ATN GRAMMAR MODELING IN APPLIED LINGUISTICS

T. P. KEHLER
Department of Mathematics and Physics
Texas Woman's University

R. C. WOODS
Department of Computer Science
Virginia Technological University

ABSTRACT: Augmented Transition Network grammars have significant areas of unexplored application as a simulation tool for grammar designers. The intent of this paper is to discuss some current efforts in developing a grammar testing tool for the specialist in linguistics. The scope of the system under discussion is to display structures based on the modeled grammar. Full language definition with facilitation of semantic interpretation is not within the scope of the systems described in this paper. Application of grammar testing to an applied linguistics research environment is emphasized. Extensions to the teaching of linguistics principles and to refinement of the primitive ATN functions are also considered.

1. Using Network Models in Experimental Grammar Design

Application of the ATN to general grammar modeling for simulation and comparative purposes was first suggested by Woods(1). Motivating factors for using the network model as an applied grammar design tool are:

1. The model provides a means of organizing structural descriptions at any level, from surface syntax to deep propositional interpretations.
2. A network model may be used to represent different theoretical approaches to grammar definition.
3. The graphical representation of a grammar permitted by the network model is a relatively clear and precise way to express notions about structure.
4. Computational simulation of the grammar enables systematic tracing of subcomponents and testing against text data.

Grimes (2), in a series of linguistics workshops, demonstrated the utility of the network model even in environments where computational testing of grammars was not possible. Grimes, along with other contributors to the referenced work, illustrated the flexibility of the ATN in tagmemic analysis of grammatical structures. ATN implementations have mostly focused on effective natural language understanding systems, assuming a computationally sophisticated research environment. Implementations are often in an environment which requires some in-depth understanding and support of LISP systems. Recently much of the information on the ATN formalism, applications and techniques for implementation was summarized by Bates (3). Though many systems have been developed, little attention has been given to creating an interactive grammar modeling system for an individual with highly developed linguistics skills but poorly developed computational skills.

The individual involved in field linguistics is concerned with developing concise workable descriptions of some corpus of data in a given language. Particular problems in developing rules for interpreting surface structures are proposed and discussed in relation to the data. In field linguistics applications, this involves developing a taxonomy of structural types followed by hypothesizing underlying rule systems which provide the highest level of data integration at a syntactic as well as semantic level of analysis. The ATN is proposed as a tool for assisting the linguist to develop systematic descriptions of language data. It is assumed that the typical user will interface with the system at a point where an ATN and lexicon have been developed. The ATN is developed from the theoretical model chosen by the linguist.

Once the ATN is implemented as a computational procedure, the user enters test data, displays structures, examines the lexicon, and edits the grammar to produce a refined ATN grammar description. The displayed structures provide a labeled structural interpretation of the input string based on the linguistic model used. Tracing of the parse may be used to follow the process of building the structural interpretation. Computational implementation requires giving attention to the details of the interrelationships of grammatical rules and the interaction between the grammar rule system and the lexical representation. Testing the grammar against data forces a level of systemization that is significantly more rigorous than discussion oriented evaluation of grammar systems.

2. Design Considerations

The general design goal for the grammar testing system described here is to provide a tool for developing experimentally driven, systematic representation models of language data. Engineering of a full language understanding system is not the primary focus of the efforts described in this paper. Ideally, one would like to provide a tool which would attract applied linguists to use such a system as a simulation environment for model development.

The design goals for the systems described are:

1. Ease of use for both novice and expert modes of operation,
2. Perspicuity of grammar representation,
3. Support for a variety of linguistic theories,
4. Transportability to a variety of systems.

The prototype grammar design system consists of a grammar generator, an editor, and a monitor. The function of the grammar editor is to provide a means of defining and manipulating grammar descriptions without requiring the user to work in a specific programming language environment. The editor is also used to edit lexicons. The editor knows about the ATN environment and can provide assistance to the user as needed.

The monitor's function is to handle input and output of grammar and lexicon files, manage displays and traces of parsings, provide consultation on the system use as needed, and enable the user to cycle from editor to parsing with minimum effort. The monitor can also be used to provide facilities for studying grammar efficiency. Transportability of the grammar modeling system is established by a program generator which enables implementation in different programming languages.

3. Two Implementations of Grammar Testing Systems

To develop some understanding on the design and implementation requirements for a system as specified in the previous section, two experimental grammar testing systems have been developed. A partial ATN implementation was done by Kehler(4) in a system (SNOPAR) which provided some interactive grammar and development facilities. SNOPAR incorporated several of the basic features of a grammar generator and monitor, with a limited editor, a grammar generator and a number of other features.

Both SNOPAR and ADEPT are implemented in SNOBOL and both have been transported across operating systems (i.e. TOPS-20 to IBM's CMS). For implementation of text editing and program grammar generation, the SNOBOL4 language is reasonable. However, the lack of comprehensive list storage management is a limitation on the extension of this implementation to a full natural language understanding system. Originally, SNOBOL was used because a suitable LISP was not available to the implementor.

3.1 SNOPAR

SNOPAR provides the following functions: grammar creation and editing, lexicon creation and editing, execution (with some error trapping), tracing/debugging and file handling. The grammar creation portion has as an option use of an interactive grammar to create an ATN. One of the goals in the design of SNOPAR was to introduce a notation which was easier to read than the LISP representation most frequently used.

Two basic formats have been used for writing grammars in SNOPAR. One separates the context-free syntax type operations from the tests and actions of the grammar. This action block format is of the following general form:

```
arc-type-block
state      arc-type          :S(TO(test-action-block))
           arc-type          :S(TO(test-action-block))
 '
 '                           :F(FRETURN)
```

where arc-type is a CAT, PARSE or FINDWRD etc., and the test-action-block appears as follows:

```
test-action-block
state    arc-test ! action  :S(TO(arc-type-block))
         arc-test ! action  :S(TO(arc-type-block))
```

where an arc-test is a COMPAR or other test and an action is a SETR or BUILDS type action. Note that an additional intermediate state is introduced for the test and actions of the ATN.

The more standard format used is given as:

```
state--> arc-type --> condition-test-and-action-block
                                          --> new-state
```

An example noun phrase is given as:

```
NP CAT('DET') SETR('NP','DET',Q)   :S(TO('ADJ'))
  CAT('NPR') SETR('NP','NPR',Q)
                          :S(TO('POPNP'))F(FRETURN)
ADJ CAT('ADJ') SETR('NP','ADJ',Q)  :S(TO('ADJ'))
  CAT('N') SETR('NP','N',Q)
                          :S(TO('N'))F(FRETURN)
NPP PARSE(PP()) SETR('NP','NPP',Q):S(TO('NPP'))
POPNP NP = BUILDS(NP)     :(RETURN)
```

The Parse function calls subnetworks which consist of Parse, Cat or other arc-types. Structures are initially built through use of the SETR function which uses the top level constituent name (e.g. NP) to form a list of the constituents referenced by the register name in SETR. All registers are treated as stacks. The BUILDS function may use the implicit register name sequence as a default to build the named structure. The top level constituent name (i.e. NP) contains a list of the registers set during the parse which becomes the default list for structure building. There are global stacks for history taking and back up functions.

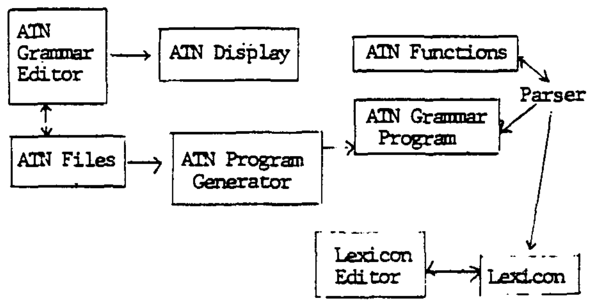
Typically, for other than the initial creation of a grammar by a naive user, the ATN function library of the system is used in conjunction with a system editor for grammar development. Several ATN grammars have been written with this system.

3.2 ADEPT

In an effort to make an easy-to-use simulation tool for linguists, the basic concepts of SNOPAR were extended by Woods(5) to a full ATN implementation in a system called ADEPT. ADEPT is a system for generating ATN programs through the use of a network editor, lexicon editor, error correction and detection component, and a monitor for execution of the grammar. Figure 1 shows the system organization of ADEPT.

The editor in ADEPT provides the following functions:

- network creation
- arc deletion or editing
- arc insertion
- arc reordering
- state insertion and deletion



The four main editor command types are summarized below:

E	<net>	Edits a network (Creates it if it doesn't exist)
E	<state>,<state>	Edit arc information
D	<net>	Deletes a network
D	<state>	Deletes a state
D	<state>,<state>	Delete an arc
I	<state>	Insert a state
I	<state>,<state>	Insert an arc
O	<state>	Order arcs from a state
L	<filename>	List networks

State, network, and arc editing are distinguished by context and the arguments of the E, D, or I commands. For a previously undefined E net causes definition of the network. The user must specify all states in the network before starting. The editor processes the state list requesting arc relations and arc information such as the tests or arc actions. The states are used to help diagnose errors caused by misspelling of a state or omission of a state.

Once the network is defined, arcs may by edited by specifying the origin and destination of the arc. The arc information is presented in the following order: arc destination, arc type, arc test and arc actions. Each of these items is displayed, permitting the user to change values on the arc list by typing in the needed information. Multiple arcs between states are differentiated by specifying the order number of the arc or by displaying all arcs to the user and requesting selection of the desired arc.

New arcs are inserted in the network by the I command. Whenever an arc insert is performed all arcs from the state are numbered and displayed. After the user specifies the number of the arc that the new arc is to follow, the arc information is entered.

Arcs may be reordered by specifying the starting state for the arcs of interest using the O command. The user is then requested to specify the new ordering of the arcs.

Insertion and deletion of a state requires that the editor determine the states which may be reached from the new state as well as finding which arcs terminate on the new state. Once this information has been established, the arc information may be entered.

When a state is deleted, all arcs which immediately leave the state or which enter the state from other states are removed. Error conditions existing in the network as a result of the deletion are then reported. The user then either verifies the requested deletion and corrects any errors or cancels the request.

Grammar files are stored in a list format. The PUT command causes all networks currently defined to be written out to a file. GET will read in and define a grammar. If the network is already defined, the network is not read in.

By placing a series of checking functions in an ATN editor, it is possible to filter out many potential errors before a grammar is tested. The user is able to focus on the grammar model and not on the specific programming requirements. A monitor program provides a top level interface to the user once a grammar is defined for parsing sentences. In addition, the monitor program manages the stacks as well as the SEND, LIFT and HOLD lists for the network grammar. Switches may be set to control the tracing of the parse.

An additional feature of the Woods ADEPT system is the use of easy to read displays for the lexicon and grammar. An example arc is shown:

```
(NP)--CAT('DET')-- (ADJ)
        ***NO TESTS. ***
     ACTIONS
        SETR('DET')
```

ADEPT has been used to develop a small grammar of English. Future experiments are planned for using ADEPT in an linguistics applications oriented environment.

4. Experiments in Grammar Modeling

Utilization of the ATN as a grammar definition system in linguistics and language education is still at an early stage of development. Weischedel et.al.(6) have developed an ATN-based system as an intelligent CAI too for teaching foreign language. Within the SNOPAR system, experiments in modeling English transformational grammar exercises and modeling field linguistics exercises have been carried out. In field linguistics research some grammar development has been done. Of interest here is the systematic formulation of rule systems associated with the syntax and semantics of natural language subsystems. Proposed model grammars can be evaluated for efficiency of representation and extendibility to a larger corpus of data. Essential to this approach is the existence of a self-contained easy-to-use transportable ATN modeling systems. In the following sections some example applications of grammar testing to field linguistics exercises and application to modeling a language indigenous to the Philippines are given.

4.1 An Exercise Computationally Assisted Taxonomy

Typical exercises in a first course in field linguistics give the student a series of phrases or sentences in a language not known to the student. Taxonomic analysis of the data is to be done producing a set of formulas for constituent types and the hierarchical relationship of constituents. In this particular case a tagmemic analysis is done. Consider the following three sentences selected from Apinaye exercise (Problem 100)(7):

kukren kokoi	the monkey eats
kukren kokoi rach	the big monkey eats
ape rach mih mech	the good man works well

First a simple lexicon is contructed, from this and other data. Secondly, immediate constituent analysis is carried out to yield the following tagmemic formulae:

```
ICL := Pred:VP + Subj:NP
NP  := Head:N + Nmod:AD
VP  := Head:V + Vmod:AD
```

The ATN is then defined as a simple syntactic organization of constituent types. The SNOPAR representation of this grammar would be:

```
ICL       PARSE(VP()) SETR('ICL','Pred',Q)
                               :S(TO('SU'))F(FRETURN)
SU        PARSE(NP()) SETR('ICL','Subj',Q)
                               :S(TO('POPICL'))F(FRETURN)
POPICL    ICL = BUILDS(ICL)    :(RETURN)
VP        CAT('V') SETR('VP','Head',Q)
                               :S(TO('VMOD'))F(FRETURN)
VMOD      CAT('AD') SETR('VP','VMod',Q)
POPVP     VP = BUILDS(VP)      :(RETURN)
NP        CAT('N') SETR('NP','Head',Q)
                               :S(TO('NMOD'))F(FRETURN)
NMOD      CAT('AD') SETR('NP','NMod',Q)
POPNP     NP = BUILDS(NP)      :(RETURN)
END
```

Thus permitting the parse of the first sentence (Kukren kokoi) as:

```
(ICL
   (Pred
      (VP
         (Head KUKREN)))
   (Subj
      (NP
         (Head KOKOI))))
```

English gloss may be used as in the following example:

```
GLOSS:
WORK MUCH MAN WELL/GOOD     The good man works a lot.
STATE:ICL INPUT:
(ICL
   (Pred
      (VP
         (Head APE
         (VMod RACH)))
   (Subj
      (NP
         (Head MIH)
         (NMod MECH))))
```

Each sentence in the exercise may be entered, making corrections to the grammar as needed. Once the basic notions of syntax and hierarchy are established, the model may then be extended to incorporate context-sensitive and semantic features. Frequently, in proposing a taxonomy for a series of sentences, one is tempted to propose numerous structural types in order to handle all of the data. The orientation of grammar testing encourages the user to look for more concise representations. Tracing the sentence parse can yield information about the efficiency of the representation. Tracing is also illustrative to the student, permitting many parsings to be observed.

4.2 Cotabato Manobo

An ATN representation of a grammar for Cotabato Manobo was done by Errington(8) using the manual proposed by Grimes(2). Recently, the grammar was implemented and tested using SNOPAR. The implementation took place over a three month period with initial implementation at the word level and eventual extension to the clause level with conjunctions and embedding. Comments were used throughout the grammar to explain the rational for particular arc types, tests or actions.

A wide variety of clause types are handled by the grammar. A specific requirement in the Manobo grammar is the ability to handle a significant amount of testing on the arcs. For example, it is not uncommon to have three or four arcs of the same type differentiated by checks on registers from previous points in the parse. With nine network types, this leads to a considerable amount of time being spent in context checking. A straight forward approach to the grammar design leads to a considerable amount of backing up in the parse. While a high speed parse was not an objective of the design, it did point out the difficulty in designing grammars of significant size without getting in to programming practice and applying more efficient parsing routines. Since an objective of the project is to provide a system which emphasizes the linguistics and not programming practice, it was necessary to maintain descriptive clarity at the sacrifice of performance. An example parse for a clause is given:

```
EG KÀEN SA ETAW SA BEGAS -- The person is eating rice
GLOSS:
EAT THE PERSON.PEOPLE THE RICE
STATE:CL INPUT:
(CL
   (VP
      (VB
         (VBWD
            (VAFF EG)                  action is 'eat'
            (VINS PRES)
            (VMOD BASIC)
            (VFOC ACTORF)
            (VTYPE1 TRANS)
            (VSTEM1 KAEN)))
      (VPTYPE VERB)))
   (FOC                                focus is 'the people'
      (NP
         (DET SA)
         (NUC
            (NPNUC
               (N ETAW)))))
(ACTOR                                 actor is 'the people'
   (NP
      (DET SA)
      (NUC
         (NPNUC
            (N ETAW)))))
(NONACT                                object is 'rice'
   (NP
      (DET SA)
      (NUC
         (NPNUC
            (N BEGAS))))))
```

5. Summary and Conclusions

Development of a relatively easy to use, transportable grammar design system can make possible the use of grammar modeling in the applied linguistics environment, in education and in linguistics research. A first step in this effort has been carried out by implementing two experimental systems, SNOPAR and ADEPT, which emphasize notational clarity and an editor/monitor interface to the user. The network editor is designed to provide error handling, correction and interaction with the user in establishing a network model of the grammar.

Some applications of SNOPAR have been made to testing taxonomically based grammars. Future use of ADEPT in the linguistics education/research is planned.

Developing a user-oriented ATN modeling system for linguists provides certain insights to the ATN model itself. Such things as the perspicuity of the ATN representation of a grammar and the ATN model applicability to a variety of language types can be evaluated. In addition, a more widespread application of ATNs can lead to some standardization in grammar modeling.

The related issue of developing interfaces for user extension of grammars in natural language processing systems can be investigated from increased use of the ATN model by the person who is not a specialist in artificial intelligence. The systems general design does not limit itself to application to the ATN model.

6. References

1. Woods, W., Transition Network Grammars for Natural Language Analysis, Communications of the ACM, vol. 13, no. 10, 1970.

2. Grimes, J., Transition Network Grammars, A Guide, Network Grammars, Grimes, J., ed., 1975.

3. Bates, Madelein, The Theory and Practice of Augmented Transition Network Grammars, Lecture Notes in Computer Science, Goos, G. and Hartmanis, J., ed., 1978.

4. Kehler, T.P., SNOPAR: A Grammar Testing System, AJCL 55, 1976.

5. Woods, C.A., ADEPT - Testing System for Augmented Transition Network Grammars, Masters Thesis, Virginia Tech, 1979.

6. Weischedel, R.M., Voge, W.M., James, M., An Artificial Intelligence Approach to Language Instruction, Artificial Intelligence, Vol. 10, No. 3, 1978.

7. Merrifield, William R., Constance M. Naish, Calvin R. Rensch, Gillian Story, Laboratory Manual for Morphology and Syntax, 1967.

8. Errington, Ross, 'Transition Network Grammar of Cotabato Manobo.' Studies in Philippine Linguistics, edited by Casilda Edrial-Luzares and Austin Hale. Volume 3, Number 2. Manila: Summer Institute of Linguistics. 1979.