TWO-WAY FINITE AUTOMATA AND DEPENDENCY GRAMMAR: A PARSING METHOD FOR INFLECTIONAL FREE WORD ORDER LANGUAGES[1]

Esa Nelimarkka, Harri Jäppinen and Aarno Lehtola
Helsinki University of Technology
Helsinki, Finland

ABSTRACT

This paper presents a parser of an inflectional free word order language, namely Finnish. Two-way finite automata are used to specify a functional dependency grammar and to actually parse Finnish sentences. Each automaton gives a functional description of a dependency structure within a constituent. Dynamic local control of the parser is realized by augmenting the automata with simple operations to make the automata, associated with the words of an input sentence, activate one another.

I INTRODUCTION

This paper introduces a computational model for the description and analysis of an inflectional free word order language, namely Finnish. We argue that such a language can be conveniently described in the framework of a functional dependency grammar which uses formally defined syntactic functions to specify dependency structures and deep case relations to introduce semantics into syntax. We show how such a functional grammar can be compactly and efficiently modelled with finite two-way automata which recognize the dependants of a word in various syntactic functions on its both sides and build corresponding dependency structures.

The automata along with formal descriptions of the functions define the grammar. The functional structure specifications are augmented with simple control instructions so that the automata associated with the words of an input sentence actually parse the sentence. This gives a strategy of local decisions resulting in a strongly data driven left-to-right and bottom-up parse.

A parser based on this model is being implemented as a component of a Finnish natural language data base interface where it follows a separate morphological analyzer. Hence, throughout the paper we assume that all relevant morphological and lexical information has already been extracted and is computationally available for the parser.

Although we focus on Finnish we feel that the model and its specification formalism might be applicable to other inflectional free word order languages as well.

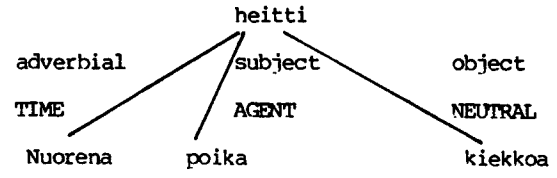
II LINGUISTIC MOTIVATION

There are certain features of Finnish which suggest us to prefer dependency grammar to pure phrase structure grammars as a linguistic foundation of our model.

Firstly, Finnish is a "free word order" language in the sense that the order of the main constituents of a sentence is relatively free. Variations in word order configurations convey thematical and discursional information. Hence, the parser must be ready to meet sentences with variant word orders. A computational model should acknowledge this characteristic and cope efficiently with it. This demands a structure within which word order variations can be conveniently described. An important case in point is to avoid structural discontinuities and holes caused by transformations.

We argue that a functional dependency-constituency structure induced by a dependency grammar meets the requirements. This structure consists of part-of-whole relations of constituents and labelled binary dependency relations between the regent and its dependants within a constituent. The labels are pairs which express syntactic functions and their semantic interpretations.

For example, the sentence "Nuorena poika heitti kiekkoa" ("As young, the boy used to throw the discus") has the structure



or, equivalently, the linearized structure

((Nuorena)$_{\text{advl}}$ (poika)$_{\text{subj}}$ heitti (kiekkoa)$_{\text{obj}}$),
TIME AGENT NEUTRAL

---

1 This research is supported by SITRA (Finnish National Fund for Research and Development).

where parentheses indicate constituent boundaries and, within each constituent, the word without parentheses is the head. To be more precise, an inflected word appears as a complex of its syntactic, morphological and semantic properties. Hence, our sentence structure is a labelled tree whose nodes are complex expressions.

The advantage of the functional dependency structures lies in the fact that many word order varying transformations can be described as permutations of the head and its labelled dependants in a constituent. Reducing the depth of structures (e.g. by having a verb and its subject, object, adverbials on the same level) we bypass many discontinuities that would otherwise appear in a deeper structure as a result of certain transformations. As an example we have the permutations

((Poika)$_{subj}$ heitti (kiekkoa)$_{obj}$ (nuorena)$_{advl}$)

(Heittikö (poika)$_{subj}$ (nuorena)$_{advl}$ (kiekkoa)$_{obj}$)

and

((Kiekkoako)$_{obj}$ (poika)$_{subj}$ heitti (nuorena)$_{advl}$).

("The boy used to throw the discus when he was young", "Did the boy use to throw...?", "Was it discus that the boy used to throw...?", respectively.)

The second argument for our choices is the well acknowledged prominent role of a finite verb in regard to the form and meaning of a sentence. The meaning of a verb includes, for example, knowledge of its deep cases, and the choice of a particular verb to express this meaning determines to a great extent what deep cases are present on the surface level and in what functions. Moreover, due to the relatively free word order of Finnish, the main means of indicating the function of a word in a sentence is the use of surface case suffixes, and very often the actual surface case depends not only on the intended function or role but on the verb as well.

Finally, we wish to describe the sentence analysis as a series of local decisions of the following kind. Suppose we have a sequence $C_1, \ldots, C_{i-1}, C_i, C_{i+1}, \ldots, C_n$ of constituents as a result of earlier steps of the analysis of an input sentence, and assume further that the focus of the analyzer is at the constituent C_i. In such a situation the parser has to decide whether C_i is

(a) a dependant of the left neighbour C_{i-1},

(b) the regent of the left neighbour C_{i-1},

(c) a dependant of some forthcoming right neighbour,

(d) the regent of some forthcoming right neighbour.

Observe that decisions (c) and (d) refer either to a constituent which already exists on the right side of C_i or which will appear there after some steps of the analysis. Further, it should be noticed that we do not want the parser to make any hypothesis of the syntactic or semantic nature of the possible dependency relation in (a) and (c) at this moment.

We claim that a functional combination of dependency grammar and case grammar can be put into a computational form, and that the resulting model efficiently takes advantage of the central role of a constituent head in the actual parsing process by letting the head find its dependants using functional descriptions. We outline in the next sections how we have done this with formally defined functions and 2-way automata.

III FORMALLY DEFINED SYNTACTIC FUNCTIONS

We abstract the restrictions imposed on the head and its dependant in a given subordinate relation. Recall that a constituent consists of the head - a word regarded as a complex of its relevant properties - and of the dependants - from zero to n (sub)constituents.

The traditional parsing categories such as the (deep structure) subject, object, adverbial and adjectival attribute will be modelled as functions

$$f: \mathcal{D}_f \rightarrow \mathcal{C},$$

where $\mathcal{C}$ is the set of constituents and $\mathcal{D}_f \subset \mathcal{C} * \mathcal{C}$ is the domain of the function.

The domain of a function f will be defined with a kind of Boolean expression over predicates which test properties of the arguments, i.e. the regent and the potential dependant. In the analysis this relation is used to recognize and interprete an occurance of a <head,dependant>-pair in the given relation. The actual mapping of such pairs into $\mathcal{C}$ builds the structure corresponding to this function.

For notational and implementational reasons we specify the functions with a conditional expression formalism. A (primitive) conditional expression is either a truth valued predicate which tests properties of a potential constituent head (R) and its dependant (D) and deletes non-matching interpretations of an ambiguous word, or an action which performs one of the basic construction operations such as labelling (:=), attaching (:-), or deletion, and returns a truth value.

Primitive expressions can be written into series (P_1 P_2 ... P_n) or in parallel (P_1;P_2; ...; P_n) to yield complex expressions. Logically, the former corresponds roughly to an and-operation and the latter an or-operation. A conditional operation -> and recursion yield new complex expressions from old ones.

As an example, consider the expressions 'Object', 'RecObj' and 'IntObj' in Figure 1.

```
RELATION: Object
      ((RecObj)(IntObj)) -> (D := Object)(C := D)(DEL D))

RELATION: RecObj
      ((R = +Transitive -Nominal)(D = +Nominal -Sentence)
       -> ((D = Part) -> (D = PL);
                ((D = SG) -> (D = -Countable);
                          ((D = +Countable)(R = ( Neg
                                                  +pobj )))));
          ((D = PersPron Acc)(R = Pos) -> T);
          ((D = ( Noun ))(R = Pos)
               -> ((D = Nom) -> (D = PL);
                          ((D = SG)(R = ( Pass
                                          (Act Impv ( (P 2P )) ) )));
             ((D = Gen SG)(R = Act ( Ind
                                     Cond
                                     Pot
                                     (Impv 3P) )))))

      ((R = +Transitive +Nominal)( D = -Sentence +Nominal)
                -> (D = ( Nom Gen Acc Part)))


RELATION: IntObj
      ((R = ( MovingVerb1 CopuVerb1 )) -> (D := Neutral));
      ((R = CommunicVerb1)(D = +Informative) -> (D := Neutral))
```

Figure 1.

The relation 'RecObj' approximates the syntactic and morphological restrictions imposed on a verb and its nominal object in Finnish. (It represents partly the partitive-accusative opposition of an object, and, for an accusative object, its nominative-genetive distribution.) The relation 'IntObj', on the other hand, tries to interprete the postulated object using semantic features and a subcategorization of verbs with respect to deep case structures and their realizations. The semantic restrictions imposed on the underlying deep cases are checked at this point. 'Object', after a succesful match of these syntactic and semantic conditions, labels the postulated dependant (D) as 'Object' and attaches it to the postulated regent (R).

IV FUNCTIONAL DESCRIPTIONS WITH TWO-WAY AUTOMATA

We introduced the formal functions to define conditions and structures associated with syntactic dependency relations. What is also needed is a description of what dependants a word can have and in what order.

In a free word order language we would face, for example, a paradigm fragment of the form

(subj) V (obj) (advl)

(advl) (subj) V (obj)

V (subj) (obj) (advl)

(obj) (subj) V (advl)

.
.
.

for functional dependency structures of a verb. (Observe that we do not assume transformations to describe the variants.) We combine the descriptions of such a paradigm into a modified two-way finite automaton.

A 2-way finite automaton consists of a set of states, one of which is the initial state and some of which are final states, and of a set of transition arcs between the states. Each arc recognizes a word, changes the state of the automaton and moves the reading head either to the left or right.

We modify this standard notion to recognize left and right dependants of a word starting from its immediate neighbour. Instead of recognizing words (or word categories) these automata recognize functions, i.e. instances of abstract relations between a postulated head and its either neighbour. In addition to a mere recognition the transitions build the structures determined by the observed function, e.g. attach the neighbour as a dependant, label it in agreement with the function and its interpretation.

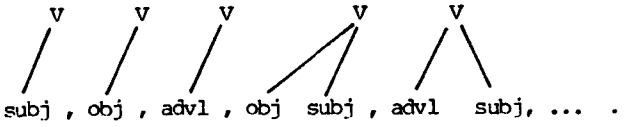
```
STATE: ?V LEFT
     ((D = +Phrase) -> (Subject -> (C := ?VS ));
                       (Object -> (C := ?VO ));
                       (Adverbial -> (C := ?V ));
                       (GenSubj -> (C := VS? ));
                       (SentAdvl -> (C := ?V ));
                       (T =) (C := V? )));
     ((D = -Phrase) -> (C := V? ))

STATE: V? RIGHT
     ((D = +Phrase) -> (Subject -> (C := VS? ));
                       (Object -> (C := VO? ));
                       (SentSubj -> (C := VSentS?));
                       (SentObj -> (C := VSentO? ));
                       (Adverbial -> (C := V? ));
                       (SentAdvl -> (C := VSentA? ));
                       (T -> (C := ?VFinal )));
     ((D = -Phrase) -> (C := V? )(BuildPhraseOn RIGHT))

STATE: ?VS LEFT
     ((D = +Phrase) -> (Object -> (C := ?VSO ));
                       (Adverbial -> (C := ?VS ));
                       (SentAdvl -> (C := VS? ));
                       (T -> (C := VS? )));
     ((D = -Phrase) -> (C := VS? ))
```

Figure 2.

Figure 2. exhibits part of a verb automaton which recognizes and builds, for example, partial structures like



The states are divided into 'left' and 'right' states to indicate the side where the dependant is to be found. Each state indicates the formal functions which are available for a verb in that particular state. A succesfull application of a function transfers the verb into another state to look for further dependants.

Heuristic rules and look-ahead can also be used, For example, the rule

```
((R1 = ', )(R2 = 'että )(C = +Sattr)
   -> (C := N?Sattr) (BuildPhraseOn RIGHT))
```

in the state N? of the noun automaton anticipates an evident forthcoming sentence attribute of, say, a cognitive noun and sets the noun to the state N?Sattr to wait for this sentence.

V PARSING WITH A SEQUENCE OF 2-WAY AUTOMATA

So far we have shown how to associate a 2-way automaton to a word via its syntactic category. This gives a local description of the grammar. With a few simple control instructions these local automata are made to activate each other and, after a sequence of local decisions, actually parse an input sentence.

An unfinished parse of a sentence consists of a sequence $C_1, C_2, \ldots, C_n$ of constituents, which may be complete or incomplete. Each constituent is associated with an automaton which is in some state and reading position. At any time, exactly one of the automata is active and tries to recognize a neighbouring constituent as a dependant.

Most often, only a complete constituent (one featured as '+phrase') qualifies as a potential dependant. To start the completion of an incomplete constituent the control has to be moved to its associated automaton. This is done with a kind of push operation (BuildPhraseOn RIGHT) which deactivates the current automaton and activates the neighbour next to the right (see Figure 2). This decision corresponds to a choice of type (d). A complete constituent in a final state will be labelled as a '+phrase' (along with other relevant labels such as '±sentence', '±nominal', '±main'). Operations (FindRegOn LEFT) and (FindRegOn RIGHT), which correspond to choices (a) and (c), deactivate the current constituent (i.e. the corresponding automaton) and activate the leftmost or rightmost constituent, respectively. Observe that the automata need not remember when and why they were activated. Such simple "local control" we have outlined above yields a strongly data driven bottom-up and left-to-right parsing strategy which has also top-down features as expectations of lacking dependants.

VI DISCUSSION

As we have shown, our parser consists of a collection of finite transition networks which activate each other. The use of 2-way instead of 1-way automata distinguishes our parser from ATN-parsers. (There are also other major differences.) In our dependency oriented model non-terminal categories (S, VP, NP, AP, ...) are not needed, and a constituent is not postulated until its head is found. This feature separates our parser from those which build pure constituent structures without any reference to dependency relations within a constituent. In fact, each word collects actively its dependants to make up a constituent where the word is the head.

A further characteristic of our model is the late postulation of syntactic functions and semantic roles. Constituents are built blindly without any predecided purpose so that the completed constituents do not know why they were built. The function or semantic role of a constituent is not postulated until a neighbour is activated to recognize its own dependants. Thus, a constituent just waits to be chosen into some function so that no registers for functions or roles are needed.

VII REFERENCES

Hudson, R.: Arguments for a Non-transformational Grammar. The University of Chicago Press, 1976.

Hudson, R.: Constituency and Dependency. Linguistics 18, 1980, 179–198.

Jäppinen, H., Nelimarkka, E., Lehtola, A. and Ylilammi, M.: Knowledge engineering approach to morphological analysis. Proc. of the First Conference of the European Chapter of ACL, Pisa, 1983, 49-51.

Lehtola, A.: Compilation and implementation of 2-way tree automata for the parsing of Finnish. Helsinki University of Technology (forthcoming M.Sc. the thesis).

Nelimarkka, E., Jäppinen, H. and Lehtola A.: Dependency oriented parsing of an inflectional language (manuscript).