexical frames, the one daughter of that node is the word itself. For phrase frames, the daughters are basically of the form

slot
  filler

and these are given in order of actual occurrence in the sentence. If a slot is a raised slot, for example the first slot G0019 for *which chair,* then its

```
VP DO-AUX QUESTION
  G0019 (OBJ COMP COMP)
    NP SG
      DETR
        DET WH
          WHICH
      HEAD
        N SG
          CHAIR
  AUXL
    V AUX ED
      DID
  SUBJ
    NP SG
      HEAD
        N SG
          MARY
  HEAD
    V
      THINK
  COMP
    VP
      SUBJ
        NP SG
          HEAD
            N SG
              JOHN
      HEAD
        V ED EN
          SAID
      COMP
        VP
          SUBJ
            NP SG
              HEAD
                N PRON SG
                  HE
          ADVL
            ADV
              ALMOST
          HEAD
            V ED EN
              BOUGHT
```

Figure 5. Parse tree for the sentence, "Which chair did Mary think John said he almost bought?"

"origin" is shown beside it. The origin (OBJ COMP COMP) means that the original slot from which it came was OBJ, and the path to it is through two COMP's. This means that the slot G0019 came from the third-level embedded VP *he almost bought,* so *which chair* is the object of *bought.*

Six additional examples, of varying complexity, are given in the Appendix to this paper which is included in the microfiche supplement.

9. Summary

We have offered a grammatical system and parser organized around slots and slot-filling, with a constrained use of states. The parser is driven by the maintenance of the *available slots* list, ASLOTS, consisting of those slots that may *yet* be filled. Two advantages of this were emphasized. One is that ASLOTS permits the expression of dependency relations in a natural and direct way. The other is that ASLOTS serves as the vehicle for the *raising* operation, which appears to be applicable to several grammatical constructions, such as WH-movement.

The parser is bottom-up and phrases are constructed *middle-out* from their head words. This scheme is instrumental for both of the above advantages of ASLOTS. First, the dependency information associated with head words in the lexicon helps initialize ASLOTS appropriately. Second, middle-out construction is appropriate because raised slots might be filled on the left or the right.

The system seems to represent a good combination of data-directed and goal-directed processing. The actual lexical data in the sentence not only influence the initialization of ASLOTS lists, but also control whatever agreement checks may be necessary (such as subject-verb agreement and morphological requirements of auxiliaries). Once the ASLOTS list of a phrase frame is determined, it forms a direct and central expression of goals for filling out the frame.

References

Cook, W. A. (1969). *Introduction to Tagmemic Analysis.* Holt, Rinehart and Winston, New York.

Heidorn, G. E. (1972). *Natural Language Inputs to a Simulation Programming System.* Technical Report NPS-55HD72101A, Naval Postgraduate School, Monterey, California.

Heidorn, G. E. (1975). Augmented phrase structure grammars. In *Theoretical Issues in Natural Language Processing,* B. L. Nash-Webber and R. C. Schank (Eds.), pp. 2-5, Association for Computational Linguistics.

Hudson, R. A. (1971). *English Complex Sentences.* North-Holland, Amsterdam.

Hudson, R. A. (1976). *Arguments for a Non-transformational Grammar.* University of Chicago Press, Chicago.

Hudson, R. A. (1977). The power of morphological rules. *Lingua, 42,* 73-89.

Kac, M. B. (1978). *Corepresentation of Grammatical Structure.* University of Minnesota Press, Minneapolis.

McCord, M. C. (1975). On the form of a systemic grammar. *Journal of Linguistics, 11,* 195-212.

McCord, M. C. (1977). Procedural systemic grammars. *International Journal of Man-Machine Studies, 9,* 255-286.

Minsky, M. (1975). A framework for representing knowledge. In *The Psychology of Computer Vision,* P. H. Winston (Ed.), pp. 211-277. McGraw-Hill, New York.

Winograd, T. (1975). Frame representations and the declarative/procedural controversy. In *Representation and Understanding,* D. G. Bobrow and A. Collins (Eds.), pp. 185-210, Academic Press, New York.

Woods, W. A. (1970). Transition network grammars for natural language analysis. *CACM, 13,* 591-606.

Woods, W. A. (1973). An experimental parsing system for transition networks. In *Natural Language Processing,* R. Rustin (Ed.), pp. 111-154, Algorithmics Press, New York.