# Dependency Parsing

Johan Hellström and Ian Kumlien

January 23, 2006

**Abstract**

This paper presents an implementation of a deterministic parsing algorithm for dependency grammar in Swedish Natural Language. The pursued implementation is based upon the Java programming language and not Perl or Prolog commonly used in this field of research. This project constitutes a part of the undergraduate course *Language Processing and Computational Linguistics* by the department of Computer Science, Lund University.

## 1 Introduction

An essential part of processing natural language is to properly understand and determine the hierarchy of dependence. This means, given an arbitrary sentence, to find the main word and how all the other words come to depend upon this one word. The main word by convention or almost without exception turns out to be the main verb or the most central proper noun, and the dependences to this word naturally can be expressed as a tree structure. In this structure the main word is elevated to the root of the tree allowing for two branches, left and right, designed to express the dependencies of the words found on either side, still maintaining the original order of words, creating sub-trees for dependencies.

## 2 The Model of Our Parser

The art of dependency parsing of Swedish texts has been expertly explored by Joakim Nivre [2], and it was his work on parsing which inspired us to implement a similar parser designed in Java, not Perl or Prolog which most often is selected for natural text parsers today.

### 2.1 Nivre's Principal Parser

The principles of this kind of parser was developed by Joakim Nivre[1], and has many similarites to the basic shift-reduce algorithm for context-free grammars[2] The principal parser make use of a rule set composed of suggested word class pairs ordered in expected frequency and four methods; Right-arc (RA), Left-arc (LA), Shift and Reduce as presented in table 1.

[1] principles and outline by Nivre [1].
[2] extensively defined by Aho et al [4].

| | | |
|---|---|---|
| Initialization | $\langle \mathbf{nil}, W, \emptyset \rangle$ | |
| Termination | $\langle S, \mathbf{nil}, A \rangle$ | |
| Left-Arc | $\langle n\|S, n'\|I, A \rangle \rightarrow \langle S, n'\|I, A \cup \{(n', n)\} \rangle$ | $\texttt{LEX}(n) \leftarrow \texttt{LEX}(n') \in R$ $\neg\exists n''(n'', n) \in A$ |
| Right-Arc | $\langle n\|S, n'\|I, A \rangle \rightarrow \langle n'\|n\|S, I, A \cup \{(n', n)\} \rangle$ | $\texttt{LEX}(n) \rightarrow \texttt{LEX}(n') \in R$ $\neg\exists n''(n'', n') \in A$ |
| Reduce | $\langle n\|S, I, A \rangle \rightarrow \langle S, I, A \rangle$ | $\exists n'(n', n) \in A$ |
| Shift | $\langle S, n\|I, A \rangle \rightarrow \langle n\|S, I, A \rangle$ | |

Table 1: Formal description of Nivre's Parser [1]

# 3 Implementation Outline

For a matter of practicality, offering attractive possibilities of reuse and customizing, the parser application is actually composed of three semi-independent parts, intended to be executed in sequence. The source text, a fully annotated collection of Swedish natural language called "Talbanken MALT" [6], first was processed to determine the frequency of different pairs of word groups appearing in typical. The 100 most frequent, in ascending order, were selected as set of rules, considered to well enough represent typical Swedish natural language in general. Next, the parser principles of Joakim Nivre were consulted [1]. While using an unaltered syntactic approach we introduced different choices of classes, native or optimized for Java performance. Our parser made use of the annotation tagging offered in the text-source as far as word classes were concerned, but ignored the sentence structure tagging during this phase. Finally, the outcome of our parsing was compared with the previously neglected sentence structure tagging available in the source text. Statistics were created based upon word correctness and full sentence completeness.

## 3.1 Rule Extraction

The rules used in the parser are a list of pairs of word classes ordered by how frequently they occur. By pre-parsing the entire source of natural text for every single pair of words and keeping score of what combinations of word classes is most frequent, the most proper account of how frequent word pairs in typical Swedish language are is obtained. This list will control the order in which word class combinations are considered as well as determine the time effectiveness of parsing since the list will be consulted top-to-bottom until the specific pair is matched or else the search will fail

## 3.2 Parsing Dependencies

The basic implementation consists of four different operations, in order they are: Left-Arc, Right-Arc, Reduce and Shift. In our implementation we added a extra top priority shift operation that makes sure that there are elements on the stack.

First construct a string from the word classes involved, elements: the first element on the stack and current element.

Here is some documented pseudocode, this is all done in a iterative way, this the continue statements below.

```
if stack.isEmpty then:
  shift

Left-Arc:
  if stack.peek.isDone is false then: /*if the element hasn't been handled*/
    if tmp in rules_left then:        /*if it's in the list of rules      */
      stack.peek.setDone_LeftArc      /*the element is a left arc         */
      stack.pop                       /*remove the element from the stack */
      continue                        /*continue the iteration            */
    else:
      if element in rules_root then:  /*if it's in the list of rules      */
        stack.peek.setDone_Root       /*the element is a root element     */
        stack.pop                     /*remove the element from the stack */
        continue                      /*continue the iteration            */

Right-Arc:
  if stack.peek.isDone is false then: /*if the element hasn't been handled*/
    if tmp in rules_right then:       /*if it's in the list of rules      */
      element.setDone_RightArc        /*the element is a right arc        */
      stack.push element              /*push the element to the stack     */
      continue                        /*continue the iteration            */
    else:
      if element in rules_root then:  /*if it's in the list of root rules */
        element.setDone_Root          /*the element is a root element     */
        stack.push element            /*push the element to the stack     */
        continue                      /*continue the iteration            */

Reduce:
  if stack.peek.isDone is true then:  /*If the element on the stack is done*/
    stack.pop                         /*then remove it                    */
    continue                          /*continue the iteration            */

Shift:
  stack.push element                  /*push current elementto the stack  */
```

Each rule continues the iteration and thus terminates any additional parsing done.

### 3.3 Correctness Evaluation

Since the source text is fully annotated, even when sentence structure is considered, the process of determining the correctness of the parsing is simply a matter of processing the source text yet another time, comparing the parser's findings with the tags of annotation. Since the annotation is available word-by-word, the correctness statistics can be calculated by word as well as by sentence

# 4 Results and Conclusions

By parsing the corpora, keeping score of the occurrences of word class pairs, we select the top 100 of these for rule set. This is then used in the Nivre style (Left, Right, Reduce and Shift) dependency parser. When a correctness score parse finally is performed, our findings are that 70,515 words from the corpora of 103,939 words are correctly determined. This would then give an average of 67.84%. For the completion of entire sentences, in essence a 100% correctness of words per sentence, the result is the somewhat modest count of 546 out of 6,316 sentences, or 8.64%.

# 5 Comments and Future Improvements

We need to point out that no further guides or pre-processing steps were taken, besides the general top 100 pair-of-word-classes statistics.
Further improvements, would surely be made by recognizing subclasses of certain word classes, say for instance making the parser sensitive to specific prepositions linked to certain verbs. But the path along this thinking is long and by these steps the method is no longer as general as now is the case. Different language parsers would of course have all different subclasses and the top-100 count would be even more dependent upon the nature of the training corpora. By applying this subclass distinction, the results would improve at the cost of narrowing the range of application.
Neither did the parse allow for different context recognition. Certainly, one would not consider the often short-and-to-the-point chapter headlines to be as fluent and be composed of as colourful language as the rest as the text. By adding methods to make this distinction, one would in fact have even more valid statistics from the word class frequency parse and probably raise the score. But once more, general application would be sacrificed. The types of headline composition vary as the nature of the corpora does and also, most likely the trends of headlining is different in different languages.
In our code there is a small race condition. If Right-Arc is triggered and we get a Left-Arc right after, it can set the *setDone_right* element to be root as well, while this is intended behavior it still wrong. Any given sentence should only have one root element. This is easily fixed with some post processing or better rules, but it was not one of our goals due to time constraints.

# 6 Acknowledgements

We would like to thank Pierre Nugues for his patience and guidence during this project, which initally was a challenge due to some confusion about the details of the algorithm development. It was by invaluable guidence and much effort the completion of this project came to be.

## References


[1] Joakim Nivre (2003), *An Efficient Algorithm for Dependency Parsing*, School of Mathematics and Systems Engineering, Växjö University, 12 p.

[2] Joakim Nivre (2005), *Inductive Dependency Parsing of Natural Language Text*, School of Mathematics and Systems Engineering, Växjö University, 209 p.

[3] Pierre Nugues (August 2005), *An Introduction to Language Processing with Perl and Prolog*, Department of Computer Science, Lund University, 544 p.

[4] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman (1986), *Compilers: Principles, Techniques and Tools*, Addison Wesley.

[5] (1997), *Stockholm Umeå Corpus*, Produced by Department of Linguistics, Umeå University and Department of Linguistics, Stockholm University.

[6] (1997), *Talbanken MALT*,
Available at Nivre's website; http://w3.msi.vxu.se/ nivre/research/maltDT.html (Certified jan 2006)


This paper was printed using LaTeX $2_\varepsilon$.