PARSING WITH LOGICAL VARIABLES

Timothy W. Finin and Martha Stone Palmer
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104

ABSTRACT

Logic based programming systems have enjoyed an increasing popularity in applied AI work in the last few years. One of the contributions to Computational Linguistics made by the Logic Programming Paradigm has been the **Definite Clause Grammar.** In comparing DCG's with previous parsing mechanisms such as ATN's, certain clear advantages are seen. We feel that the most important of these advantages are due to the use of **Logical Variables** with **Unification** as the fundamental operation on them. To illustrate the power of the Logical Variable, we have implemented an experimental ATN system which treats ATN registers as Logical Variables and provides a unification operation over them. We would like to simultaneously encourage the use of the powerful mechanisms available in DCG's, and demonstrate that some of these techniques can be captured without reference to a resolution theorem prover.

1. Introduction

Logic based programming systems have enjoyed an increasing popularity in applied AI work in the last few years. One of the contributions to Computational Linguistics made by the Logic Programming Paradigm has been the **Definite Clause Grammar.** An excellent introduction to this formalism can be found in [Pereira] in which the authors present the formalism and make a detailed comparison to Augmented Transition Networks as a means of both specifying a language and parsing sentences in a language.

We feel that the major strengths offered by the DCG formalism arise from its use of **Logical variables** with **Unification** as the fundamental operation on them. These techniques can be abstracted from the theorem proving paradigm and adapted to other parsing systems (see [Kay] and [Bossie]). We have implemented an experimental ATN system which treats ATN registers as Logic variables and provides a unification operation over them.

The DCG formalism provides a powerful mechanism for parsing based on a context free grammar. The grammar rule

S -> NP VP

can be seen as the universally quantified logical statement,

For all x, y, and z :
NP(x) /\ VP(y) /\ Concatenate(x,y,z) => S(z).

where "x" and "y" represent sequences of words which can be concatenated together to produce a sentence, "S." Prolog, a programming language based on predicate calculus, allows logical statements to be input as Horn clauses in the following (reversed) form:

```
s(Z) <- np(X),vp(Y),Concatenate(X,Y,Z).
```

The resolution theorem prover that "interprets" the Prolog clauses would take the negation of S as the goal and try and produce the null clause. Thus the preceding clause can be interpreted procedurally as, "To establish goal S, try and establish subgoals, NP, VP and Concatenate." DCG's provide syntactic sugar on top of Prolog so that the arrow can be reversed and the "Concatenate" predicate can be dispensed with. The words in the input string are looked at sequentially each time a "[Word]" predicate is executed which implicitly tests for concatenation (see figure 1). DCG's allow grammar rules to be expressed very cleanly, while still allowing ATN-type augmentation through the addition of arbitrary tests on the contents of the variables.

Pereira and Warren argue that the DCG formalism is well suited for specifying a formal description of a language and also for use with a parser. In particular, they assert that it is a significant advance over an ATN approach on both philosophical and practical grounds. Their chief claims are that:

1. DCGs provide a common formalism for theoretical work in Computational Linguistics and for writing efficient natural language processors.

2. The rule based nature of a DCG result in systems of greater clarity and modularity.

3. DCG's provide greater freedom in the range of structures that can be built in the course of analyzing a constituent. In particular the DCG formalism makes it easy to create structures that do not follow the structure implied by

the rules of a constituent and easy to create a structure for a constituent that depends on items not yet encountered in the sentence.

The first two points have been discussed in the past whenever the ATN formalism is compared with a rule-based grammar (see [Pratt] , [Heidorn] , [Codd] , or [Bates]). The outcome of such discussions vary. It is safe to say that how one feels about these points depends quite heavily on past experience in using the two formalisms.

We find the third point to be well founded, however. It is clear that the DCG differs most from previous rule-based parsing systems in its inclusion of Logical variables. These result in greater flexibility in building structures to represent constituents that do not follow the inherent structure determined by the rules themselves. They also allow one to create structures which refer to items that have not yet been discovered in the course of analysing the sentence.

We have built an experimental ATN system which can treat ATN registers as Logical variables and, we feel, capture these important strengths offered by the DCG formalism in the otherwise standard ATN formalism.

The second section gives a more detailed desciption of DCG's and presents a simple grammar. In the third section we show an ATN grammar which is "equivalent" to the DCG grammar and discuss the source of its awkwardness. The fourth section then presents an ATN formalism extended to include viewing ATN registers as Logical variables which are subject to the standard unification operation. The final section concludes this note and suggests that logical variables might be fruitfully introduced into other parsing algorithms and systems.

2. Definite Clause Grammars

Figure 1 shows a simple DCG grammar adapted from [Pereira] . Figure 2 gives a sentence in the language recognized by this grammar together with the associated surface syntactic structure and the semantic structure built by the grammar.

The way in which unification produces the appropriate bindings for this example is actually quite subtle, and requires a detailed analysis of the parse, as represented by the refutation graph in Figure 3. For the the refutation graph the Prolog clauses have been put into clausal normal form. Some liberties have been taken with the ordering of the predicates in the interest of compactness.

In trying to establish the "s(P)" goal, the "np(X,P1,P)" is first attempted. The "P1" is an empty variable that is a "place-holder" for predicate information that will come from the verb. It will "hold" a place in the sentence structure that will be provided by the determiner. "P" is destined to contain the sentence structure. The

Fig. 1. A Definite Clause Grammar

```
s(P) -> np(X, P1, P),  vp(X, P1).

np(X, P1, P) -> det(X, P2, P1, P),
                n(X, P3),
                relclause(X, P3, P2).

np(X, P, P) -> name(X).

vp(X, P) -> transv(X, Y, P1),  np(Y, P1, P).
vp(X, P) -> intransv(X, P).

relclause(X, P1, (And P1 P2)) -> [that], vp(X, P2).
relclause(X, P, P) -> [].

det(X, P1, P2, (ForAll X (=> P1 P2))) -> [every].
det(X, P1, P2, (ForSome X (And P1 P2))) -> [a].

n(X, (man X)) -> [man].
n(X, (woman X)) -> [woman].
n(X, (dog X)) -> [dog].

name(john) -> [john]
name(mary) -> [mary]
name(fido) -> [fido]

transv(X, Y, (loves X Y)) -> [loves].
transv(X, Y, (breathes X Y)) -> [breathes].

intransv(X, (loves X) -> [loves].
intransv(X, (lives X) -> [lives].
intransv(X, (breathes X) -> [breathes].
```

Fig. 2. A Sentence, Structure and Representation

SENTENCE

"john loves every woman who breathes"

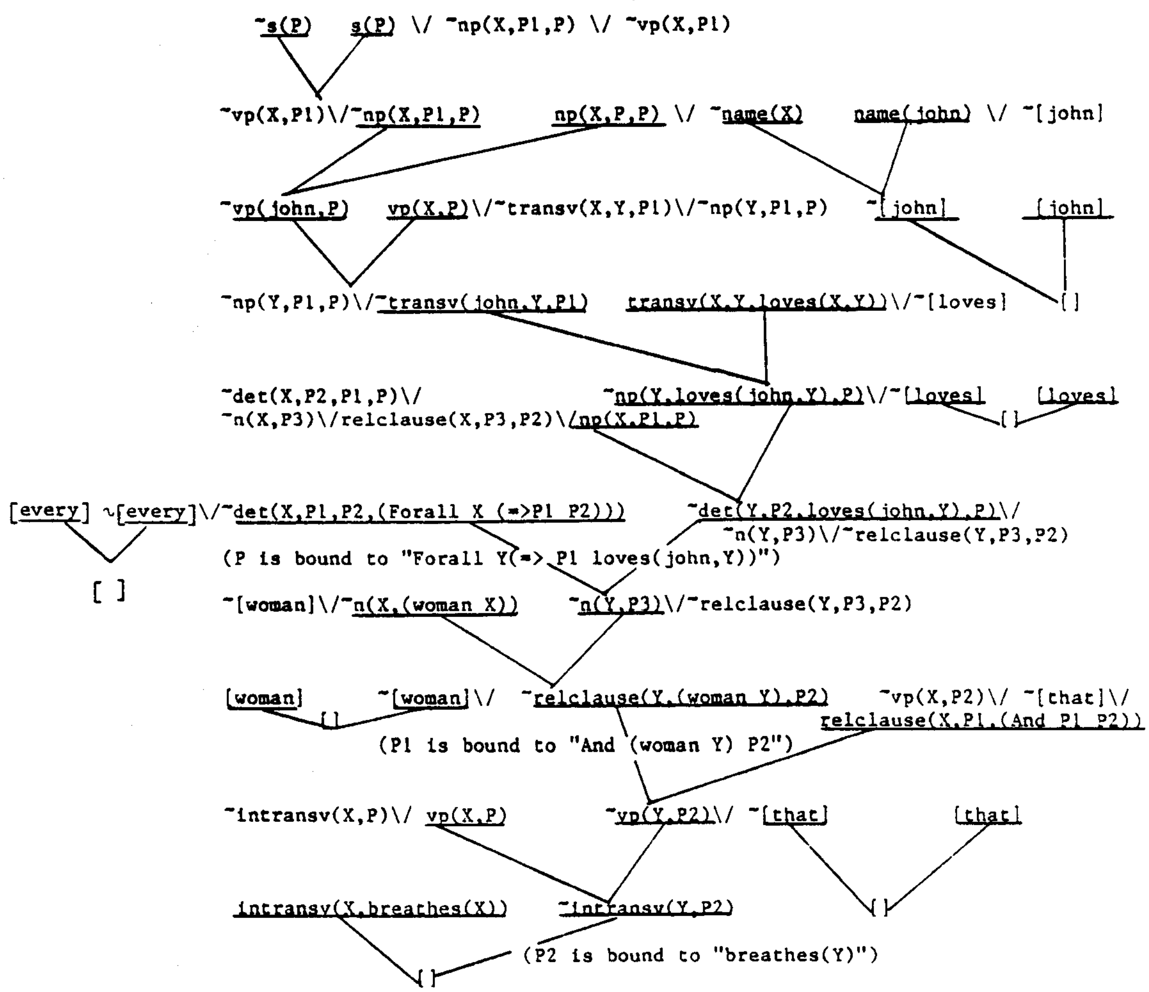
SYNTACTIC STRUCTURE

```
(S (NP (NAME john))
   (VP (TRANSV loves)
       (NP (DET every)
           (NOUN woman)
           (REL (VP (INTRANSV breathes)))))))
```

SEMANTIC REPRESENTATION

```
(ForAll X1 (=> (And (woman X1) (breathes X1))
               (loves john X1)))
```

Fig. 3. Refutation Graph



first "np" clause will be matched, but it will eventually fail since no determiner is present. The second "np" clause will succeed, having forever identified the contents of "P1" with the contents of "P, " whatever they may be. Since there is no determiner in the first noun phrase, there is no quantification information. The quantificational structure must be supplied by the verb phrase, so the structure for the sentence will be the same as the structure for the verb phrase. The variable "X" will be bound to "John".

In trying to establish "vp(John,P1), " the first "vp" clause will succeed, since "loves" is a transitive verb. It is important not to get the variables confused. Within the "vp" clause our original "P1" has been renamed "P" and and we have a new "P1" variable that will be instantiated to "(loves John Y)" by the success of the "transv" goal. The "Y" is as yet undetermined, but we can see that it will be supplied by the next "np(Y,(loves John Y),P)" goal. It shows great foresight on "transv's" part to pass back a variable in such a way that it will correspond to a variable that has already been named. This pattern is repeated throughout the grammar, with powerfull repurcussions. It is even clearer in the success of the "np(Y,(loves John Y),P)" goal, where the presence of the determiner "every" causes "P" to be bound to

```
(Forall Y (=> P1 (loves John Y))
```

This "P" is of course the "P" mentioned above which has been waiting for the verb phrase to supply it with a quantificational structure.

As the relative clause for this "np" is processed, the "P1" embedded in this structure, (our second new P1!), is eventually bound to "(And (woman Y) (breathes Y))" giving us the full structure:

```
(Forall Y (=> (And (woman Y) (breathes Y))
              (loves John Y)))
```

This is what is returned as the binding to the first "P1" in the original "vp(X,P1)" goal. Since our "np(X,P1,P)" goal identified "P" with "P1, " our "s(P)" goal succeeds with the binding of

```
(Forall Y (=> (And (woman Y) (breathes Y))
              (loves John Y)))
```

for "P" - the final structure built for the sentence.

In following the execution of this grammar it becomes clear that very strong predictions are made about which parts of the parse will be supplying particular types of information. Determiners will provide the quantifiers for the propositional structure of the sentence, the first noun phrase and the noun phrase following the verb will be the two participants in the predicate implied by the verb, etc. Obviously this is a simple grammar, but the power of the logical variables can only be made use of through the encoding of these strong linguistic assumptions. DCG's seem to provide a mechanism well qualified for expressing such assumptions and then executing them. Coming up with the assumptions in the first place is, of course, something of a major task in itself.

3. Comparing DC and ATN Grammars

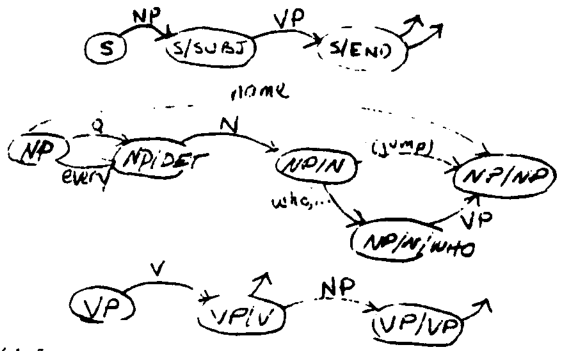
Figure 4 shows an ATN grammar which is the "equivalent" of the DCG grammar given in Figure 1. The format used to specify the grammar is the one described in [finin1] and [finin2] . There are only two minor ways that this particular formalism differs from the standard ATN formalism described in [Woods70] or [Bates] . First, the dollar sign character (i.e. $) followed by the name of a register stands for the contents of that register. Second, the function DEFATN defines a set of arcs, each of which is represented by a list whose first element is the name of the state and whose remaining elements are the arcs emanating from the state.

In addition, this example uses a very simple lexical manager in which a word has (1) a set of syntactic categories to which it belongs (2) an optional set of features and (3) an optional root form for the word. These attributes are associated with a word using the function LEX, which supplies appropriate default values for unspecified arguments.

In the standard ATN model, a PUSH arc invokes a sub-computation which takes no arguments and, if successful, returns a single value. One can achieve the effect of passing parameters to a sub-computation by giving a register an initial value via a SENDR register setting action. There are two methods by which one can achieve the effect of returning more than one value from a sub-computation. The values to be returned can be packaged into a list or the LIFTR register setting action can be used to directly set values in the higher level computation. This grammar makes use of SENDR and LIFTR to pass parameters into and out of ATN computations and thus the actions of the DCG example.

Consider what must happen when looking for a noun phrase. The representation for a NP will be a predicate if the noun phrase is indefinite (i.e. "a man" becomes (man X)) or a constant if the noun phrase is a name (i.e. "john" becomes john). In this simple language, a NP is dominated by a either a sentence (if it is the subject) or by a verb phrase (if it is the object). In either case, the NP also determines, or must agree with, the overall structure used to represent the dominating constituent. If the NP is a simple name, then it exerts no additional influence on the representation of its dominator. If the NP is not a name, then it is indefinite and will eventually result in a quantified expression for the dominating sentence or verb phrase. In this case we need to tell the dominating computation what the predicate, quantifier, connective, and variable name must be. In this ATN grammar, this is done by having the NP network return a value to represent the NP predicate and lift values for the quantifier, connective and variable name.

Fig. 4. An Equivalent ATN Grammar




```
(defatn

 (s (push np t (setr subj *) (to s/subj)))

 (s/subj (push vp t (setr vp *)
              (sendr subjvar $var) (to s/end)))

 (s/end (pop (list $quant $var
                  (list $connect $subj $vp)) $subj)
       (pop $vp (null $subj)))

 (np (wrd a t (liftr quant 'ForSome)
              (liftr connect 'And)(to np/det))
     (wrd every t (liftr quant 'ForAll)
                  (liftr connect '=>)(to np/det))
     (cat name t (setr var *) (to np/np)))

 (np/det (cat n t (setr var (gensym))
              (setr n (list * $var)) (to np/n)))

 (np/n (wrd (who that which) t (to np/n/who))
       (jump np/np t))

 (np/np (pop $n t (liftr var)))

 (np/n/who
  (push vp t (sendr subjvar $var)
             (setr n (list 'And $n *)) (to np/np)))

 (vp (cat v t (setr v *) (to vp/v)))

 (vp/v (push np (getf trans $v) (setr obj *)
             (setr objvar $var) (to vp/vp))
       (pop (list $v $subjvar) (getf intrans $v)))

 (vp/vp (pop (list $quant $objvar
                  (list $connect $obj
                        (list $v $subjvar $objvar)))
             $obj)
        (pop (list $v $subjvar $objvar) (null $obj))))

; (lex <word> <category> <features> <rootform>)

(lex man n)
(lex woman n)
(lex loves v (intrans trans))
(lex breathes v (intrans trans))
(lex lives v (intrans))
(lex john name)
(lex mary name)
(lex fido name)
```

Similarly, when we are looking for a verb phrase, we must know what token (i.e. variable name or constant) represents the subject (if the verb phrase is dominated by a S) or the head noun (if the verb phrase acts as a relative clause). This is done by **sending** the **subjvar** register in the sub-computation the appropriate value via the SENDR function. The techniques used to quantification and build an overall sentence structure in this ATN grammar are similar to those used in the BBN Lunar Grammar [Woods72] .

This heavy use of SENDR and LIFTR to communicate between levels in the grammar makes the ATN grammar cumbersome and difficult to understand. In the next secton we investigate treating ATN registers as logic variables and providing a unification operation on them.

4. Replacing ATN Registers with ATN Variables

Although the previous ATN grammar does the job, it is clearly awkward. We can achieve much of the elegance of the DCG example by treating the ATN registers as **logical variables** and including a unification operation on them. We will call such registers <u>ATN Variables</u>.

Since our ATN variables must not be tampered with between unifications, assignment operations such as SETR, LIFTR and SENDR are precluded. Thus the only operations on ATN Registers are **access** and **unify**. It is possible to provide operations similar to the standard SENDR and LIFTR by defining unification operations which do the unification in the another environment, but we have not explored these possibilities.

The scheduler component of the ATN parser has been modified to be sensitive to the success or failure of attempted unifications. If a unification operation on an arc fails, the arc is blocked and may not be taken.

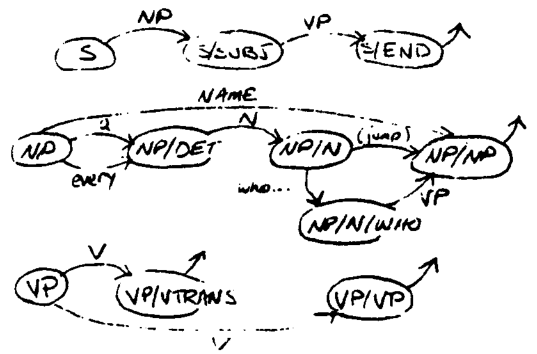
Figure 5 shows a grammar in the extended ATN formalism. A symbol preceded by a "$" represents an ATN Variable and "*" will again stand for the current constituent. Thus in the state S in the grammar:

```
(S (PUSH NP (UNIFY '($SUBJVAR $VP $S) *)
            (TO S/SUBJ)))
```

the parser pushes to the state NP to parse a noun phrase. If one is found, it will pop back with a value which will then be unified with the expression ($SUBJVAR $VP $S). If this unification is successful, the parser will advance to state S/SUBJ. If it fails, the arc is blocked causing the parser to backtrack into the NP network.

Although our grammar succeeds in mimicking the behaviour of the DCG, there are some open questions involving the use of unification in parsing natural languages. An examination of this ATN grammar shows that we are really using unification as a method of passing parameters. The full power of unification is not needed in this example since the

Fig. 5. An Equivalent ATN Grammar with ATN Variables




```
(defatn

 (s (push np (unify '($subjvar $vp $s) *)
       (to s/subj)))

 (s/subj (push vp t (unify '$vp *) (to s/s)))

 (s/s (pop $s t))

 (np (wrd a t (unify
              '$np
              '(ForSome $var (And $pred $hole)))
             (to np/det))
     (wrd every t (unify
                   '$np
                   '(ForAll $var (=> $pred $hole)))
                  (to np/det))
     (cat name t (unify '$np '$hole)
                 (unify '$var *)
                 (to np/np)))

 (np/det (cat n t (unify '$var (gensym))
                  (unify '$pred '(* $var))
                  (to np/n)))

 (np/n (wrd (who that which) t (to np/n/who))
      (jump np/np t))

 (np/np (pop (list $var '$hole $np) t ))

 (np/n/who
  (push vp t (unify '$subjvar '$var)
             (unify '$pred '(And $pred *))
             (to np/np)))

 (vp (cat v (getf trans *)
            (unify '$v '(* $subjvar $objvar))
            (to vp/vtrans))
     (cat v (getf intrans *)
            (unify '$v '(* $subjvar))
            (to vp/vp)))

 (vp/vtrans (push np t (unify '($objvar $v $vp) *)
                       (to vp/vp)))

 (vp/vp (pop $vp t))
```

grammar does not try to find "most-general unifiers" for complicated sets of terms. Most of the time it is simply using unification to bind a variable to the contents of another variable. The most sophisticated use involves binding a variable in a term to another copy of that term which also has a variable to be bound as in the "a man loves a woman" example in Figure 6. But even this binding is a simple one-way application of standard unification. It is not clear to the authors whether this is due to the simple nature of the grammars involved or whether it is an inherent property of the directedness of natural language parsing.

A situation where full unification might be required would arise when one is looking for a constituent matching some partial description. For example, suppose we were working with a syntactic grammar and wanted to look for a singular noun phrase. We might do this with the following PUSH arc:

```
(PUSH NP T (UNIFY * '(NP (DET $DET)
                         (NUMBER SINGULAR)
                         (ADJ $ADJS) ...))
           ...)
```

If we follow the usual schedule of interpreting ATN grammars the unification will not occur until the NP network has found a noun phrase and popped back with a value. This would require a fully symmetric unification operation since there are variables being bound to values in both arguments. It is also highly inefficient since we may know right away that the noun phrase in the input is not singular. What we would like is to be able to do the unification just after the push is done, which would more closely parallel a Prolog-based DCG parse. Then an attempt to "unify" the number register with anything other than singular will fail immediately.

This could be done automatically if we constrain a network to have only one state which does a pop and place some additional constraints on the forms that can be used as values to be popped. Although we have not explored this idea at any length, it appears to lead to some interesting possibilities.

5. Conclusions

We have found the use of logical variables and unification to be a powerful technique in parsing natural language. It is one of the main sources of the strengths of the Definite Clause Grammar formalism. In attempting to capture this technique for an ATN grammar we have come to several interesting conclusions. First, the strength of the DCG comes as much from the skillful encoding of linguistic assumptions about the eventual outcome of the parse as from the powerful tools it relies on. Second, the notion of logical variables (with unification) can be adapted to parsing systems ouside of the theorem proving paradigm. We have successfully adapted these techniques to an ATN parser and are beginning to embed them in an existing parallel bottom-up parser [finin3] . Third, the full power of unification may not be necessary to successfully use logical variables in natural lanuage parsers.

Fig. 6. Example Parses with the ATN Grammar

"john loves every woman who breathes"

```
(ForAll X1 (=> (And (woman X1) (breathes X1))
               (loves john X1)))
```

"john loves a woman"

```
(ForSome X1 (And (woman X1) (loves john X1)))
```

"a man loves a woman"

```
(ForSome X1
         (And (man X1)
              (ForSome X2 (And (woman X2)
                               (loves X1 X2)))))
```

"every man who lives loves"

```
(ForAll X1 (=> (And (man X1) (lives X1))
               (loves X1)))
```

"every man who loves mary loves a woman who loves john"

```
(ForAll X1
        (=> (And (man X1) (loves X1 mary))
            (ForSome X2 (And (And (woman X2)
                                  (loves X2 John))
                             (loves X1 X2)))))
```

"every man who loves a woman who loves every dog loves every dog"

```
(ForAll X1
  (=> (And (man X1)
           (ForSome X2
             (And (And (woman X2)
                       (ForAll X3
                         (=> (dog X3)
                             (loves X2 X3))))
                  (loves X1 X2))))
      (ForAll X4
        (=> (dog X4) (loves X1 X4)))))
```

6. References

1. Bates, M., Theory and Practice of Augmented Transition Network Grammars, in Natural Language Communication with Computers, L. Bolc (Ed.), Springer-Verlag, 1978.

2. Bossie, S., "A Tactical Component for Text Generation: Sentence Generation Using a Functional Grammar", report MS-CIS-1982-26, Computer and Information Science, University of Pennsylvania, 1982.

3. Codd, E. F., Arnold, R.S., Cadiou, J-M., Chang, C. L. and Roussopoulos, N., RENDEZVOUS Version 1: An Experimental English-Language Query Formulation System for Casual Users of Relational Data Bases, Report RJ2144, IBM Research Laboratory, San Jose, January 1978

4. Colmerauer, A., "Metamorphosis Grammars", in L. Bolc (Ed.), Natural Language Communication with Computers, Springer-Verlag, 1978.

5. Finin, T., An Interpreter and Compiler for Augmented Transition Networks, Coordinated Science Laboratory technical report T-48, University of Illinois, 1977.

6. Finin, T., Parsing with ATN Grammars; to appear in Leonard Bolc (ed.) Data Base Question Answering Systems, Springer-Verlag, Berlin, 1982.

7. Finin, T. and B. L. Webber, BUP - A Bottom Up Parser, report MS-CIS-1982-27, Computer and Information Science, University of Pennsylvania, 1982.

8. Heidorn, G., Augmented Phrase Structure Grammar, TINLAP-1, 1975.

9. Kay, M., "Functional Grammar", Proceedings of the Fifth Annual Meeting of the Berkeley Linguistic Society, 1979.

10. Pratt, V. "LINGOL, A Progress Report", IJCAI 4, 1975.

11. Pereira, F. and D. Warren, "Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks"., Artificial Intelligence 13 (1980), 231-278.

12. Winograd, T., Language as a Cognitive Process, Addison-Wesley Publishing Co., Inc, 1983, 349-351.

13. Woods, W., Transition Network Grammars f Natural Language Analysis, CACM 13:10, 1970.

14. Woods, W. A., R. M. Kaplan and B. L. Webbe , "The Lunar Sciences Natural Language Informatio System: Final Report", BBN report 2378, 1972.