FINITE-STATE PARSING AND DISAMBIGUATION

Kimmo Koskenniemi

University of Helsinki
Department of General Linguistics
Hallituskatu 11
00100 Helsinki, Finland

ABSTRACT

A language-independent method of finite-state surface syntactic parsing and word-disambiguation is discussed. Input sentences are represented as finite-state networks already containing all possible roles and interpretations of its units. Also syntactic constraint rules are represented as finite-state machines where each constraint excludes certain types of ungrammatical readings. The whole grammar is an intersection of its constraint rules and excludes all ungrammatical possibilities leaving the correct interpretation(s) of the sentence. The method is being tested for Finnish, Swedish and English.

INTRODUCTION

The present approach is surface oriented and shallow, and it does not aim to uncover semantically oriented distinctions. An important source of inspiration has been Fred Karlsson's syntactic parser for Finnish, FPARSE (1985). The present approach tries to formalize the underlying ideas of that parser in a finite-state framework (cf. Karlsson 1989a,b). The finite-state formalism attacks the very basic things in syntax such as: what are the correct readings of ambiguous words, what are the clauses in a complex sentence, how the words form constituents, and what are the syntactic roles of the constituents.

Let us consider the full framework of automatic syntactic parsing. One possible partition of the whole process is given in the following figure 1.

The morphological analysis is done eg. by using the two-level model (Koskenniemi 1983). Comprehensive systems exist now for Finnish, Swedish, English and Russian (about 30-40,000 root entries in each), and some twenty smaller ones.

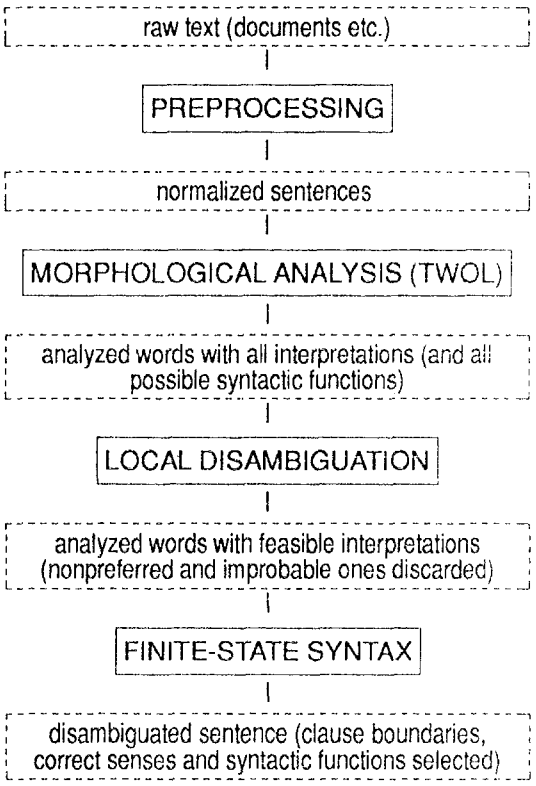

Figure 1.

The local disambiguation is an essential step eg. in Swedish, because many longer word-forms have several possible interpretations. In part, the local disambiguation supplements the two-level description by imposing more sophisticated restrictions on eg. compounds, and by reducing redundant or duplicate analyses (eg. in case a derived word both exists as a given lexicalized entry and is productively generated from its root). The remaining logic concerns weighing various alternatives and excluding readings which are significantly less probable than the best ones.

FINITE-STATE SYNTAX

The actual finite-state syntax consists of three components:

- Syntactic disambiguation of word-forms which have multiple interpretations.
- Determination of clause boundaries.
- Determining the head-modifier relations of words and the surface syntactic functions of the heads.

These components are well defined but they depend on each other in a nontrivial way. It is more convenient to write constraint rules for disambiguation and head-modifier relations if one can assume that the clause boundaries are already there. And conversely, the clause boundaries are easier to determine if we have the correct readings of words available. The approach adopted in this paper shows one solution where one may describe the constraints freely, ie. one may act as if the other modules had already done their work.

Representation of sentences

The way we have chosen in order to solve this interdependence, relies on the representation of sentences and the constraint rules. Each sentence is represented as a finite-state machine (fsm) that accepts all possible **readings** of the sentence. The task of the grammar is to accept the correct reading(s) and exclude incorrect ones. In a reading we include:

- One interpretation of each word-form.
- One possible type of clause boundary or its absence for each word boundary.
- One possible syntactic tag for each word.

An example of a sentence in this representation is given in figure 2 on the next page. In the input sentence each word is represented as an analysis given by the morphological analyzer. The representation consists of one or more interpretations, and each interpretation, in turn, of a base form and a set of morphosyntactic features, eg. "katto" N ELA SG.

Word and clause boundaries

For word boundaries we have four possibilities:

@@ A sentence boundary, which occurs only at the very beginning and end of the sentence (and is the only possibility there).

@ A normal word boundary (where there is no clause boundary).

@/ A clause boundary separating two clauses, where one ends and the other starts.

@< Beginning of a center embedding, where the preceding clause continues after the embedding has been completed.

@> End of a center embedding.

Each word is assumed to belong to exactly one clause. This is taken strictly as a formal basis and implies that words in a subordinate clause only belong to the subordinate clause, not to their main clause. Furthermore, this implies a very flat structure to sentences. Tail recursion is treated as iteration.

There has been a long dispute on the finite-state property of natural languages. We have observed that one level of proper center embedding is fairly common in our corpuses and that these instances also represent normal and unmarked language usage. We do not insist on the absence of a second or third level of center embedding. We only notice that there are very few examples of these in the corpuses, and even these are less clear examples of normal usage.

The present version of the finite-state syntax accepts exactly one level of center embedding. The formalism and the implementation can be extended to handle a fixed number of recursive center-embeddings, but we will not pursue it further here.

Grammatical tags

One grammatical tag is attached with each word. Tags for heads indicate the syntactic role of the constituent, eg. MAIN-PRED, SUBJ, OBJ, ADV, PRED-COMP, and tags for modifiers reflect the part of speech of the head and the direction where it is located, eg. N→, ←N.

This kind of simple tagging induces a kind of a constituent structure to the sentence closely resembling classical parsing.

GRAMMAR

The proposed grammar constructs no analysis for input sentences. Instead, the grammar excludes the incorrect readings. The ultimate result of the parsing is already present as one reading in the initial representation of the sentence which acts as an input to the parser. The result is just hidden among a large number of incorrect readings.

Input sentences

The following is an example of a sentence "kalle voisi uida paljonkin" (English glosses 'Char-

les could swim much+also') to be input to the finite-state syntax:

```
(@@
 "kalle" PROP N NOM SG (/ SUBJ OBJ
                          PRED-COMP)
 (/ @ @< @> @/)
 (/ ("voida" VCHAIN V COND ACT
                     SG3 MAIN-PRED)
    ("voida" VCHAIN V COND ACT NEG))
 (/ @ @< @> @/)
 (/ ("uida" V INF1 NOM)
    ("uida" V PRES PSS NEG))
 (/ @ @< @> @/)
 (/ ("paljon" ADV kin)
    ("paljon" AD-A kin))
@@)
```

Figure 2

This is an expression standing for a finite-state network. Alternatives are denoted by lists of the form:

```
(/ (alternative-1)
   (alternative-2)
   ...)
```

The input expression lists thus some 256 distinct readings in spite of its concise appearance. (The input is here still simplified because of the omission of the syntactic function tags.)

Constraint rules

Each constraint is formulated as a readable statement expressing some necessity in all grammatical sentences, eg.:

```
NEG  ==>  NEGV .. _
```

This constraint says that if we have an occurrence of a feature NEG (denoting a negative form of a verb), then we must also have a feature NEGV (denoting a negation) in the same clause. ".." denotes arbitrary features and stems, excluding clause boundaries except for full embeddings.

Types of constraint rules

Several types of constraint rules are needed:

- Technical constraints for feasible clause bracketing.
- Disambiguation rules (eg. imperatives only in sentence initial positions, negative forms require a negation, AD-A requires an adjective or adverb to follow; etc.)
- Clause boundary constraints (relative pronouns and certain conjunctions are preceded by a boundary, even other boundaries need some explicit clue to justify their possibility).
- Every clause may have at most one finite verb and (roughly speaking) also must have one finite verb.

Examples of constraint rules

The following rule constrains the occurrence infinitives by requiring that they must be preceded by a verb taking an infinitive complement (signalled by the feature VCHAIN).

```
INF1 NOM  ==>  VCHAIN .. _
```

Imperatives should occur only at the beginning of a sentence. A coordination of two or more imperatives is also permitted (if the first imperative is sentence initial):

```
IMPV  ==>
    [@@ | IMPV . @/ . [ COMMA
                      | COORD] ] . _
```

(Here COMMA is a feature associated with the punctuation token, and COORD a feature present in coordinating conjunctions.)

The following disambiguation rule requires that modifiers of adjectives and adverbs must have their head present:

```
AD-A  ==>  _  . @ . [A | ADV]
```

For clause boundaries we need a small set of constraint rules. Part of them specify that in certain contexts (such as before relative pronouns or subjunctions) there must be a boundary. The remaining rules specify converse constraints, ie. what kinds of clues must be present in order for a clause boundary to be present.

All these constraints are ultimately implemented as finite-state machines which discard he corresponding ungrammatical readings. All constraint-automata together leave (hopefully) exactly one grammatical reading, the correct one. The grammar as a whole is logically an intersection of all constraints whereas the process of syntactic analysis corresponds to the intersection of the grammar and the input sentence.

Output

With a very small grammar consisting of about a dozen constraint rules, the input sentence given in the above example is reduced into the following result:

```
(@@ "kalle" PROP N NOM SG SUBJ
 @ "voida" VCHAIN V COND ACT
                     SG3 MAIN-PRED
 @ "uida" V INF1 NOM
 @ "paljon" AD kin
@@)
```

Monotonicity

The formalism and implementation proposed for the finite-state syntax is monotonic in the sense that no information is ever changed. Each constraint simply adds something to the discriminating power of the whole grammar. No constraint rule may ever forbid something that would later on be accepted as an exception.

This, maybe, puts more strain for the grammar writer but gives us better hope of understanding the grammar we write.

IMPLEMENTATION

The constraint rules are implemented by using Ron Kaplan's finite-state package. In the preliminary phase constraints are hand-coded into expressions which are then converted into fsm's. We have planned to construct a compiler which would automatically translate rules in the proposed formalism into automata like the one used for morphological two-level rules (Karttunen et al. 1987).

The actual run-time system needs only a very restricted set of finite-state operations, intersection of the sentence and the grammar. The grammar itself might be represented as one large intersection or as several smaller ones which are intersected in parallel. The sentence as a fsm is of a very restricted class of finite-state networks which simplifies the run-time process.

An alternative and obvious framework for implementing constraint rules is Prolog which would be convenient for the testing phase. Prolog would, perhaps, have certain limitations for the production use of such parsers.

ACKNOWLEDGEMENTS

The work has been funded by the Academy of Finland and it is a part of the activities of the Research Unit for Computational Linguistics at the University of Helsinki. Special thanks are due to Ron Kaplan and Lauri Karttunen for the use of the finite-state package.

REFERENCES

Karttunen, L., K. Koskenniemi, and R. Kaplan 1987. A compiler for two-level phonological rules. In: Dalrymple et al. Tools for morphological analysis. Report No. CSLI-87-108, Center for the Study of Language and Information, Stanford University.

Karlsson, F. 1985. Parsing Finnish in terms of a process grammar. In: Computational Morphosyntax: Report on Research 1981-84. (Ed.) F. Karlsson, University of Helsinki, Department of General Linguistics, Publications, No. 13. pp. 137-176.

--- 1989a. Parsing and constraint grammar. Research Unit for Computational Linguistics, University of Helsinki. Manuscript, 45 pp.

--- 1989b. Constraint grammar as a framework for parsing running text. Paper submitted to Coling'90.

Koskenniemi, K. 1983. Two-level morphology: A general computational model for word-form recognition and production. University of Helsinki, Department of General Linguistics, Publications, No. 11.