Transition Network Grammars for Natural Language Analysis

W. A. Woods
Harvard University, Cambridge, Massachusetts

The use of augmented transition network grammars for the analysis of natural language sentences is described. Structure-building actions associated with the arcs of the grammar network allow for the reordering, restructuring, and copying of constituents necessary to produce deep-structure representations of the type normally obtained from a transformational analysis, and conditions on the arcs allow for a powerful selectivity which can rule out meaningless analyses and take advantage of semantic information to guide the parsing. The advantages of this model for natural language analysis are discussed in detail and illustrated by examples. An implementation of an experimental parsing system for transition network grammars is briefly described.

KEY WORDS AND PHRASES: computational linguistics, grammars, grammar models, linguistics, natural language analysis, parsing, semantic interpretation, transition network grammars, transformational grammars
CR CATEGORIES: 3.42, 4.12

1. Motivation

One of the early models for natural language grammars was the finite state transition graph. This model consists of a network of nodes and directed arcs connecting them, where the nodes correspond to states in a finite state machine and the arcs represent transitions from state to state. Each arc is labeled with a symbol whose input can cause a transition from the state at the tail of the arc to the state at its head. This model has the attractive feature that the sequences of words which make up a sentence can be read off directly by following the paths through the grammar from the initial state to some final state. Unfortunately, the model is grossly inadequate for the representation of natural language grammars due to its failure to capture many of their regularities. A most notable inadequacy is the absence of a pushdown mechanism that permits one to suspend the processing of a constituent at a given level while using the same grammar to process an embedded constituent.

Suppose, however, that one added the mechanism of recursion directly to the transition graph model by fiat. That is, suppose one took a collection of transition graphs each with a name, and permitted as labels on the arcs not only terminal symbols but also nonterminal symbols naming complex constructions which must be present in order for the transition to be followed. The determination of whether such a construction was in fact present in a sentence would be done by a "subroutine call" to another transition graph (or the same one). The resulting model of grammar, which we will call a *recursive transition network*, is equivalent in generative power to that of a context-free grammar or pushdown store automaton, but as we will show, allows for greater efficiency of expression, more efficient parsing algorithms, and natural extension by "augmentation" to more powerful models which allow various degrees of context dependence and more flexible structure-building during parsing. We argue in fact that an "augmented" recursive transition network is capable of performing the equivalent of transformational recognition (cf. Chomsky [6, 7]) without the necessity of a separate inverse transformational component, and that this parsing can be done in an amount of time which is comparable to that of predictive context-free recognition.

The research reported here was supported in part by NSF grant GS-2301.

2. Recursive Transition Networks

A *recursive transition network* is a directed graph with labeled states and arcs, a distinguished state called the start state, and a distinguished set of states called final states. It looks essentially like a nondeterministic finite state transition diagram except that the labels on the arcs may be state names as well as terminal symbols. The interpretation of an arc with a state name as its label is that the state at the end of the arc will be saved on a pushdown store and the control will jump (without advancing the input tape) to the state that is the arc label. When a final state is encountered, then the pushdown store may be "popped" by transferring control to the state which is named on the top of the stack and removing that entry from the stack. An attempt to pop an empty stack when the last input character has just been processed is the criterion for acceptance of an input string. The state names that can appear on arcs in this model are essentially the names of constructions that may be found as "phrases" of the input tape. The effect of a state-labeled arc is that the transition that it represents may take place if a construction of the indicated type is found as a "phrase" at the appropriate point in the input string.

Figure 1 gives an example of a recursive transition network for a small subset of English. It accepts such sentences as "John washed the car" and "Did the red barn collapse?" It is easy to visualize the range of acceptable sentences from inspection of the transition network. To recognize the sentence "Did the red barn collapse?" the network is started in state S. The first transition is the aux transition

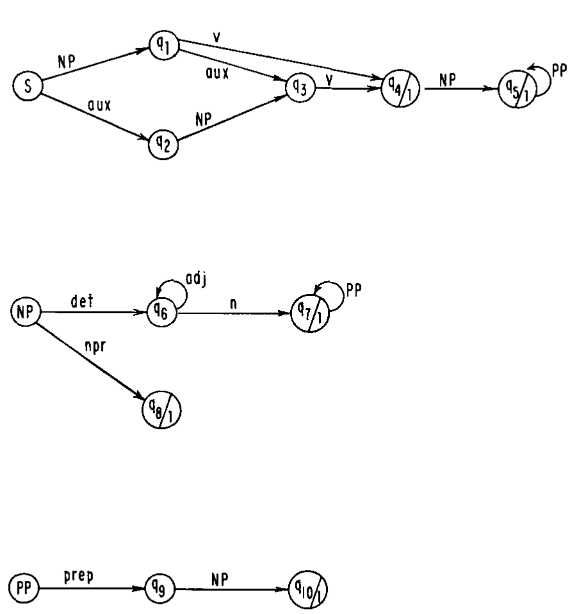


FIG. 1. A sample transition network. S is the start state. q_4, q_5, q_7, q_8, and q_{10} are the final states.

to state q_2 permitted by the auxilliary "did". From state q_2 we see that we can get to state q_3 if the next "thing" in the input string is an NP. To ascertain if this is the case, we call the state NP. From state NP we can follow the arc labeled det to state q_6 because of the determiner "the". From here, the adjective "red" causes a loop which returns to state q_6, and the subsequent noun "barn" causes a transition to state q_7. Since state q_7 is a final state, it is possible to "pop up" from the NP computation and continue the computation of the top level S beginning in state q_3 which is at the end of the NP arc. From q_3 the verb "collapse" permits a transition to the state q_4, and since this state is final and "collapse" is the last word in the string, the string is accepted as a sentence.

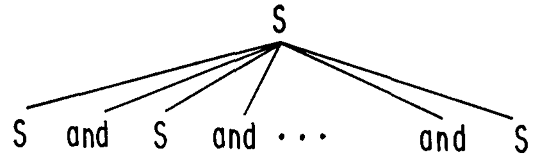
In the above example, there is only one accepting path through the network—i.e. the sentence is unambiguous with respect to the grammar. It is an inherent feature of natural language, however, that except for contrived subsets of the language there will be ambiguous sentences which have several distinct analysis paths through the transition network. The transition network model therefore is fundamentally a nondeterministic mechanism, and any parsing algorithm for transition network grammars must be capable of following any and all analysis paths for any given sentence.

The fact that the recursive transition network is equivalent to a pushdown store automaton is not difficult to establish. Every recursive transition network is essentially a pushdown store automaton whose stack vocabulary is a subset of its state set. The converse fact that every pushdown store automaton has an equivalent transition net could be established directly, but can be more simply established by noting that every pushdown store automaton has an equivalent context-free grammar which has an equivalent recursive transition net.

3. Augmented Transition Networks

It is well known (cf. Chomsky [6]) that the strict context-free grammar model is not an adequate mechanism for characterizing the subtleties of natural languages. Many of the conditions which must be satisfied by well-formed English sentences require some degree of agreement between different parts of the sentence which may or may not be adjacent (indeed which may be separated by a theoretically unbounded number of intervening words). Context-sensitive grammars could take care of the weak generation of many of these constructions, but only at the cost of losing the linguistic significance of the "phrase structure" assigned by the grammar (cf. Postal [27]). Moreover, the unaided context-free grammar model is unable to show the systematic relationship that exists between a declarative sentence and its corresponding question form, between an active sentence and its passive, etc. Chomsky's theory of transformational grammar [7], with its distinction between the surface structure of a sentence and its deep structure, answers these objections but falls victim to inadequacies of its own (cf. Schwarcz [28] or McCawley [21]). In this section we describe a model of grammar based on the notion of a recursive transition network which is capable of performing the equivalent of transformational recognition without the need for a separate transformational component and which meets some of the objections that have been raised against the traditional model of transformational grammar.

The basic recursive transition network model as we have described it is weakly equivalent to the context-free grammar model and differs in strong equivalence only in its ability to characterize unbounded branching, as in structures of the form:



The major features which a transformational grammar adds to those of the context-free grammar are the abilities to move fragments of the sentence structure around (so that their positions in the deep structure are different from those in the surface structure), to copy and delete fragments of sentence structure, and to make its actions on constituents generally dependent on the contexts in which those constituents occur. We can add equivalent facilities to the transition network model by adding to each arc of the transition network an arbitrary condition which must be satisfied in order for the arc to be followed, and a set of structure building actions to be executed if the arc is followed. We call this version of the model an *augmented* transition network.

The augmented transition network builds up a partial structural description of the sentence as it proceeds from state to state through the network. The pieces of this partial description are held in *registers* which can contain any rooted tree or list of rooted trees and which are automatically pushed down when a recursive application of the transition network is called for and restored when the lower level (recursive) computation is completed. The structure-building actions on the arcs specify changes in the contents of these registers in terms of their previous contents, the contents of other registers, the current input symbol, and/or the result of lower level computations. In addition to holding pieces of substructure that will eventually be incorporated into a larger structure, the registers may also be used to hold flags or other indicators to be interrogated by conditions on the arcs.

Each final state of the augmented network has associated with it one or more conditions which must be satisfied in order for that state to cause a "pop". Paired with each of these conditions is a function which computes the value to be returned by the computation. A distinguished register, *, (which usually contains the current input word when a word is being scanned) is set to the result of the lower level computation when the network returns to an arc which has called for a recursive computation. Thus the register * in every case contains a representation of the "thing" (word or phrase) which caused a transition.

3.1. Representation of Augmented Networks.

To make the discussion of augmented transition networks more concrete, we give in Figure 2 a specification of a language in which an augmented transition network can be represented. The specification is given in the form of an extended context-free grammar in which a vertical bar separates alternative ways of forming a construction and the Kleene star operator (*) is used as a superscript to indicate arbitrarily repeatable constituents. The nonterminal symbols of the grammar consist of English descriptions enclosed in angle brackets, and all other symbols except the vertical bar and the superscript * are terminal symbols (including the parentheses, which indicate list structure). The * which occurs as an alternative right-hand side for the rule for the construction ⟨form⟩, however, is a terminal symbol and is not to be confused with the superscript *'s which indicate repeatable constituents. The first line of the figure says that a transition network is represented by a left parenthesis, followed by an arc set, followed by any number of arc sets (zero or more), followed by a right parenthesis. An arc set, in turn, consists of a left parenthesis, followed by a state name, followed by any number of arcs, followed by a right parenthesis, and an arc can be any one of the four forms indicated in the third rule of the grammar. The remaining rules are interpreted in a similar fashion. Nonterminals whose expansions are not given in Figure 2 have names which should be self-explanatory.

```
⟨transition network⟩ → (⟨arc set⟩⟨arc set⟩*)
⟨arc set⟩ → (⟨state⟩⟨arc⟩*)
⟨arc⟩ → (CAT ⟨category name⟩⟨test⟩⟨action⟩* ⟨term act⟩)|
        (PUSH ⟨state⟩⟨test⟩⟨action⟩* ⟨term act⟩)|
        (TST ⟨arbitrary label⟩⟨test⟩⟨action⟩* ⟨term act⟩)|
        (POP ⟨form⟩⟨test⟩)
⟨action⟩ → (SETR ⟨register⟩⟨form⟩)|
           (SENDR ⟨register⟩⟨form⟩)|
           (LIFTR ⟨register⟩⟨form⟩)
⟨term act⟩ → (TO ⟨state⟩)|
             (JUMP ⟨state⟩)
⟨form⟩ → (GETR ⟨register⟩)|
         *|
         (GETF ⟨feature⟩)|
         (BUILDQ ⟨fragment⟩⟨register⟩*)|
         (LIST ⟨form⟩*)|
         (APPEND ⟨form⟩⟨form⟩)|
         (QUOTE ⟨arbitrary structure⟩)
```

Fig. 2. Specification of a language for representing augmented transition networks

The expressions generated as transition networks by the grammar of Figure 2 are in the form of parenthesized list structures, where a list of the elements A, B, C, and D is represented by the expression (A B C D). The transition network is represented as a list of arc sets, each of which is itself a list whose first element is a state name and whose remaining elements are arcs leaving that state. The arcs also are represented as lists, possible forms of which are indicated in the figure. (The conditions and functions associated with final states are represented as (pseudo) "arcs" with no actions and no destination.) The first element of each arc is a word which names the type of the arc, and the third element is an arbitrary test which must be satisfied in order for the arc to be followed. The CAT arc is an arc which can be followed if the current input symbol is a member of the lexical category named on the arc (and if the test is satisfied), while the PUSH arc is an arc which causes a pushdown to the state indicated. The TST arc is an arc which permits an arbitrary test to determine whether an arc is to be followed. In all three of these arcs, the actions on the arc are the structure-building actions, and the terminal action specifies the state to which control is passed as a result of the transition. The two possible terminal actions, TO and JUMP, indicate whether the input pointer is to be advanced or not advanced, respectively—that is, whether the next state is to scan the next input word or whether it is to continue to scan the same word. The POP arc is a dummy arc which indicates under what conditions the state is to be considered a final state, and the form to be returned as the value of the computation if the POP alternative is chosen. (One advantage of representing this information as a dummy arc is the ability to order the choice of popping with respect to the other arcs which leave the state.)

The actions and the *forms* which occur in the network are represented in "Cambridge Polish" notation, a notation in which a function call is represented as a parenthesized list whose first element is the name of the function and whose remaining elements are its arguments. The three actions indicated in Figure 2 cause the contents of the indicated register to be set equal to the value of the indicated *form*. SETR causes this to be done at the current level of computation in the network, while SENDR causes it to be done at the next lower level of embedding (used to send information down to a lower level computation) and LIFTR causes it to be done at the next higher level computation (used to return additional information to higher level computations).

The *forms* as well as the conditions (*tests*) of the transition network may be arbitrary functions of the register contents, represented in some functional specification language such as LISP (McCarthy et al. [20]), a list processing programming language based on Church's lambda calculus and written in Cambridge Polish notation. The seven types of forms listed in Figure 2 are a basic set which is sufficient to illustrate the major features of the augmented transition network model. GETR is a function whose value is the contents of the indicated register, * is a form whose value is usually the current input word, and GETF is a function which determines the value of a specified feature for the current input word. (In the actions which occur on a PUSH arc, * has the value of the lower level computation which permitted the PUSH transition.)

BUILDQ is a useful structure-building form which takes a list structure representing a fragment of a parse tree with specially marked nodes and returns as its value the result of replacing those specially marked nodes with the contents of indicated registers.[1] Specifically, for each occurrence of the symbol + in the list structure given as its first argument, BUILDQ substitutes the contents of one of the listed registers (the first register replacing the first + sign, the second register the second +, etc.). In addition, BUILDQ replaces occurrences of the symbol * in the fragment with the value of the form *.

The remaining three forms are basic structure-building forms (out of which any BUILDQ can be duplicated) which respectively make a list of the values of the listed arguments, append two lists together to make a single list, and produce as value the (unevaluated) argument form. An illustrative fragment of an augmented transition network is given in Figure 3. In Section 3.2 the operation of this network is described and some of the features of the augmented transition network model are discussed.

[1] The BUILDQ function which is implemented in the experimental parsing system (See Section 10) is considerably more versatile than the version described here. Likewise, the implemented parser contains additional formats for arcs as well as other extensions to the language specified here. There has been no attempt to define a basic irredundant set of primitive conditions, actions, and forms, but rather an effort has been made to allow flexibility for adding "natural" primitives which facilitate the writing of compact grammars. For this reason, the set of possible conditions, actions, and forms has been left open-ended to allow for experimental determination of useful primitives. However, the arc formats and actions described here, together with arbitrary LISP expressions for conditions and *forms*, provides a model which is equivalent in power to a Turing machine and therefore complete in a theoretical sense.

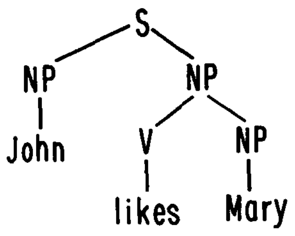
3.2. An Illustrative Example. Figure 3 gives a fragment of an augmented transition network represented in the language of Figure 2. This fragment is an augmentation of the portion of the transition network of Figure 1 which consists of the states S/, Q1, Q2, Q3, Q4, and Q5. The augmented network builds a structural representation in which the first constituent of a sentence is a *type* (either DCL or Q) which indicates whether the sentence is declarative or interrogative, the second constituent is the subject noun phrase, the third is an auxilliary (or NIL if there is no auxilliary), and the fourth is the verb phrase constituent. This representation is produced regardless of the order in which the subject noun phrase and the auxilliary occur in the sentence. The network also produces a representation of a verb phrase constituent even though there is no pushdown in the network corresponding to a verb phrase. It will be helpful, both for the understanding of the notation and for the understanding of the operation of the augmented network, to follow through an example at this point using the network fragment of Figure 3.

```
(S/ (PUSH NP/ T
        (SETR SUBJ *)
        (SETR TYPE (QUOTE DCL))
        (TO Q1))
    (CAT AUX T
        (SETR AUX *)
        (SETR TYPE (QUOTE Q))
        (TO Q2)))
(Q1 (CAT V T
        (SETR AUX NIL)
        (SETR V *)
        (TO Q4))
    (CAT AUX T
        (SETR AUX *)
        (TO Q3)))
(Q2 (PUSH NP/ T
        (SETR SUBJ *)
        (TO Q3)))
(Q3 (CAT V T
        (SETR V *)
        (TO Q4)))
(Q4 (POP (BUILDQ (S+++(VP+)) TYPE SUBJ AUX V) T)
    (PUSH NP/ T
        (SETR VP (BUILDQ (VP (V+) *) V))
        (TO Q5)))
(Q5 (POP (BUILDQ (S++++) TYPE SUBJ AUX VP) T)
    (PUSH PP/ T
        (SETR VP (APPEND (GETR VP) (LIST *)))
        (TO Q5)))
```

Fig. 3. An illustrative fragment of an augmented transition network

Before proceeding to work an example, however, it is necessary to explain the representation of the parse trees which is used by the network fragment. The parse trees are represented in a parenthesized notation in which the representation of a node consists of a list whose first element is the name of the node and whose remaining elements are the representations of the constituents of that node.

For example, the parse tree



would be represented in this notation by the expression:

(S (NP John) (VP (V likes) (NP Mary))

This representation can also be viewed as a labeled bracketing of the sentence in which a left bracket for a phrase of type X is represented by a left parenthesis followed by an X, and the matching right bracket is simply a right parenthesis.

Let us now consider the operation of the augmented network fragment of Figure 3 for the input sentence "Does John like Mary?"

1. We begin the process in state S/ scanning the first word of the sentence, "does". Since this word is an auxilliary, its dictionary entry would mark it as a member of the category AUX and therefore (since its arbitrary condition T is the universally true condition) the arc (CAT AUX T · · ·) can be followed. (The other arc which pushes down to look for a noun phrase will not be successful.) In following this arc, we execute the actions: (SETR AUX *), which puts the current word "does" into a register named AUX, (SETR TYPE (QUOTE Q)), which puts the symbol "Q" into a register named TYPE, and (TO Q2), which causes the network to enter state Q2 scanning the next word of the sentence "John".

2. State Q2 has only one arc leaving it, which is a push to state NP/. The push will be successful and will return a representation of the structure of the noun phrase which will then become the value of the special register *. We will assume that the representation returned is the expression "(NP John)". Now, having recognized a construction of type NP, we proceed to perform the actions on the arc. The action (SETR SUBJ *) causes the value "(NP John)" to be placed in the register SUBJ, and the action (TO Q3) causes us to enter the state Q3 scanning the next word "like". The register contents at this point are:

TYPE:	Q
AUX:	does
SUBJ:	(NP John)

3. From state Q3, the verb "like" allows a transition to state Q4, setting the contents of a register V to the value "like" in the process, and the input pointer is advanced to scan the word "Mary".

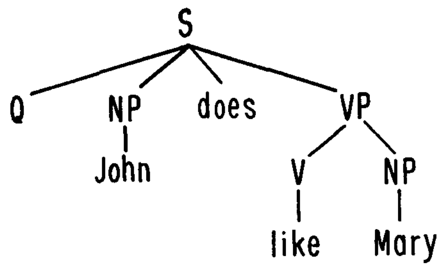
4. Q4, being a final state could choose to "POP", indicating that the string that has been processed so far is a complete sentence (according to the grammar of Figure 1); however, since this is not the end of the sentence, this alternative is not successful. However, the state also has an arc which pushes down to state NP/, and this alternative will succeed, returning the value "(NP Mary)". The action (SETR VP (BUILDQ (VP (V +) *) V)) will now take the structure fragment "(VP (V +) *)" and substitute the current value of * for the occurrence of * in the fragment and replace the occurrence of + with the contents of the indicated register V. The resulting structure, "(VP (V like) (NP Mary))" will be placed in the register VP, and the action (TO Q5) causes a transition to state Q5 scanning beyond the end of the input string. The register contents at this point are:

TYPE:	Q
AUX:	does
SUBJ:	(NP John)
V:	like
VP:	(VP (V like) (NP Mary)).

5. We are now scanning the end of the sentence, and since Q5 is a final state (i.e. it has a "POP" arc) and the condition T is satisfied, the sentence is accepted. The form "(BUILDQ (S + + + +) TYPE SUBJ AUX VP)" specifies the value to be returned as the analysis of the sentence. The value is obtained by substituting the contents of the registers TYPE, SUBJ, AUX, and VP for the successive instances of the symbol "+" in the fragment "(S + + + +)" to give the final sentence analysis

(S Q (NP John) does (VP (V like) (NP Mary))),

which represents the parse tree:



In ordinary context-free recognition, the structural descriptions of sentences are more or less direct representations of the flow of control of the parser as it analyzes the sentence. The structural descriptions assigned by the structure building rules of an augmented transition network, as we can see from the example, are comparatively independent of the flow of control of the algorithm. This is not to say that they are not determined by the flow of control of the parser, for this they surely are; rather we mean to point out that they are not isomorphic to the flow of control as in the usual context-free recognition algorithms. It is possible for a constituent that is found in the course of analysis to appear in the final structural description several times or not at all, and its location may be entirely different from that in which it was found in the surface structure. In addition, the structural description assigned to a constituent at one point during the analysis may be changed or transformed before that structure is incorporated into the final structural description of the sentence as a whole. These facilities, plus the ability to test arbitrary conditions, allow the equivalent of a transformational deep structure to be constructed while the parser is performing transitions that are isomorphic to the surface structure of a sentence.

4. Transformational Recognition

The usual model of transformational grammar is a generative model consisting of a context-free (*base*) grammar and a set of transformational rules which map syntax trees into new (*derived*) syntax trees. The generation of a sentence with such a grammar consists of first constructing a *deep structure* using the base component grammar and then transforming this deep structure into a *surface structure* by successive applications of transformations. The terminal nodes (or leaves) of the surface structure tree give the final form of the sentence. This model of transformational grammar is totally oriented toward the generation of sentences rather than their analysis, and although there is clearly an algorithm for the use of such a grammar to analyze a sentence—namely the procedure of "analysis by synthesis" (Matthews [23])—this algorithm is so inefficient as to be out of the question for any practical application. (The analysis by synthesis method consists of applying the rules in the "forward" (generative) direction in all possible ways to generate all of the possible sentences of the language while looking to see if the sentence which you are trying to analyze turns up in the list.)

Two attempts to formulate more practical algorithms for transformational recognition (Petrick [26] and Mitre [24]) resulted in algorithms which were either too time consuming for the analysis of large numbers of sentences or else lacking in formal completeness. Both of these algorithms attempt to analyze sentences by applying the transformations in reverse, a procedure which is far less straightforward than it sounds. The difficulty with simply performing the transformations in reverse is twofold. First, the transformations operate on tree structures and produce tree structures as their values. In the forward direction, they begin with the deep structure tree and end with the surface structure tree. To reverse this process, it is first necessary to obtain a surface structure tree for the input sentence. However, there is no component in the transformational model which characterizes the possible surface structures (their only characterization is implicit in the changes which can be made in the deep structures by means of the transformations). Both the Mitre and the Petrick analysis procedures solve this problem by constructing an "augmented grammar" which consists of the rules of the original base component grammar plus additional rules which characterize the structures that can be added by transformations. In the Mitre procedure, this "surface grammar" is constructed by hand and no formal procedure is available for constructing it from the original transformational grammar. In the Petrick procedure, there is a formal procedure for obtaining an augmented grammar but it will not necessarily terminate unless the length of the possible input sentences is first circumscribed. When sentences longer than the chosen length are encountered, more augmented grammar rules must be generated.

In the Mitre procedure, the augmented grammar is used to assign a complete "tentative" surface structure which is then subjected to inverse transformations. In the Petrick procedure, inverse transformations are applied to partially built up surface structures and the processes of applying transformations and building structure are interwoven. In both systems, the inverse transformations may or may not produce a legitimate deep structure. If they do, the sentence is accepted, but if they do not, the tentative surface structure was spurious and is rejected. There is no way to construct a context-free surface grammar which will assign all and only legitimate surface structures. One must settle for one which will assign all legitimate surface structures plus additional spurious ones. Moreover, the only way to tell the two apart is to perform the inverse transformations and check the resulting "tentative" deep structures.

A second difficulty with the Petrick algorithm is the combinatorial explosion of the number of possible inverse transformation sequences that can be applied to a given surface structure tree. Although many of the transformations when applied in the forward direction are obligatory so that only one possible action can be taken, almost all of the inverse transformations are optional. The reason for this is that even though a given structure looks like it could have been produced by a given forward transformation so that the inverse transformation can be performed, there is no guarantee that the same structure could not have arisen in a transformational derivation in some other way. Therefore both the alternative of applying the inverse transformation and that of not applying it must be tried whenever an inverse transformation can apply. The number of active paths can grow exponentially with the number of transformations applied. Moreover, the forward transformations usually do not specify much information about the structure which results from applying the transformation (even though the linguist may know a good deal about what the resulting structure must be like). For this reason, the inverse transformations are not as selective as their forward counterparts and many more spurious applications of transformations are allowed. That is, whereas most forward sequences of transformations will lead to successful surface structures, most inverse sequences will not lead to legitimate deep structures, and a large amount of wasted effort is therefore expended on dead-end paths. The Mitre parser avoids the nondeterminism of the inverse transformational process by constructing a deterministic set of inverse transformational rules ad hoc to a particular grammar. This method, however, is not guaranteed to produce all legitimate deep structures of a sentence, and there is no formal procedure for constructing the necessary set of inverse transformations.

5. Augmented Transition Networks for Transformational Recognition

In 1965 Kuno [18] suggested that it should be possible to augment the surface structure grammar of a transformational grammar in such a way that it "remembered" the equivalent deep structure constructions and could build the deep structure of the sentence while doing the surface structure parsing—without the necessity of a separate inverse transformational component. The model which he proposed at that time, however, was not adequate to deal with some of the more powerful transformational mechanisms such as the extraposition of a constituent from an arbitrarily deep embedding. The augmented transition network, on the other hand, provides a model which is capable of doing everything that a transformational grammar can do and is therefore a realization of part of the Kuno prediction. It remains to be seen whether a completely mechanical procedure can be developed to take a transformational grammar in the usual formalism and translate it into an equivalent augmented transition network. I conjecture, however, that such is the case.

Even if such a mechanical procedure is available, it may still be more appropriate to use the transition network model directly for the original linguistic research and grammar development. The reasons for this are several. First, the transition network that could be developed by a mechanical procedure from a traditional transformational grammar could not be expected to be as efficient as that which could be designed by hand. Moreover, the transition network model provides a mechanism which satisfies some of the objections which have been raised by linguists against the transformational grammar as a linguistic model (such as its incompatibility with many psycholinguistic facts which we know to characterize human language performance).

A third reason for preferring the transition network model to the usual formulation of transformational grammar is the power which it contains in its arbitrary conditions and its structure building actions. The model is equivalent to a Turing machine in power, and yet the actions which it performs are "natural" ones for the analysis of language. Most linguistic research in the structure of language and mechanisms of grammar has attempted deliberately to build models which do not have the power of a Turing machine but which make the strongest possible hypotheses about language mechanisms by proposing the least powerful mechanism that can do the job. As a result of this approach, many variations of the transformational grammar model have been proposed with different basic repertories of transformational mechanisms. Some have cyclic transformation rules, others do not; some have a distinct "post cycle" that operates after all of the cyclic rules have been applied. There are various types of conditions that may be asked: some models have double structural descriptions, some have ordered rules, some have obligatory rules, some have blocking rules, etc. In short, there is not a single transformational grammar model, there are many models which are more or less incomparable. If one such model can handle some features of language and another can handle different features, there is no systematic procedure for incorporating them both into a single model. In the augmented transition network model, the possibility exists of adding to the model whatever facility is needed and seems natural to do the job. One can add a new mechanism by simply inventing a new basic predicate to use in conditions or a new function to use in the structure-building rules. It is still possible to make strong hypotheses about the types of conditions and actions that are required, but when one finds that he needs to accomplish a given task for which his basic model has no "natural" mechanism, there is no problem extending the augmented transition network model to include it. This requires only the relaxation of the restrictions on the types of conditions and actions, and no reformulation of the basic model.

6. Previous Transition Network Models

The basic idea of the recursive transition network—that of merging the right-hand sides of context-free grammar rules which have the same left-hand side into a single transition diagram that merges the common parts of the different rules—has been known to the designers of syntax-directed compilers and artificial programming languages at least since 1963 when it was described in a paper by Melvin Conway [8]. The concern of that time, however, was not with the full generality of the nondeterministic mechanism, but rather with a set of sufficient conditions that would guarantee the diagram to be deterministic. Conway describes a rudimentary form of action associated with the arcs of his transition diagram, but these actions are limited to output commands which write information into a separate output stream that serves as input to the code-generation component. (The model is very close to the usual model of a finite state transducer with the exception of the additional recursion capability.) There is no analog to the holding of temporary pieces of information in registers, or the subsequent use of such information in conditions on the arcs.

More recently, two natural language parsing systems based on a form of recursive transition network have been described in the literature. Thorne, Bratley, and Dewar [29] describe a procedure for natural language analysis based on a "finite state transition network" (which is applied recursively), and Bobrow and Fraser [1] describe a system which is "an elaboration of the procedure described by Thorne, Bratley, and Dewar." Although these systems bear considerable similarity to the one we have described, they differ from it in a number of important respects which we will describe shortly. Let us first, however, briefly describe the two systems.

6.1. The Thorne System. The Thorne system [29] assigns a representation of syntactic structure which attempts to represent simultaneously the deep structure and the surface structure of a sentence. Constructions are listed in the order in which they are found in the surface structure, with their deep structure functions indicated by labeling. Inversions in word order are indicated by marking the structures which are found "out of place" (i.e. in positions other than their deep structure positions) without moving them from their surface structure positions, and later in the string the position where they would have occurred in the deep structure is indicated by the appropriate deep structure function label followed by an asterisk. (They do not describe a procedure for constituents which are found in the surface structure to the right of their deep structure positions. Apparently their grammar does not deal with such constructions.)

Thorne views his grammar as a form of transformational grammar whose base component is a finite state grammar and permits recursion to take place only via transformations. According to Thorne, the majority of transformation rules can be viewed as "meta rules" in the sense that "they operate on other rules to produce derived rules rather than operating on structural descriptions to produce new structural descriptions." He uses an augmented transition network containing both the original deep structure rules plus these derived rules as the grammar table to drive his parsing algorithm, but is not able to handle the word order inversion transformations and the conjunction transformations in this way. Instead, he implements these features as exceptions embedded in his parsing program.

6.2. The System of Bobrow and Fraser. Bobrow and Fraser [1] describe a parsing system which is an elaboration of the Thorne parser. Like the Thorne parsings, the general form of their analysis "resembles the surface structure analysis of the sentence, with added indications of moved constituents and where they are located in deep structure." This grammar model is also a form of augmented transition network, whose actions include setting flags and function labels and whose conditions include testing previously set flags. Unlike the Thorne system, however, Bobrow's system provides a facility for transferring information back to some previously analyzed constituent. In general, the conditions on an arc can be arbitrary LISP functions (the system is programmed in LISP), and the actions for transferring information can be arbitrary LISP functions. The conditions and actions actually implemented in the system, however, are limited to flag testing and to transferring new deep structure function labels back into previously recognized structures.

According to Bobrow[2] the major differences between his system and that of Thorne are the use of symbolic flag names (instead of bit positions), a facility for mnemonic state names, the ability to transfer information back to previously analyzed constituents, and a facility for active feature values in the dictionary (these are actually routines which are stored in the dictionary entry for the word rather than merely activated by features stored in the dictionary).

6.3. Comparison with the Present Model. In comparing the augmented transition network model described in this paper with the systems of Bobrow and Fraser [1] and of Thorne et al. [29], there are two domains of comparison which must be distinguished: the formal description of the model and the implementation of the parsing system. One of the major differences between this parsing system and those of Bobrow and Thorne is the degree to which such a distinction is made. In [29] Thorne does not describe the augmented transition network model which is used except to point out that the grammar table used by the parsing program "has the form of a finite state network or directed graph—a form appropriate for the representation of a regular grammar." The transition network model is apparently formalized only in the form in which it actually occurs in the parsing program (which is not described). The conditions on the arcs seem to be limited to tests of agreement of features associated with lexical items and constituents, and the actions are limited to recording the current constituent in the output representation, labeling constituents, or inserting dummy nodes and markers. The mechanisms for word order inversion and conjunction are not represented in the network but are "incorporated into the program."

The Bobrow and Fraser paper [1] improves considerably on the power of the basic transition network model used by Thorne et al. It adds the facility for arbitrary conditions and actions on the arcs, thus increasing the power of the model to that of a Turing machine. In this system as in Thorne's, however, there is no distinction between the model and the implementation. Although the conditions and actions are arbitrary as far as the implementation is concerned, there is no separate formal model which characterizes the data structures on which they operate. That is, in order to add such an arbitrary condition, one would have to know how the LISP implementation of the parsing algorithm works and where and how its intermediate results are stored. The range of conditions and actions available without such information—i.e. the condition and action subroutines actually provided in the implementation—consists of setting and testing flags and transmitting func-

[2] Personal communication.

tion labels back into previously analyzed constituents. In both Bobrow's and Thorne's systems the actual representation of constituent structure is isomorphic to the recursive structure of the analysis as determined by the history of recursive applications of the transition network, and it is produced automatically by the parsing algorithm.

The augmented transition network, as we have defined it, provides a formalized transition network model with the power of a Turing machine *independent of the implementation*. The model explicitly provides for the isolation of various partial results in named registers and allows arbitrary conditions and actions *which apply to the contents of these registers*. Thus it is not necessary for a grammar writer to know details of the actual implementation of the parsing algorithm in order to take advantage of the facility for arbitrary conditions and actions.[3] The building of the constituent structure is not performed automatically by the parsing algorithm in this model, but must instead be specified by explicit structure-building rules. The result of this feature is that the structures assigned to the sentence no longer need to be isomorphic to the recursion history of the analysis, but are free to move constituents around in the representation. Thus the representation produced by the parser may be a true deep structure representation of the type assigned by the more customary transformational grammar models (or it could also be a surface structure representation, a dual-purpose representation as in the Thorne and Bobrow systems, or any of a number of other representations such as dependency representations). The explicit structure-building actions on the arcs together with the use of registers to hold pieces of sentence structure (whose function and location may not yet have been determined) provides an extremely flexible and efficient facility for moving constituents around in their deep structure representations and changing the interpretation of constituents as the course of an analysis proceeds. It is even possible to build structures with several levels of nesting while remaining at a single level of the transition network, and conversely to go through several levels of recursion of the network while building a structure which has only one level. No facility like this is present in either the Thorne or the Bobrow systems.

Another feature of the augmented transition network model presented here which distinguishes it from the Thorne and Bobrow systems is the language for the specification of the transition network grammar. This language is designed to be convenient and natural for the grammar designer rather than for the machine or for a computer programmer. It is possible in a few pages to completely specify the possible syntactic forms for the representation of an augmented transition network. Each arc is represented by a mnemonic name of the type of arc, the arc label, an arbitrary condition, and a list of actions to be executed if the arc is followed. The condition and actions are represented as expressions in Cambridge Polish notation with mnemonic function names, and care has been exercised to provide a basic repertoire of such functions which is "natural" to the task of natural language analysis. One of the goals of the experimental transition network parsing system which I have implemented is to evolve such a set of natural operations through experience writing grammars for it, and many of the basic operations described in this paper are the result of such evolution. One of the unique characteristics of the augmented transition network model is the facility to allow for evolution of this type.

[3] In the experimental parsing system which has been implemented, there is sometimes an advantage to using conditions or actions which apply to features of the implementation that are not in the formal model. Actions of this sort are considered to be *extensions* to the basic model, and the features of the implementation which allow them to be added easily are largely features of the BBN LISP system [2] in which the system is written.

7. Advantages of the Augmented Transition Network Model

The augmented transition network model of grammar has many advantages as a model for natural language, some of which carry over to models of programming languages as well. In this section we review and summarize some of the major features of the transition network model which make it an attractive model for natural language.

7.1. Perspicuity. Context-free grammars have been immensely successful (or at least popular) as models for natural language in spite of formal inadequacies of the model for handling some of the features that occur in existing natural languages. They maintain a degree of "perspicuousness" since the constituents which make up a construction of a given type can be read off directly from the context-free rule. That is, by looking at a rule of a context-free grammar, the consequences of that rule for the types of constructions that are permitted are immediately apparent. The pushdown store automaton, on the other hand, although equivalent to the context-free grammar in generative power does not maintain this perspicuousness. It is not surprising, therefore, that linguists in the process of constructing grammars for natural language have worked with the context-free grammar formalism and not directly with pushdown store automata even though the pushdown store automaton, through its finite state control mechanism, allows for some economies of representation and for greater efficiency in resulting parsing algorithms.

The theory of transformational grammar proposed by Chomsky [6] is one of the most powerful tools for describing the sentences that are possible in a natural language and the relationships that hold among them, but this theory as it is currently formalized (to the limited extent to which it is formalized) loses the perspicuousness of the context-free grammar. It is not possible in this model to look at a single rule and be immediately aware of its consequences for the types of construction that are possible. The effect of a given rule is intimately bound up with its interrelation to other rules, and in fragments of transformational grammars for real languages it may require an extremely complex analysis to determine the effect and purpose of any given rule. The augmented transition network provides the power of a transformational grammar but maintains much of the perspicuousness of the context-free grammar model. If the transition network model were implemented on a computer with a graphics facility for displaying the network, it would be one of the most perspicuous (as well as powerful) grammar models available.

7.2. Generative Power. Even without the conditions and actions on the arcs, the recursive transition network model has greater strong generative power than the ordinary context-free grammar. This is due to its ability to characterize constructions which have an unbounded number of immediate constituents. Ordinary context-free grammars cannot characterize trees with unbounded branching without assuming an infinite set of rules. Another way of looking at the recursive transition network model is that it is a finite representation of a context-free grammar with a possibly infinite (but regular) set of rules. When conditions and actions are added to the arcs, the model attains the power of a Turing machine, although the basic operations which it performs are "natural" ones for language analysis. Using these conditions and actions, the model is capable of performing the equivalent of transformational analysis without the need for a separate transformational component.

Another attractive feature of the augmented transition network model is the fact that one does not seem to have to sacrifice efficiency to obtain power. In the progression from context-free grammars to context-sensitive grammars to transformational grammars, the time required for the corresponding recognition algorithms increases enormously. The transition network model, however, while achieving all the power of a transformational grammar, does so without apparently requiring much more time than is required for predictive context-free recognition. (This is illustrated to some extent by the example in Section 8.)

An additional advantage of the augmented transition network model over the transformational grammar model is that it is much closer to a dual model than the transformational grammar. That is, although we have described it as a recognition or analysis model which *analyzes* sentences, there is no real restriction against running the model in a generative mode to *produce* or *generate* sentences. The only change in operation that would be required is that conditions which look ahead in the sentence would have to be interpreted in the generation algorithm as decisions to be made which, if chosen, will impose constraints on the generation of subsequent portions of the sentence. The transformational grammar model, on the other hand, is almost exclusively a generative model. The analysis problem for the transformational grammar is so extremely complicated that no reasonably efficient recognition algorithm for transformational grammar has yet been found.

7.3. Efficiency of Representation. A major advantage of the transition network model over the usual context-free grammar model is the ability to merge the common parts of many context-free rules, thus allowing greater efficiency of representation. For example, the single regular-expression rule S → (Q) (NEG) NP VP replaces the four rules:

S → NP VP
S → Q NP VP
S → NEG NP VP
S → Q NEG NP VP

in the usual context-free notation. The transition network model can frequently achieve even greater efficiency through merging because of the absence of the linearity constraints that are present in the regular expression notation.

The merging of redundant parts of rules not only permits a more compact representation but also eliminates the necessity of redundant processing when doing the parsing. That is, by reducing the size of the grammar representation, one also reduces the number of tests which need to be performed during the parsing. In effect, one is taking advantage of the fact that whether or not a rule is successful in the ordinary context-free grammar model, information is frequently gained in the process of matching it (or attempting to match it) which has implications for the success or failure of later rules. Thus, when two rules have common parts, the matching of the first has already performed some of the tests required for the matching of the second. By merging the common parts, one is able to take advantage of this information to eliminate the redundant processing in the matching of the second rule.

In addition to the direct merging of common parts of different rules when constructing a transition network model, the *augmented* transition network, through its use of flags, allows for the merging of *similar* parts of the network by recording information in registers and interrogating it with conditions on the arcs. Thus it is possible to store in registers some of the information that would otherwise be implicitly remembered by the state of the network and to merge states whose transitions are similar except for conditions on the contents of registers. For example, consider two states whose transitions are alike except that one is "remembering" that a negative particle has already been found in the sentence, while the other permits a transition which will accept a negative particle. These two states can be merged by setting a flag to indicate the presence of a prior negative particle and placing a condition on the arc which accepts the negative particle to block it if the negative flag is set.

The process of merging similar parts of the network through the use of flags, while producing a more compact representation, does not result in an improvement in processing time and usually requires slightly more time. The reason for this is the increased time required to test the conditions and the presence of additional arcs which must be processed even though the conditions will prevent them from being followed. In the absurd extreme, it is possible to reduce any transition network to a one-state network by using a flag for each arc and placing conditions on the arcs which forbid them to be followed unless one of the flags for a possible immediately preceding arc has been set. The obvious inefficiency here is that at every step it would be necessary to consider each arc of the network and apply a complicated test to determine whether the arc can be followed. There is thus a trade-off between the compactness of the representation which can be gained by the use of flags and the increase in processing time which may result. This seems to be just one more example of the ubiquitous space-time trade-off that occurs for almost any computer programming problem.

In many cases, the use of registers to hold pieces of an analysis provides automatic flags, so that it is not necessary to set up special registers to remember such information. For example, the presence of a previous negative particle in a sentence can be indicated by the nonemptiness of a NEG register which contains the particle. Similarly, the presence of an auxilliary verb is indicated by the nonemptiness of an AUX register which contains the auxilliary verb.

7.4. Capturing Regularities. One of the linguistic goals of a grammar for a natural language is that the grammar capture the *regularities* of the language. That is, if there is a regular process that operates in a number of environments, the grammar should embody that process in a single mechanism or rule and not in a number of independent copies of the same process for each of the different contexts in which it occurs. A simple example of this principle is the representation of the prepositional phrase as a constituent of a sentence because the construction consisting of a preposition followed by a noun phrase occurs often in English sentences in many different environments. Thus the model which did not treat prepositional phrases as constituents would be failing to capture a generality. This principle is a variation of the *economy principle*, which says that the best grammar is that which can characterize the language in the fewest number of symbols. A grammar which made essentially independent copies of the same information would be wasting symbols in its description of the language, and that model which merged these multiple copies into a single one would be a better grammar because it used fewer symbols. Thus the economy principle tends to favor grammars which capture regularities.

The transition network model, with the augmentation of arbitrary conditions on the arcs and the use of registers to contain flags and partial constructions, provides a mechanism for recognizing and capturing regularities. Whenever the grammar contains two or more subgraphs of any size which are essentially copies of each other, it is a symptom of a regularity that is being missed. That is, there are two essentially identical parts of the grammar which differ only in that the finite state control part of the machine is remembering some piece of information, but otherwise the operation of the two parts of the graph are identical. To capture this generality, it is sufficient to explicitly store the distinguishing piece of information in a register (e.g. by a flag) and use only a single copy of the subgraph.

7.5. Efficiency of Operation. In addition to the efficiency of operation which results from the merging of common parts of different rules, the transition network model provides a number of other advantages for efficient operation. One of these is the ability to postpone decisions by reworking the network. A great inefficiency of many grammars for natural language is the procedure whereby the grammar "guesses" some basic feature of a construction too early in the process of recognizing it—for example, guessing whether a sentence is active or passive before the processing of the sentence has begun. This results in the parser having to follow several alternatives until that point in the sentence where enough information is present to rule out the erroneous guesses. A much more desirable approach is to leave the decision unmade until a point in the construction is reached where the necessary information is present to make the decision. The transition network model allows one to take this approach.

By using standard finite state machine optimization techniques (see Woods [32]) it is possible to "optimize" the transition network by making it deterministic except for the pushdown operations (where nondeterminism can be reduced but not necessarily eliminated). That is, if several arcs with the same label leave some state, a modified network can be constructed which has at most one arc with a given label leaving any given state. This results in an improvement in operation efficiency because of the reduced number of active configurations which need to be followed during the parsing. The deterministic network keeps identical looking analyses merged until that point at which they are no longer identical, thus postponing the decision as to which path it is on until the first point where the two paths differ, at which point the input symbol usually determines the correct path. The augmented transition network may not permit the completely automatic optimization which the unaugmented model permits, but it is still possible to adopt the general approach of reducing the number of active configurations by reducing the nondeterminism of the network, thus postponing decisions until the point in the input string where they make a difference. The holding of pieces of the analysis in registers until their appropriate function is determined allows one to wait until such decisions have been made before building the syntactic representation, which may depend on the decision. This facility allows one to postpone decisions even when building deep structure representations of the type assigned by a transformational grammar.

The necessity of following several active configurations during parsing is a result of the potential ambiguity of natural language. The source of this ambiguity lies in the recursion operation of the network, since without recursion the network would be a finite state machine and could be made completely deterministic. We show elsewhere [32] that it is possible to eliminate much of the recursion from a transition network (in fact we can eliminate all of the recursion except for that induced by self-embedding symbols), thus reducing still further the number of active configurations which need to be followed. In the augmented network model, one seems in practice to be able to use conditions on the arcs to determine uniquely when to push down for a recursion, leaving only the action of popping up as the source of ambiguity and the cause for multiple active configurations. The use of appropriate conditions (including semantic ones) on the POP arcs of the network allows one to reduce this ambiguity still further.

One of the most interesting features of the use of registers in the augmented transition network is the ability to make tentative decisions about the sentence structure and then change one's mind later in the sentence without backtracking. For example, when one is at the point in parsing a sentence where he is expecting a verb and he encounters the verb "be," he can tentatively assign it as the main verb by putting it in the main verb register. If he then encounters a second verb indicating that the "be" was not the main verb but an auxilliary helping verb, then the verb "be" can be moved from the main verb register into an auxilliary verb register and the new main verb put in its place. This technique, like the others, tends to reduce the number of active configurations which need to be followed during the parsing. In Section 8 we give an example which provides a number of illustrations of this technique of making tentative decisions and then changing them.

7.6. Flexibility for Experimentation. Perhaps one of the most important advantages of the augmented transition network model is the flexibility that the model provides for experimental linguistic research. The open-ended set of basic operations which can be used on the arcs allows for the development of a fundamental set of "natural" operations for natural language analysis through experience obtained while writing grammars. A powerful BUILDQ function was developed in this way and has proven extremely useful in practice. The use of the hold list and the virtual transitions in Section 8 is another example of the evolution of a special "natural" operation to meet a need.

A second area of experimentation that is facilitated by the transition network model is the investigation of different types of structural representations. The explicit structure-building actions on the arcs of the network allow one to experiment with different syntactic representations such as dependency grammars, tagmemic formulas, or Fillmore's case grammar [11]. It should even be possible to produce some types of semantic representation by means of the structure-building actions on the arcs.

Finally, it is possible to use the conditions on the arcs to experiment with various types of semantic conditions for guiding the parsing and reducing the number of "meaningless" syntactic analyses that are produced. Within the framework of the augmented transition network one can begin to take advantage of much of the extra-syntactic information which human beings seems to have available during parsing. Many good ideas in this area have gone untried for want of a formalism which could accommodate them.

8. A Second Example

In this section we give an example that illustrates some of the advantages of the augmented transition network which we have been discussing—especially the facilities for making tentative decisions that are changed as the

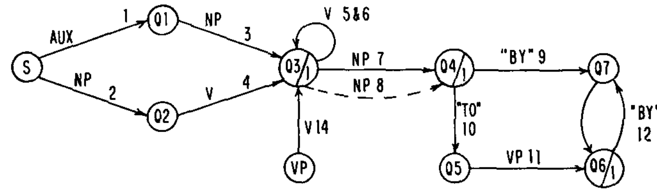


Fig. 4(a). A partial transition network—pictorial representation with numbered arcs

Q3:
  Condition: (INTRANS (GETR V))
  Form: (BUILDQ (S++(TNS+) (VP (V+))) TYPE SUBJ TNS V)
Q4 and Q6:
  Condition: T
  Form: (BUILDQ (S++(TNS+) (VP (V+)+)) TYPE SUBJ TNS V OBJ)

Fig. 4(b). A partial transition network—conditions and forms for final states

	Conditions	*Actions*
1.	T	(SETR V *)
		(SETR TNS (GETF TENSE))
		(SETR TYPE (QUOTE Q))
2.	T	(SETR SUBJ *)
		(SETR TYPE (QUOTE DCL))
3.	T	(SETR SUBJ *)
4.	T	(SETR V *)
		(SETR TNS (GETF TENSE))
5.	(AND (GETF PPRT)	(HOLD (GETR SUBJ))
	(EQ (GETR V)	(SETR SUBJ (BUILDQ
	(QUOTE BE)))	(NP (PRO SOMEONE))))
		(SETR AGFLAG T)
		(SETR V *)
6.	(AND (GETF PPRT)	(SETR TNS (APPEND
	(EQ (GETR V)	(GETR TNS)
	(QUOTE HAVE)))	(QUOTE PERFECT)))
		(SETR V *)
7.	(TRANS (GETR V))	(SETR OBJ *)
8.	(TRANS (GETR V))	(SETR OBJ *)
9.	(GETR AGFLAG)	(SETR AGFLAG NIL)
10.	(S-TRANS (GETR V))	(SENDR SUBJ (GETR OBJ))
		(SENDR TNS (GETR TNS))
		(SENDR TYPE (QUOTE DCL))
11.	T	(SETR OBJ *)
12.	(GETR AGFLAG)	(SETR AGFLAG NIL)
13.	T	(SETR SUBJ *)
14.	(GETF UNTENSED)	(SETR V *)

Fig. 4(c). A partial transition network—conditions and actions on arcs parsing proceeds. Figure 4 gives a fragment of a transition network which characterizes the behavior of the auxiliary verbs "be" and "have" in indicating the passive construction and the perfect tense. We will consider the analysis provided by this sample network for the sentence "John was believed to have been shot"—a sentence with a fairly complex syntactic structure. In doing so, we will see that the augmented transition network clearly characterizes the changing expectations as it proceeds through the analysis, and that it does this without the necessity of backtracking or pursuing different alternatives.

Figure 4 is divided into three parts: (a) a pictorial representation of the network with numbered arcs, (b) a description of the conditions and forms associated with the final states, and (c) a list of the conditions and actions associated with the arcs of the network. In Figure 4(a), the pictorial representation, S, NP, and VP are nonterminal symbols; AUX and V are lexical category names; and the arcs labeled "TO" and "BY" are to be followed only if the input word is "to" or "by" respectively. The dotted arc with label NP is a special kind of "virtual" arc which can be followed if a noun phrase has been placed on a special "hold list" by a previous HOLD command. It removes the item from the hold list when it uses it. The hold list is a feature of the experimental parsing system which provides a natural facility for dealing with constituents that are found out of place and must be inserted in their proper location before the analysis can be complete. The items placed on the hold list are marked with the level at which they were placed on the list, and the algorithm is prevented from popping up from that level until the item has been "used" by a virtual transition at that level or some deeper level.

Final states are represented in Figure 4(a) by the diagonal slash and the subscript 1, a notation which is common in the representation of finite state automata. The conditions necessary for popping up from a final state and the expression which determines the value to be returned are indicated Figure 4(b). The parenthesized representation of tree structure is the same as that used in Section 3.2. Conditions TRANS and INTRANS test whether a verb is transitive or intransitive, respectively, and the condition S-TRANS tests for verbs like "believe" and "want", which can each take an embedded nominalized sentence as its "object". Features PPRT and UNTENSED, respectively, mark the past participle form and the standard untensed form of a verb.

We begin the analysis of the sentence "John was believed to have been shot" in state S, scanning the first word of the sentence "John". Since "John" is a proper noun, the push-down for a noun phrase on arc 2 will be successful, and the actions for that arc will be executed placing the noun phrase (NP (NPR JOHN)) in the subject register SUBJ and recording the fact that the sentence is declarative by placing DCL in the TYPE register. The second word of the sentence "was" allows the transition of arc 4 to be followed, setting the verb register V to the standard form of the verb "BE" and recording the tense of the sentence in the register TNS. The register contents at this point correspond to the tentative decision that "be" is the main verb of the sentence, and a subsequent noun phrase or adjective (not shown in the sample network) would continue this decision unchanged.

In state Q3, the input of the past participle "believed" tells us that the sentence is in the passive and that the verb "be" is merely an auxilliary verb indicating the passive. Specifically, arc 5 is followed because the input word is a past participle form of a verb and the current content of the verb register is the verb "be". This arc revises the tentative decisions by holding the old tentative subject on the special hold list, setting up a new tentative subject (the indefinite someone), and setting the flag AGFLAG which indicates that a subsequent agent introduced by the preposition "by" may specify the subject. The main verb is now changed from "be" to "believe" and the network returns to state Q3 scanning the word "to". The register contents at this point are:

SUBJ:	(NP (PRO SOMEONE))
TYPE:	DCL
V:	BELIEVE
TNS:	PAST
AGFLAG:	T

and the noun phrase (NP (NPR JOHN)) is being held on the hold list.

None of the arcs leaving state Q3 is satisfied by the input word "to". However, the presence of the noun phrase "John" on the hold list allows the virtual transition of arc 8 to take place just as if this noun phrase had been found at this point in the sentence. (The transition is permitted because the verb "believe" is marked as being transitive.) The effect is to tentatively assign the noun phrase (NP (NPR JOHN)) as the object of the verb "believe". If this were the end of the sentence and we chose to pop up from the resulting state Q4, then we would have the correct analysis "someone believed John".

The input of the word "to" to state Q4 tells us that the "object" of the verb "believe" is not merely the noun phrase "John" but is a nominalized sentence with "John" as its tentative subject. The effect of arcs 10 and 11 is to send down the necessary information to an embedded calculation which will complete the embedded clause and return the result as the object of the verb "believe". Arc 10 prepares to send down the noun phrase (NP (NPR JOHN)) as the embedded subject, the tense PAST, and the type DCL. Arc 11 then pushes down to state VP scanning the word "have".

At this point, we find ourselves in an embedded computation with the register contents:

SUBJ:	(NP (NPR JOHN))
TYPE:	DCL
TNS:	PAST

Arc 14 permits a transition if the current input is a verb in its standard untensed, undeclined form, i.e. one cannot say "John was believed to *has* been shot." Since "have" is such a form, the transition is permitted and the main verb of the embedded sentence is tentatively set to "have" as would befit the sentence "John was believed to have money."

The subsequent past participle "been" following the verb "have" causes transition 6, which detects the fact that the embedded sentence is in the perfect tense (the effect of the auxilliary "have") and adopts the new tentative verb "be" as would befit the sentence "John was believed to have been a druggist." The register contents for the embedded computation at this point are:

SUBJ:	(NP (NPR JOHN))
TYPE:	DCL
TNS:	PAST PERFECT
V:	BE

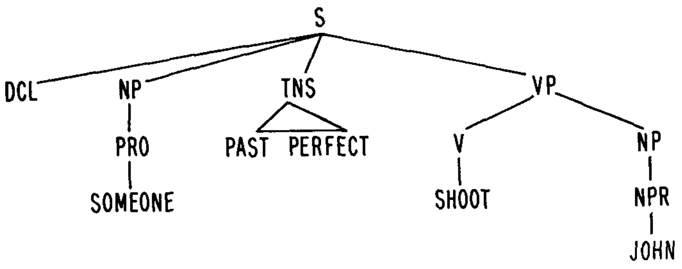
Once again in state Q3, the input of the past participle "shot" with a tentative verb "be" in the verb register indicates that the sentence is in the passive, and transition 5 puts the noun phrase (NP (NPR JOHN)) on the hold list and sets up the indefinite subject (NP (PRO SOMEONE)). Although we are now at the end of the sentence, both the presence of the noun phrase on the hold list and the fact that the verb "shoot" is transitive prevent the algorithm from popping up. Instead, the virtual transition of arc 8 is followed, assigning the noun phrase "John" as the object of the verb "shoot". The register contents for the embedded computation at this point are:

SUBJ:	(NP (PRO SOMEONE))
TYPE:	DCL
TNS:	PAST PERFECT
V:	SHOOT
AGFLAG:	T
OBJ:	(NP (NPR JOHN))

At this point, we are at the end of the sentence in the final state Q4, with an empty hold list so that the embedded computation can return control to the higher level computation which called it. The value returned, as specified by the form associated with the state Q4, is

```
(S DCL (NP (PRO SOMEONE)) (TNS PAST PERFECT)
       (VP (V SHOOT) (NP (NPR JOHN))))
```

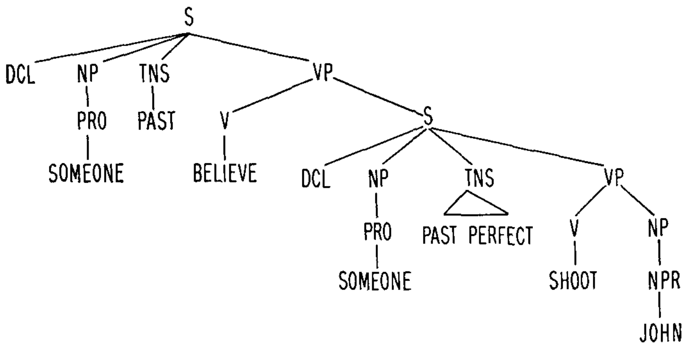
corresponding to the tree:



The higher level computation continues with the actions on arc 11, setting the OBJ register to the result of the embedded computation. Since the higher level computation is also in a final state, Q6, the sentence is accepted, and the structure assigned to it (as specified by the form associated with state Q6) is:

```
(S DCL (NP (PRO SOMEONE)) (TNS PAST) (VP (V BELIEVE)
  (S DCL (NP (PRO SOMEONE)) (TNS PAST PERFECT)
        (VP (V SHOOT) (NP (NPR JOHN))))))
```

which in tree form is represented as:



This structure can be paraphrased "Someone believed that someone had shot John." If the sentence had been followed by the phrase "by Harry" there would have been two possible interpretations depending on whether the additional phrase were accepted by the embedded computation or the top level computation. Either case would have resulted in replacing one of the indefinite subjects SOMEONE with the definite subject "Harry". The structure produced in one case would be paraphrased "Someone believed that Harry had shot John", while the other would be "Harry believed that someone had shot John."

9. Parsing with Transition Network Grammars

Since the unaugmented transition network grammar model is actually a mere permutation of the elements of a pushdown store automaton, a number of existing parsing algorithms for context-free grammars apply more or less directly to the transition network model. The basic top-down and bottom-up parsing strategies have their analog for recursive transition networks, and the opposing strategies for dealing with ambiguous sentences—following all of the analyses "in parallel" or following one analysis at a time (saving information at the choice points)—are both applicable to this type of model. In particular, one of the most powerful and efficient parsers for context-free grammars that has yet been discovered, the Earley recognition algorithm [9, 10], can be adapted with minor modification to use transition network grammars, and indeed an improvement in operating efficiency can accrue from doing so.

The Earley Algorithm, by following all analyses in parallel in a particularly careful way, obtains a representation of all the parses of a string with respect to a context-free grammar in an amount of time which can be bounded by Kn^3, where n is the length of the string and K is a constant depending only on the grammar and not on the input string. Moreover, for certain subclasses of context-free grammars, the time required can be shown to lie within smaller bounds—n^2 for linear grammars and others, and n for $LR(k)$ (Knuth [17]) grammars with lookahead k (and others). Many of these results have been obtained by other algorithms which capitalize on special features of different classes of grammars [15, 16, 33], but the Earley algorithm is the only such algorithm which works for any given context-free grammar (with no restriction to special forms, absence of left-recursion, etc.) within the n^3 bound, and furthermore, it *automatically* achieves the smaller bounds for the special case grammars without having to be told that the grammar falls within a special category (i.e. it does not invoke special techniques for these cases).

We give elsewhere (Woods [32]) a modified version of the Earley algorithm which can be used to parse sentences with respect to an (unaugmented) transition network grammar within the same time bounds, and we show there that a number of mechanical "optimization" techniques can be applied to a transition network grammar to reduce the constant of proportionality in the time bound.

The parsing problem for *augmented* transition networks using the Earley algorithm is somewhat more complicated due to the carrying of information in registers and the use of explicit structure-building actions. The potential for transitions which are conditional on the contents of registers makes it difficult to determine when configurations are "equivalent" and can be merged for further processing, and the use of registers and explicit structure-building actions complicates the task of choosing a suitable representation for the merged configurations. It is relatively straightforward to extend the Earley algorithm to overcome these difficulties, but since the n^3 time bound depends critically on the merging of equivalent configurations and a fixed bound on the amount of time to process each transition from a (merged) state, it is not clear what the bounds on the resulting algorithm will be.

If we distinguish between "flag" registers which can contain only "flags" chosen from a finite vocabulary and "content" registers which can hold arbitrary structure, and if we restrict the conditions and actions on the arcs so that: (1) the conditions can refer only to flag registers and symbols in the input string (e.g. for lookahead); (2) the conditions and actions themselves require a bounded amount of time; and (3) there is only one content register, which can be added to at the ends or built upon by means of the BUILDQ function but which cannot be "looked into", then we can build a version of the Earley recognition algorithm which will operate within the general n^3 time bound (and smaller n^2 or n bounds in special cases). However, if we relax these conditions appreciably, an increase in the time bound is inevitable—e.g. a condition or an action can itself require more than n^3 steps.

9.1. The Case for Series Parsing. For many applications of natural language analysis it is not necessary (or even desirable) to obtain a representation of all of the possible parsings of the input sentence. In applications where natural language is to be used as a medium of communication between a man and a machine, it is more important to select the "most likely" parsing in a given context and to do this as soon as possible. There will undoubtedly be cases where there are several "equally likely" parsings or where the "most likely" parsing turns out not to be the "correct" one (i.e. the one intended by the speaker); hence a nondeterministic algorithm is still required and a facility for eventually discovering any particular parsing is necessary. What is not necessary is a parallel approach which spends time discovering all of the analyses at once. In such an application a series approach (with an appropriate mechanism for selecting which analysis to follow first) has more to offer than a parallel approach since in most cases it will simply avoid following up the other alternatives. An appropriate record can be kept (if desired) of the analyses of well-formed substrings discovered by previous alternatives (in order to eliminate repetitive analysis of the same substring), but the savings in parse time for such an approach does not always justify the storage required to store all of the partial substring analyses.

The success of the series approach described above depends, of course, on the existence of a mechanism for selection of the semantically "most likely" parsing to be followed first. The augmented transition network grammar provides several such mechanisms. First, by ordering the arcs which leave the states of the network, it is possible to impose a corresponding ordering on the analyses which result. The grammar designer can thus adjust this ordering in an attempt to maximize the "a priori likelihood" (dependent only on the structure of the sentence as seen by the grammar, but not on other factors) that the first analysis chosen will be the correct one. Furthermore, by replicating some arcs with different conditions, it is possible to make this ordering dependent on particular features of the sentence being processed—in particular it can be made dependent on semantic features of the words involved in the sentence. Two additional features for selecting "most likely" analyses have been added to the model in the implemented experimental parsing system—a special "weight" register which contains an estimate of the "likelihood" of the current analysis (which can be used to suspend unlikely looking paths in favor of more likely ones) and a selective modifier-placement facility which uses semantic information to determine the "most likely" governing construction for modifiers in the sentence.

10. Implementation

An experimental parsing system based on the ideas presented here has been implemented in BBN LISP on the SDS 940 time-sharing systems at Harvard University and at Bolt, Beranek & Newman, Inc., and is being used for a number of experiments in grammar development and parsing strategies for natural language analysis. The objectives of this implementation have been the provision of a flexible tool for experimentation and evolution, and for this reason the system has been constructed in a modular fashion which lends itself to evolution and extension without major changes to the overall system structure. The system has already undergone several cycles of evolution with a number of new features being developed in this way, and more are expected as the research continues.

The implemented system contains a general facility for semantic interpretation (described in Woods [30, 31]), and a major motivation for the implementation is to explore the interaction between the syntactic and semantic aspects of the process of sentence "understanding". Special emphasis has been placed on the use of semantic information to guide the parsing, the minimization of the number of blind-alley analysis paths which need to be followed, and the ordering of analyses of sentences in terms of some measure of "likelihood". Experiments to date include a selective modifier placement facility using semantic information, several approaches to the problems of conjunction (including conjoined sentence fragments), and a facility for lexical and morphological analysis. Several different grammars have been developed and tested on the system, and a variety of English constructions and parsing strategies have been and are being explored. A report of the details of this implementation and of the experiments which are being performed with it is in preparation.

Received May 1970

REFERENCES

1. Bobrow, D. G., and Fraser, J. B. An augmented state transition network analysis procedure. Proc. Internat. Joint Conf. on Artificial Intelligence, Washington, D.C., 1969, pp. 557–567.
2. Bobrow, D. G., Murphy, D., and Teitelman, W. *BBN LISP System.* Bolt, Beranek and Newman Inc., Cambridge, Mass., 1968.
3. Book, R., Even, S., Greibach, S., and Ott, G. Ambiguity in graphs and expressions. Mimeo. rep., Aiken Computat. Lab., Harvard U., Cambridge, Mass., 1969. *IEEE Trans Comp* (to appear).
4. Cheatham, T. E., and Sattley, K. Syntax-directed compiling. Proc. AFIPS 1964 Spring Joint Comput. Conf., Vol. 25, Spartan Books, New York, pp. 31–57.
5. Chomsky, N. Formal properties of grammars. In *Handbook of Mathematical Psychology, Vol. 2*, R. D. Luce, R. R. Bush, and E. Galanter (Eds.). Wiley, New York, 1963.
6. ——. A transformational approach to syntax. In *The Structure of Language*, J. A. Fodor, and J. J. Katz (Eds.), Prentice-Hall, Englewood Cliffs, N. J., 1964.
7. ——. *Aspects of the Theory of Syntax.* MIT Press, Cambridge, Mass., 1965.
8. Conway, M. E. Design of a separable transition-diagram compiler. *Comm. ACM 6*, 7 (July 1963), 396–408.
9. Earley, J. An efficient context-free parsing algorithm. Ph.D.th. Dep. Computer Sci., Carnegie-Mellon U., Pittsburgh, Pa., 1968.
10. Earley, J. An efficient context-free parsing algorithm. *Comm. ACM 13*, 2, (Feb. 1970), 94–102.
11. Fillmore, C. J. The case for case. In *Universals in Linguistic Theory*, E. Bach, and R. Harms (Eds.), Holt, Rinehart and Winston, New York, 1968.
12. Ginsburg, S. *The Mathematical Theory of Context-Free Languages.* McGraw-Hill, New York, 1966.
13. Greibach, Sheila A. A simple proof of the standard-form theorem for context-free grammars. In Mathematical linguistics and automatic translation, Rep. NSF-18, Comput. Lab., Harvard U., Cambridge, Mass., 1967.
14. Herringer, J., Weiler, M., and Hurd, E. The immediate constituent analyzer. In Rep. NSF-17, Aiken Comput. Lab., Harvard U., Cambridge, Mass., 1966.
15. Kasami, T. An efficient recognition and syntax-analysis algorithm for context-free languages. Sci. Rep. AFCRL-65-558, Air Force Cambridge Res. Lab., Bedford, Mass., 1965.
16. Kasami, T. A note on computing time for recognition of languages generated by linear grammars. *Inform. Contr.*, *10* (1964), 209–214.
17. Knuth, D. E. On the translation of languages from left to right. *Inf. Contr. 8* (1965), 607–639.
18. Kuno, S. "A system for transformational analysis. In Rep. NSF-15, Comput. Lab. Harvard U., Cambridge, Mass, 1965.
19. —— and Oettinger, A. G. Multiple path syntactic analyzer. In *Information Processing 1962*, North-Holland Publishing Co., Amsterdam, 1963.
20. McCarthy, J., et al. LISP 1.5 programmer's manual. MIT Comput. Center, Cambridge, Mass., 1962.
21. McCawley, J. D. Meaning and the description of languages. In *Kotoba No Ucho*, TEC Co. Ltd., Tokyo, 1968.
22. McNaughton, R. F., and Yamada, H. Regular expressions and state graphs for automata. *IRE Trans. EC-9* (Mar. 1960), 39–47.
23. Matthews, G. H. Analysis by synthesis of natural languages. *Proc. 1961 Internat. Conf. on Machine Translation and Applied Language Analysis.* Her Majesty's Stationery Office, London, 1962.
24. MITRE. English preprocessor manual, Rep. SR-132, The Mitre Corp., Bedford, Mass., 1964.
25. Ott, G., and Feinstein, N. H. Design of sequential machines from their regular expressions," *J. ACM 8*, 4 (Oct. 1961), 585–600.
26. Petrick, S. R. A recognition procedure for transformational grammars. Ph.D. th., Dep. Modern Languages, MIT, Cambridge, Mass., 1965.
27. Postal, P. M. Limitations of phrase structure grammars. In *The Structure of Language*, J. A. Fodor and J. J. Katz (Eds.), Prentice-Hall, Englewood Cliffs, N.J., 1964.
28. Schwarcz, R. M. Steps toward a model of linguistic performance: A preliminary sketch. *Mechanical Translation 10* (1967), 39–52.
29. Thorne, J., Bratley, P., and Dewar, H. The syntactic analysis of English by machine. In *Machine Intelligence 3* D. Michie (Ed.), American Elsevier, New York, 1968.
30. Woods, W. A. Semantics for a question-answering system. Ph.D. th., Rep NSF-19, Aiken Comput. Lab., Harvard U., Cambridge, Mass., 1967.
31. ——. Procedural semantics for a question-answering machine. Proc. AFIPS 1968 Fall Joint Comput. Conf., Vol. 33, Pt. 1, MDI Publications, Wayne, Pa., pp. 457–471.
32. ——. Augmented transition networks for natural language analysis. Rep. CS-1, Comput. Lab., Harvard U., Cambridge, Mass., 1969.
33. Younger, D. H. Context free language processing in time n^3. G. E. Res. and Devel. Center, Schenectady, N.Y., 1966.