LIST AUTOMATA WITH SYNTACTICALLY STRUCTURED OUTPUT

karel OLİVA and Martin PLATEK
Faculty of Mathematics and Physics
Charles University
Malostranské náměstí 25
CS-118 00 Praha 1 - Malá Strana
Czechoslovakia

Abstract:
A new type of abstract automaton is introduced, and both formal and linguistic implications are discussed, most importantly a new possibility of proving certain formal properties of (natural) languages and their grammars (such as context-freeness) and of refinement of the Chomsky hierarchy.

1. Introduction

In this article we want to propose a new type of (abstract) nondeterministic automaton; its most distinguishing feature is that its input data is a linear doubly linked list and its output is a syntactic structure, on condition that the computation was successful, i.e. the word represented by the input list was in the language defined by the automaton, all nondeterministic decisions of the automaton were correct (the automaton "guessed" what to do) and, hence, the computation finished in an accepting configuration.

Apart from other features, this automaton gives a uniform formal environment for the formulation of formal syntax of natural language(s), regardless of the intuitions standing behind the linguistic theory in question; here, we have in mind first of all the dependency or immediate constituent approach to language description.

The intuition standing behind the dependency approach is based on erasing words from the sentence and studying whether the resulting string is grammatical: by means of this, the relative mutual importance of words (i.e., dependency, as the relation between the syntactically "more important" word, governor, and the "less important", dependent, word) can be stepwise determined and then expressed e.g. in a dependency tree of the sentence. Clearly, in more complex cases, it is impossible to subsume all these relations in a sentence purely by means of dependency, since there are also other relations to be found between words (such as coordination or apposition), as well as it is impossible to express all possible relations of dependency in the form of a tree, because in certain cases a single dependent word might have more than one governor (e.g., in cases of words depending on coordination of governors).

On the other hand, the intuitions standing behind the imediate constituent approach is that of replacing certain groups of words by others, and, again, studying the grammaticality of the result. By means of this process, the sentences can be stepwise splitted to smaller and smaller parts from which they are built of, and the structure thus obtained can be then expressed in an IC-tree.

In fact, we believe that both these intuitions are extremely insightful and that it is a regretful misunderstanding that they are still felt as oppositions rather than complementations by many linguists; though there have been several attempts to merge them into a single theory ($\overline{X}$-syntax is surely the most notable case), we are still convinced that the results do not suffice fully. The type of automaton ("acceptor") we propose is in fact able to simulate elegantly any of the two approaches during the process of computation and to reflect them also in the structure of its output. Thus, it makes no distinction between these two linguistic approaches and allows for formulations of theories based on one or the other approach or even on any their mixture.

2. Description of the Automaton

The list automaton consists of a finite control unit attached to a (finite) linear list by a head. The head is always able to read or write symbols to the item of the list on which it stands (the current item) and, in addition, it is able to read (but not to write) the symbol on the item immediatelly to the left in the list. Every item of the list (and, generally, any node in the resulting syntactic structure) consists of a set of pointers L, R, C, CH, O, H, ZL and CP and information parts Cat and Lex (see fig 1.).

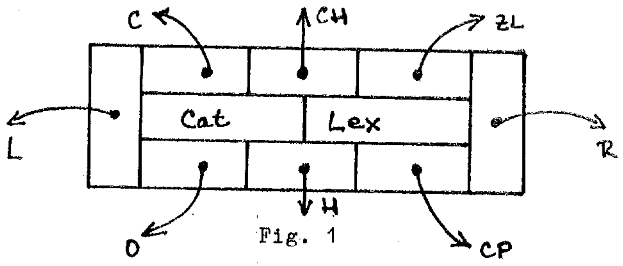

Fig. 1

The pointers serve the following purposes:
C...serves for "simple" coordination (such as "Peter and Paul")
CH..serves for embedded coordination (such as "Paul and Mary and John and Eve played tennis.")
L...at the beginning, this pointer (together with R) serves for connecting the items of the list (see fig 2. for clarification); after the computation is successfully finished, L points to the item on the left edge of the interval of items on which the current item depends (as in "John and Paul who...", see fig. 3)
R...is a pointer analogical to L; serves for connecting the items of the list initially, and after the computation, R points to the item on the right edge of the interval on which the current item depends (see fig. 3)

O...at the beginning, the value of this pointer equals to L; however, it does not change during the whole computation and, hence, keeps the information about the input order of the items in the input list

H...during and after the computation, the value of this pointer is the "head" (in the sense used in $\bar{X}$-syntax) of the phrase represented by the daughter nodes of the current node (current item) in the syntactic (sub)tree

ZL,CP..serve as auxiliary pointers in processing complicated syntactic constructions (coordinations, non-projective constructions)

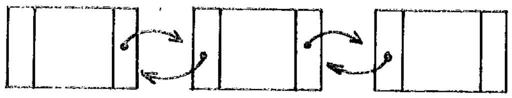

Fig. 2

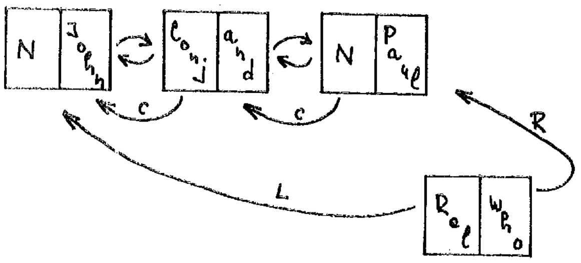

Fig. 3

Further, let P be set of pointers $\{C, CH, L, R, O, H, ZL, ZP\}$,and, in addition, let THIS be a special pointer the value of which is always the current item. Let NIL be a special "empty" value of a pointer. Then, we define the following <u>basic operations</u> of the automaton :

DEL(x,y)...for $x \in P, y \in P \cup \{THIS\}$; this operation takes the item which is the value of y and sets the value of its pointer x to NIL

CON(x,y,z)...for $x \in P$, $y, z \in P \cup \{THIS\}$; performing this operation means setting the pointer x of the item y to the value z

GO(x)...for $x \in P$; the head of the automaton moves from the current item to the item which is the value of x. If x=THIS or x=NIL the state of the control unit becomes undefined (i.e., an error occurs)

NEW(x)...for $x \in P$; a new item is created and the value of x is set to this new item. All pointer values of the new item are set to NIL, the information part of the item is copied from the current item

WRITE(i)...i is a symbol from the alphabet of the automaton; the value of Cat of the current item is set to i

All operations are performed relatively to the current item (i.e., "x" in their description means "x of the current item"). Their intuitive sense is reshaping the input list to a more complex structure by means of setting and changing the values of pointers.

Further, these basic operations can be combined to <u>complex operations</u>. For the purposes of description of Czech syntax, we defined complex operations of the following types (again, the pointers are those of the current item):

```
  I. GO(R)

 II. WRITE(a)

III. CON(R,L,R)
     CON(L,R,L)
     DEL(R,THIS)
     GO(L)

 IV. GO(L)
     CON(R,L,R)
     CON(L,R,L)
     DEL(L,THIS)
     GO(R)

  V. NEW(H)
     CON(R,L,H)
     CON(R,H,R)
     CON(L,H,L)
     CON(L,R,H)
     DEL(R,THIS)
     DEL(H,THIS)
     GO(L)
```

and other eight complex operations serving for processing coordination, non-projective constructions etc. The automaton performs each of these operations in one step of the computation; the next operation to be performed is chosen according to the current internal state of the control unit and the information read by the head (i.e., information contained in the current item and its left neighbour in the list). Performing one step of the computation means performing one of the complex operations and, possibly, changing the internal state of the control unit, both according to the transition function of the automaton.

The set of complex operations introduced has two important features: first, with the help of this set, we are convinced, it is possible to describe sufficiently complete surface syntax of Czech. Second, the set of complex operations of the automaton we use for the description of Czech syntax guarantees that any language accepted by the automaton with these operations is <u>context-free</u>. This point probably deserves further discussion: the matter is that by changing the set of basic operations (i.e., by adding some new basic operations and/or by removing the current ones) and/or by limiting the choice and ordering of basic operations in an appropriate way and/or by limiting the number of "visits" of the head on an item of the list, it is possible to characterize the explicative power of different subtypes of the automaton and, hence, to characterize different types of grammars <u>strongly equivalent</u> with the automaton in question. Thus, e.g., categorial grammars can be shown to be strongly equivalent with automata with operations I-IV and with the number of "visits" limited by constant; context-free grammars are strongly equivalent with automata with complex operations I-III and V and constant number of visits, generalized dependency grammars (this term suspiciously resembles the title of (Gazdar,Klein Pullum and Sag,85), but was in fact introduced as early as in (Gladkij,73)) are strongly equivalent with the automaton with operations I-IV etc. For automata using complex operations different from I-V we have not find any strongly equivalent type of grammars in literature. But probably the most important point concerns weak equivalence: any automaton using the complex operations defined is weakly equivalent to some context-free grammar. (And extending this weak generative capacity will be possible only on condition of adding some new complex operation(s).)

3. Conclusions

The type of automaton introduced is, in our opinion, important for several reasons.

First, it allows for stepwise refinement of the set of its complex operations: first, only an acceptor might be constructed, and only later its operations can be augmented to a real parser. Of course, the augmentation of the primary acceptor and turning it into the parser might be performed in most different ways, which allows for incarnating various linguistic theories over the initial acceptor. Generally, we can start the process of creating the automaton by constructing the complex operations from basic operations DEL and GO only, applying these two basic operations to the pointers L,R and THIS solely (i.e., only pointers from the input list). During its computation, such an acceptor will simulate the derivation of the input string (string represented by the input list). In the second step, i.e. in building the parser, we augment these primitive complex operations by adding other basic operations and/or using other pointers, to get, eventually, the intended parser.

Second, from the linguistic viewpoint, it enables to construct a recognizing automaton - a full syntactic parser (i.e., an automaton which gives a syntactic structure as its output) - which, in addition, allows to prove the context-freeness of the processed languages, but on grounds profoundly different than those of (Gazdar,Klein,Pullum and Sag,85).

Third, from the formal viewpoint, it allows to describe the whole Chomsky hierarchy of languages by a single abstract automaton with differently limited set of operations rather than with a whole set of relatively unrelated types of machines (Turing machine, linearly bounded automaton, pushdown automaton, finite automaton): this is because the operations of the proposed automaton are in fact just refined operations of the list automaton proposed in (Chytil, Plátek and Vogel,86).

References:

Chytil M.P., Plátek M. and Vogel J.: A note on the Chomsky hierarchy, Bulletin of EATCS 28, 1986

Gazdar G., Klein E., Pullum G. and Sag I.: Generalized Phrase Structure Grammar, Basil Blackwell, Oxford, 1985

Gladkij A.V.: Formalnyje grammatiki i jazyki, Mir, Moscow, 1973

Plátek M. and Vogel J.: Deterministic list automata and erasing graphs, in The Prague Bulletin of Mathematical Linguistics 45, Prague, 1986