– GSLT Course - spring 2005 -
– NLP1 Practical -

WORDS
Building a morphological analyzer

Preben Wik
preben@speech.kth.se

The task for this assignment was to design and implement a morphological analyzer for regular nouns in Swedish, using the techniques of two-level morphology and finite-state transducers. The analyzer should take a word form as input and produce a morphological representation of the word form as output.

For example, if input is flickor
Output should be flicka N UTR INDEF PL NOM

The assignment was divided into two parts

1. designing an FST that correctly maps Swedish noun forms
2. implementing the FST from 1

For me, not being so familiar with the 'grammar aspects' of nlp, the assignment consisted for a large part of considering or 'thinking about ' what we were actually trying to represent.
In the paper describing the assignment some useful information was given.
http://www.svenska.gu.se/~svelb/kurs/nlp05/nlp1morfass1.html
For example:
Swedish regular nouns are traditionally divided into five different declenations, that is there are five ways in which Swedish nouns differ in their endings (suffixes).
-or, -ar, -er/-r, -n, -ø (no suffix)
Most importantly for my solution of the task was the specification of which attributes the analysis should cover. They were as follows:

Word class	always N since the assignment should only deal with nouns
Gender	either Uter (UTR) e.g. 'en stol' or Neuter (NEU) e.g. 'ett bord'
Definiteness	either INDEF (stol - stolar) or DEF (stolen - stolarna)
Number	either SG – singular or PL - plural
Case	either NOM – Nominative form (stolen, stolarna) or GEN – Genitive form (stolens, bordens)

All words should be tagged with these functional attributes, whether there is a visible manifestation on the surface layer or not. So that for example, the Swedish word for table - ‘bord’ has the same surface level in its singular and plural form. That means that the word ‘bord’ is ambiguous and could have two possible analyses.

bord bord N NEU INDEF SG NOM

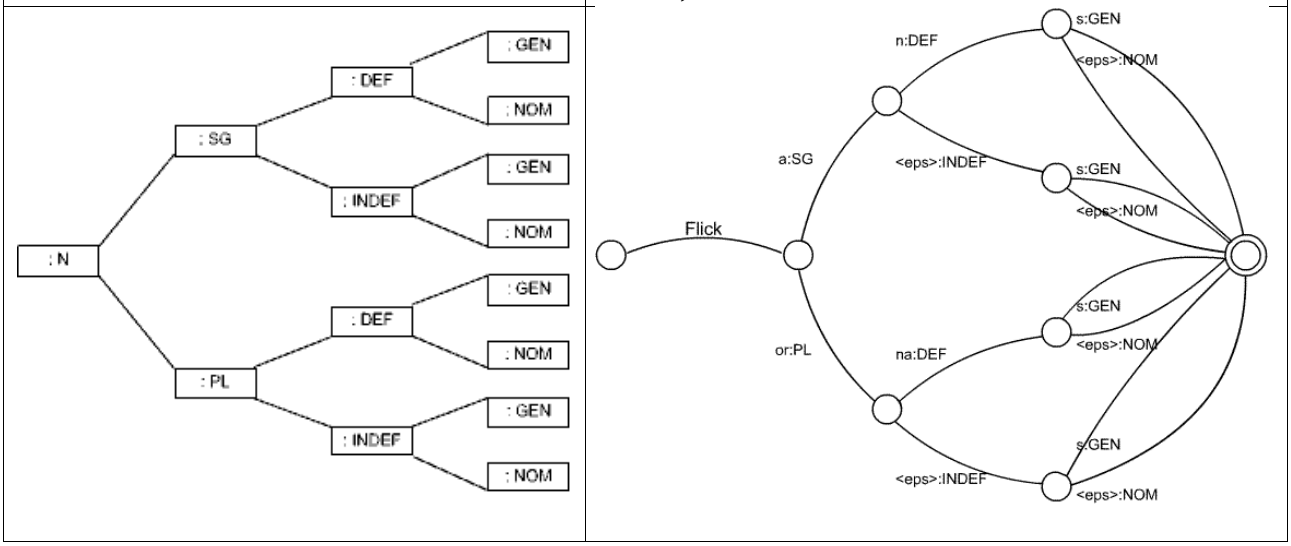
bord bord N NEU INDEF PL NOM

and in the FST both possibilities should be given in the output.

An insight to me was that I could create all the various FSTs by using a table as a template for all the word forms.

Eight combinations of each word are possible – and necessary to fill out.

Example of surface level	Lexical level
flickans	flicka , N UTR SG DEF GEN
Flickan	flicka , N UTR SG DEF NOM
Flickas	flicka , N UTR SG INDEF GEN
Flicka	flicka , N UTR SG INDEF NOM
Flickornas	flicka , N UTR PL DEF GEN
Flickorna	flicka , N UTR PL DEF NOM
Flickors	flicka , N UTR PL INDEF GEN
Flickor	flicka , N UTR PL INDEF NOM



Picture 1 Schematic view of the template used to fill out the word forms (left) and the resulting implementation of ‘flicka’ as an FST (right)

Following this algorithm, I ended up making eight different FSTs, not five – one for each declenation – as I had expected to make.

The word ‘ros’ does not follow the same pattern as the word ‘flicka’ even if they belong to the same declenation. They share the same –or plural suffix, but they behave differently in singular (if the root for flicka is flick +a for SG and +or for PL) and in genitive form singular you do not add an s to ‘ros’.

This is perhaps not the most general, and in that respect 'elegant' solution, so some of the questions I was faced with during the design was:
What characterizes a good FST?

- Efficient?
  - Efficient in what respect? Fast to design of fast to run?
- Small?
  - smallest number of nodes or smallest number of arcs?
  - Is an efficient FST the same as a small FST?
- Beautiful?
  - Some clever optimizations may reduce efficiency and/or increase size, but still be considered more 'elegant'.

I believe my answers were -at least to some extent- based on my choice of tools to use in the assignment – the FSM Library from AT&T. It a set of tools to use in building and running general FSMs. (Finite State Machines) located at http://www.research.att.com/sw/tools/fsm/
On top of this they have also built a toolkit for finite-state linguistic analysis called Lextools: http://www.research.att.com/sw/tools/lextools/ Here it is possible to use higher level linguistic descriptions (e.g. regular expressions) and create weighted finite-state transducer from these.
It is designed to be able to efficiently handle FSMs with millions of nodes (used in speech recognition), and much work has been put into tools for optimizing FSMs for fast processing

The answers to my questions regarding efficiency will on some levels then be resolved by the FSM Library and my main concern should be on fast implementation. The general solution described above will create FSTs with identical number of nodes and arcs, and so a generality is found that holds some redundancy

When creating an FST using the FSMLibrary the FST nodes and arcs are simply described as tab separated rows of numbers. For example, the FST described in Picture1 would look like this:

```
1	2	flick : N [UTR]
2	3	a : [SG]
3	4	n : [DEF]
4	6	s : [GEN]
4	6	[<epsilon>] : [NOM]
3	5	[<epsilon>] : [INDEF]
5	6	[<epsilon>] : [NOM]
5	6	s : [GEN]
2	7	or : [PL]
7	8	na : [DEF]
7	9	[<epsilon>] : [INDEF]
8	6	[<epsilon>] : [NOM]
8	6	s : [GEN]
9	6	[<epsilon>] : [NOM]
9	6	s : [GEN]
6
```

To add a word with the same morphological pattern as 'flicka' (for example klocka) an additional line is simply added like:

```
1        2                klock : N [UTR]
```

The remaining process was divided into several phases.

- Designing an FST for one of the declenations (flicka) as described above in order to test if all attributes are covered.
- Compile the description above (flicka.al) and the lexicon description (.lab .scl) into a binary .fst file

  ```
  lexarclist -l flicka.lab -S flicka.scl -F flicka.fst flicka.al
  ```
- Run this on the FST search program used in the second part of the assignment.

  ```
  lexcompre -l flicka.lab -s "flickan" | fsmcompose - flicka.fst |
  lexfsmstrings -l flicka.lab
  ```
- Make similar FSTs for all the other declenations

- Merge all the smaller FSTs into one large

  ```
  fsmunion flicka.fst hus.fst >sum.fst
  ```
  etc...
- Optimize the final FST (prune and combine nodes to reduce the size)

  ```
  fsmrmepsilon sum.fst >sum2.fst
  ```
- Write a script to test all the instances the FST should cover.

```
#!/bin/bash
echo "Hello!"
echo "This is a very simple user interface for checking the morphology of
certain Swedish nouns"
echo "Write a word form of one of the words:"
echo "flicka, pojke, ros, spik, hus, sko, bild, byte, bil, eller båt"
echo "press 'Ctrl + c' to quit"
while [ $inp != "q" ]
do
read -p "word to check: " inp
lexcompre -l syms/morph.lab -s $inp | fsmcompose - sum7.fst | lexfsmstrings -l
syms/morph.lab
done
```

The output is of the form:

% pojke

pojke pojke N[UTR][SG][INDEF][NOM]

%pojkens

pojkens pojkeN[UTR][SG][DEF][GEN]