University of Oslo
Department of Informatics

Parsing of Esperanto

Bente Christine Aasgaard

Cand. Scient. Thesis

23rd February 2006

Preface

This thesis is submitted to the Department of Informatics at the University of Oslo as part of a candidata scientarum degree.

Acknowledgements

I am most grateful to my supervisor, Dag Langmyhr, for his support, guidance and never-ending patience.

I would also like to thank my co-workers at Ifi and USIT, UiO, for giving me the possibility to write this thesis and for their support and help.

Mari Wang, Anette Gjetnes and Ellen Engdahl have helped me by reading and commenting this thesis and have contributed with many useful ideas. Peder Stray deserves special thanks for all the Postscript help I have received from him.

Abstract

In conventional computer language parsing, languages are traditionally expressed in formal grammars and parsed with LR(k) or LL(k) algorithms. However, the size, complexity and ambiguities of natural language processing make these tools unsuitable for natural language processing (NLP).

In this thesis we explore how suitable these tools are for expressing and parsing Esperanto. We present a morphology parser for Esperanto, as well as a syntax parser. We discuss ways of improving this parser technique. We present a formal grammar for Esperanto. Furthermore, we suggest an extension to EBNF which will make it more suitable for natural language processing.

Contents

List of Figures

List of Tables

List of Examples

List of abbreviations

A number of abbreviations will be used in this thesis, not all of them well known.

AI	Artificial Intelligence
BNF	Backus-Naur Form
BSO	Buro voor Systemontwikkeling
DLT	Distributed Language Translation
IM	Intermediate Language
NL	Natural Language
NLP	Natural Language Processing
OO	Object Orientation
SVO	Subject Verbal Object
SWESIL	Semantic Word Expert in the Intermediate Language

Chapter 1

Introduction

The first section of this chapter will give an introduction to the goals of this thesis, what we will try to do and what we will hopefully achieve. In the last section we will discuss what language is and how we perceive it.

1.1 Motivation

We will throughout this thesis investigate the parsability of Esperanto. The thesis will describe the planning, implementation and testing of an Esperanto parser. The parser will attempt to do both a fully morphological and syntactical parsing.

We will test whether

- the Esperanto morphology is well suited for machine parsing, or
- a full syntax parsing of Esperanto is as complex a task as for most natural languages it is related to. See section 3.1 on page 17 for an overview of Esperanto's history and creation.

1.2 Problem description

Traditional parsing techniques require a complete formal grammar (see section 4.3 on page 33 for an introduction to grammars) of the language to be parsed. That language must also be context-free and unambiguous. Natural

languages (see section 1.4 on the facing page for a discussion of language terminology) do not fit into this pattern. They are huge, complex and full of exceptions. So the traditional conclusion is that formal grammars are not suitable to describe natural languages and conventional parsing techniques and algorithms are not efficient tools to parse them.

While Esperanto is not covered by the term *natural language*, it is very similar to one. It is a multi-purpose language aiming to cover people's everyday communication needs, ranging from chats among friends to literature and poetry. It includes all the common word types and grammatical features such as cases, prepositional expressions and nested sentences, only to mention a few.

Our theory is that Esperanto still can be expressed by the use of a formal grammar and parsed with traditional techniques. The arguments supporting this theory is that even though Esperanto's grammar is complex and extensive, it is also less complicated due to a total lack of exceptions. The structure of word-building (the morphology) in Esperanto is concise and highly regular which also supports our theory.

So the goal of this thesis is to test this theory. We anticipate that we can divide the task into three sub-tasks.

1. Writing a scanner. However, a traditional scanner is insufficient, we aim at writing a complete morphology parser. The morphology of Esperanto may be regular, but it is so concise and powerful that ordinary regular expressions are not powerful enough to give us the correct word-structure. (We do however believe that it is enough to give it the lexical information needed by the parser.)

2. We will write a formal grammar for Esperanto. We do not aim at making it 100 percent complete, since the extent of that task is a little too much for this project. But of course we hope to make it cover as much as Esperanto as is possible in our time schedule.

3. Write a syntax parser suitable for Esperanto's grammar. This parser must differ from ordinary parsers in the way it must handle an ambiguous syntax. We anticipate that a given sentence in Esperanto can be expressed by more than one syntax tree, so the parser must be able to give us all possible solutions, not just one.

1.3 Thesis outline

In chapter 1 on page 1 we will define the goal of our thesis and explain the problem. In chapter 2 on page 9 we will give a brief summary of some other projects relating to Esperanto processing while we in chapter 3 on page 17 will give a brief introduction to Esperanto itself, so that the reader unfamiliar with the language may be better equipped to read the rest of this thesis. Chapter 4 on page 29 will describe our preperations, the tools we've chosen and the design we want to implement.

In chapters 5 on page 45, 6 on page 53 and 7 on page 57 we will describe the analysis of the morphology, our implementation of the scanner/morphology parser and the tests of this implementation.

Chapters 8 on page 61 will describe our analysis of the BNF grammar of Esperanto that we have written and the implementation of our meta-BNF parser (or BNF builder).

Following this pattern chapters 9 on page 67, 10 on page 73 and 11 on page 77 will describe our syntax analysis, the implementation of the syntax parser and our tests of this implementation. The final chapter 12 on page 95 discuss our conclusion and suggest further work.

The texts we will use in our testing is listed in appendix A on page 99, while appendix B on page 103 lists our translation scheme between the scanner output (object structure from our morpholgy parser) and the BNF structure the meta-BNF parser generates. The meta-BNF describing Esperanto's grammar is included in appendix C on page 111 and the BNF this meta-BNF expands into is shown in appendx D on page 115.

1.4 Parsing of natural languages

> Living grammatical patterns are always in a state of change.
>
> [12]

1.4.1 Definition of language

We think it would be wise to begin by establishing clear definitions of different types of languages. The term *language* covers a broad selection of languages, which varies not only in location and set of users but also fields of usage, media and degree of complexity.

Most people associate *language* with the spoken (and often written, but that is not necessary) language used amongst a group of people with a joint origin or located in the same area, that is languages such as English, French or Chinese. It might be tempting to call them national or state languages, but there are several examples of languages that doesn't fit into this definition. As an example, Urdu is a language used by a huge ethnic group but is not an official language of any country. English, on the other hand, is not the language of only one but several ethnic groups, as well as an official language in several countries.

But, as mentioned in the first paragraph, a language is characterized by more than this. Some languages are used as a mean for verbal communication between people, other are used in a written medium and yet some are used to interact with machines. Some languages have a very narrowly defined area of use while others are multipurpose and therefore so much more complex. Spoken languages of ethnic groups are concidered most complex since they shall cover all aspects of the everyday communication needs of the speakers.

In conclusion,one definition of language may be that it is an established protocol for some kind of commmunication and interaction between two or more participants. It is here interesting to note that the participants do not have to be two individuals, it can also be a human and a computer, or two computers. Or two different processes on the same computer.

The terms *natural language* and *artificial language* are often used to separate the time-evolved languages of ethnic groups and the man-made constructed languages such as Esperanto or Ido. This is however by many regarded as an unlucky choice of terms because it is then easy to make the conclusion that a language defined as artificial can't feel natural in use, and therefore never can be a serious contestant to a natural language as the preferred language in a given situation. It is therefore a broad acceptance of the terms *ethnic languages* and *planned languages*. These terms not only use the origin of a language to classify it, but they also say something about the evolution schema of that language. And more importantly, they don't make any assumptions on the usability of that language.

Ethnic languages can change in many ways, the major ones listed here:

- Natural mutation.
  - Languages tend to evolve from more complex and diverse grammatical features into simpler ones with fewer pattern breaking features, such as the transformation from case-based grammar into the use of prepositional expressions instead. The vocabulary showing that words are getting shorter with fewer syllables.
  - The existence of new things requires new expressions to cover them.
- Influence from other languages.
  - Cultural influence.
    A country often for some time gains dominance over one or several other countries, either military, financially of cultural. Or all three. The sheer impact of such dominance will most often cause the language of the subordinate countries to adapt parts of the dominant countrys ways and language. Sometimes the language of a subordinate country is completely suppressed and as a result dies.
  - Origin of invention.
    When a new phenomenon comes into play and gets spread across cultures, they often bring with them a set of vocabulary. Such as many of the religious words in european languages can be traced back to Greek.
- Planned changes.
  - Not very common but there are a few examples of major language reformations that has been planned.
  - Most countries operate with a standard of language, stating what is the correct grammar of their official language and also what is to be prefered. Such standards are revised and changed on a regular basis, but such changes are mostly made to make sure that the standard reflect the language that is actually in use (which of course is in constant change, as we have just explained).

The evolution of planned languages is less complex:

- Planned changes.

Not a very long list. However, in the case of Esperanto, there is two more causes of change. The continued apperance of new things requires new words to cover them. Sometimes this will force a need for new stems and people will naturally make such stems themselves. This is getting ahead of ourselves, but let us still make the following notion. The morphology of Esperanto is an agglutinating one, where you with the aid of a relatively small vocabulary of stems and a rich affix systems build words on the fly. This does not however mean that there aren't prefered and more ways to express something although one can achieve it withe several different builds. But people changes their ways and things go in and out of styles, and so will the morphological preferences of the Esperanto community do too.

It is worth to note that even though we have just presented arguments against the use of the term *natural language*, we will still use it a few times in this thesis. The term *natural language processing* is an established concept, so we will be using this expression.

This has brought us back to the quote we began this chapter with : *Living grammatical patterns are always in a state of change.* An interesting question is whether we can in the case of Esperanto, concider the morphology dynamic, while the syntax of being in a static state.

1.4.2 The nature of languages

To parse a text written in an ethnic language is not only a complex and difficult task, but also an uncertain and confusing one. The sheer size of any ethnic language alone makes the assignment difficult, and the immensely intricate structure of such languages only adds to this. But which elements contribute to confuse?

Idioms

The idioms of a language are a complicating factor because they do not fit into the patterns of that language, they do not obey the grammatical rules.

They may violate the patterns in two different ways, either syntactically[1] or semantically. Examples of the two can be the following:

- *Sure as eggs is eggs.* Meaning something is certain. Eggs are after all... eggs.
- *Step up to the plate.* Meaning to take on a challenge or responsibility.

A person will most likely find the semantically diverging idiom most difficult, because it requires of him to not only detect two or more meanings but also to spot the correct one, while he will only need to memorize the occurunce of the error(s) in the idiom or the whole idiom as one atomic pattern in the case of a syntactically diverging idiom. For our syntax parser the opposite is the case. It will be be based on a strict grammar and all exceptions to that grammar makes it more unmanagable.

[1]Many idioms are remains of syntax that was once included by the language, but they don't concur with the current grammar. Examples of this is the norwegian expressions *til fjells* and *av huse*. The first indicates that *til* once was a preposition guiding genitive, while the latter expression shows a now extinct use of dative in norwegian.

Chapter 2

Other work

In this chapter we will present some other projects that have done work related to parsing Esperanto. They are relevant to this thesis because the typical natural language project concerns itself with translating; a mere syntax parsing is not enough. The projects reviewed here has all either used Esperanto in some way to achieve their task, or they have attempted to do what we want to, a full syntax parsing, though sometimes on other languages.

We must also emphasize that the projects we have chosen are merely a handful of many similar past and current projects out there.

2.1 PC-kimmo and Esperanto

PC-kimmo is a two-level morphology parser designed to generate (produce) and/or recognize (parse) words using a two-level model of word structure in which a word is represented as a correspondence between its lexical level form and its surface level form.
Two-level morphology was first introduced by Kimmo Koskenniemi[14]. A two-level morphology system implements linked lexicons and two-level rules, where the rules are applied on the lexicons to combine them to various valid forms. Jiri Hana [8] wrote his master thesis on an two-level analysis of Esperanto morphology where he used PC-kimmo. It corresponds to the first part of our system, where we will not only be writing a scanner for Esperanto, but also a morphology parser. He reports a very successful result, with a 97.5 success rate when testing on a set of Esperanto texts containing more than 460 000 words. He modestly points out that the good number would not

be as good if tested on a more diversified set of texts, such as newspapers, spoken texts and texts written by many people from different nations. Still he claims that the decrease in success would largely be due to an increase in proper names and not common words.

The work of Hana only looked at a morphological analysis of Esperanto. Very interesting, but we hoped to do so much more.

2.2 Siv Sjøgren

Siv Sjøgren[13] wrote a thesis in 1970 describing an analysis of Esperanto, *En syntaks for datamaskinell analyse av esperanto*. The purpose was to make it possible to use Esperanto as the query language toward SQAP (Swedish Question-Answering Project). SQAP was as research project at Uppsala University aimed at developing a question-answering computer system. Her work is very much like parts of what we've done, she worked out a BNF grammar for both the morphology and parts of the syntax, though she did not make a syntax parser (that would be taken care of by SQAPs already existing framework). The work we have done has benefited much from this thesis. It differs on several points. First of all, we hoped to develop a grammar that can cover all of Esperanto. We also aimed at writing a full parser. And last we will also tried to do a a morphological analysis.

2.3 DLT

Another well-known translation project is the DLT (Distributed Language Translation) [1] project. They too used Esperanto as an intermediate language, more precisely a simplified version of Esperanto. They did finish a prototype designed to translate airplane maintenance manuals from English into French.

As mentioned there have been several projects that have made machine translators based on an intermediate language (IM), or interlingua, where Esperanto, or subsets of Esperanto, has been used as IM. Figure 2.3 on the next page gives a schematic overview over such a translator design.

The use of an IM design is meant to decrease the cost when there are more than one target language. You only need to create a translator between the IM and the target language, something that is less complex than for another

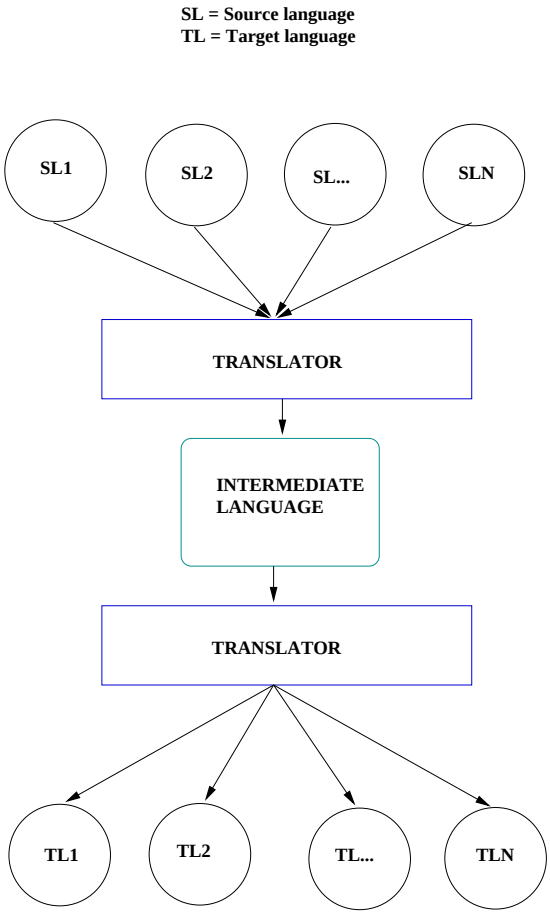

Figure 2.1: Scheme over a machine translator using an intermediate language

more irregular and complex language such as the source language is likely to be.

This system developed a large knowledge bank that contained huge amounts of information about usual connections between words. When the translation program later came across a situation where there were two possible translation for a word it would consult the knowledge bank for a solution.

It is self-evident that the size of such a database can grow infinitely since there are countless of possible connections between words, so such a knowledge bank offer only a limited solution.

To increase the profit from the knowledge bank they used SWESIL (Semantic Word Expert in the Intermediate Language), an algorithm that computes the number of steps necessary to find a link between two given words. DLT then chose the alternative in the translation that were the least amount of steps away from the context word in the knowledge bank.

SWESIL only looks at two-words combinations and any combinations based on a larger context is lost. There has later been developed a so-called *Bilinguale Knowledge Bank*[3] that builds up relations between complete texts rather than between pair of words.

Some of the problems that such knowledge banks arise is:

- The sheer size that is needed to make it adequate.
- The algorithms that is necessary to access it efficiently.
- Constructing it. There is a vast amount of words and the number of potential combinations between them is even bigger. Building a knowledge bank that is both large and correct enough is big and complex job.
- The maintenance. Using a modified language as was done in the DLT project can be a drawback; it will become static and after some time not reflect the living and in constant change language it models. Constantly updating the knowledge bank with new combinations is a tedious and never-ending task, but it will be just as difficult to remove the combinations that has grown outdated.

The DLT project took place at BSO (Buro voor Systemontwikkeling, `http://www.bso.ne`), a dutch company, between 1984 and 1990 and was financed by BSO itself and the dutch government. When it ended in 1990 it was not because of lack of fundings, but because scheduled project period was over. Why it was never reopened is a little peculiar since the initial results was regarded as promising.

The perspective of the DLT project was quite different from ours. While we have had a theoretical approach and therefore aimed at achieving a potential complete solution, the DLT project had a very practical work setting where they should achieve an actual translation. This led their attempt to only focus on parts of the language.

2.4 EOparser

EOparser is a morphology parser written in the Ruby programming language, and therefor is quite similar to parts of our work. As the makers of EOparser states themselves, EOparser is only the first step in the creation of an AI.

EOparser offer a text-based UI for querying, see example 2.1, but it can also be used as a library in other applications.
EOparser differs from our parser in several ways :

1. Output
   The program returns strictly string based output, while our parser return a structure of ruby objects.

2. Focus
   It is more concerned with the translation of the input than the grammatical building of it. Example 2.1 clearly shows that it is uninterested in presenting a words morphological attributes if it doesn't know the words semantical meaning or at least part of it.

3. Extent
   EOparser offers only a morphology parser, while we have combined that with a syntactical parser. EOparser is supposed to be used in combination with an external syntactical parser (not yet made/released), so this will not be the case in the future.

Using EOparser

```
benteaa@svartabraud ~/eoparser/EOParser% ruby EO.rb
.....................................................................
### protokolo
.
[ because (vorto 2) || minutesOfMeeting (n 2)]
### pro
.
because (vorto 2)
### toko
.

### mi
.
I (n pronoun 4)
### li
.
he (n pronoun 4)
### mia
.
I (adj pronoun 4)
### mia fotballo
..
I (adj pronoun 4) [ photograph bah (vorto 0) || photograph ball ( 0)]
### mia hundo
..
I (adj pronoun 4) dog (n 2)
### mia hundo estas bela
....
I (adj pronoun 4) dog (n 2) [ is (v present 2) || is (adj 2)] beautiful (adj 2)
### gxin
.
it (n pronoun accus. 4)
### Scrooge gxin subskribis.
...
 it (n pronoun accus. 4) under write (v past 2) .
### scrooge
.

### belela
.
beautiful from (vorto 2)
### bela
```

```
.
beautiful (adj 2)
###
```

Example 2.1

2.5 Other Esperanto-related computing projects

There are many other small and big NLP research projects that in some way make use of Esperanto. One such projects, not unlike DLT, was the work of Stephen Ben-Avi, where he also investigated the use of Esperanto as an IM language. A more peculiar example is the work of Petr Trojanski. Machine translation was an active research field in the years between the first and second World War. Hutchins [10] describes several of the russian pioneers in the field, of whom Petr Trojanskij is of special interest. He applied for a patent on his translating machine in 1933, and was granted one, not for a translating machine but for a "novel method of typesetting". It was a kind of mechanical dictionary, with proposals for coding and interpreting grammatical functions using "universal" (Esperanto-based) symbols.

Esperanto's width and widespread use combined with its highly regular construction obviously makes it an attractive language tool in a translation process so when doing a search there are many references to systems like those explored in this chapter, although most of them only briefly play with the idea without fully investigating it.

Chapter 3

Esperanto

We start this chapter by giving and introduction to the history of the Esperanto language and the Esperanto movement. We move on to section 3.2 on page 19 where we present a concise and short introduction to Esperanto's grammar.

3.1 Overview of the Esperanto language

The man behind Esperanto was Ludwig Zamenhof (1859 - 1917), a polish physician and linguist. Growing up in a multicultural and multilingual part of Warsaw, Poland, he was a first hand witness to the many conflicts that often arise when there are communication problems between ethnic groups.

He constructed the language to supply a tool to help resolve such conflicts and the subsequent problems; the prejudices and oppression that is likely to emerge when one culture gain dominance over others. The first papers about Esperanto was published in 1887 under the pseudonym of Dr. Esperanto, which also lend name to the new language.

The new language had to be easier than natural languages, and relatively just as easy for anyone, regardless of their background. This was what he regarded as the most important means to achieve this goal:

- A regular grammar without exceptions. (See section 3.2 on page 19)

- A small and easily learned vocabulary.
  One of the ways he achieved this was with the help of a large number of fixed affixes. (See chapter 3.2 on the facing page)

- Easily recognizable vocabulary.
  By fetching words from some of the largest languages in the world, foremost English, German and the Latin languages, it is easier for most people to learn the necessary words. About 70% of Esperanto vocabulary has been borrowed from Latin languages, 20% from Germanic languages and a smaller part from Slavic languages.

- Easy to speak.
  A complete and precise pronunciation definition without any exceptions makes the oral learning process easier.

The extent of the language has always varied with time. Just after the second World War the language was especially popular in Russia and Eastern Europe, but when it at some point was associated with oppositional forces it was opposed by the government.

Ironically it has also been opposed, sometimes even forbidden by law, in western countries like France and USA (and Germany before and during the war) on account of being connected to communist organizations.

Although one of the primary goals was to create a simple and regular language as possible, this feature has also been the target of criticism. It was designed to make changes easy as the users needs and wishes change, and small language reforms has occurred frequently. But there have also been more drastic reform movements. One example is the creation of Ido which a group of French linguists initiated with a larger modification of Esperanto in 1907.

The estimated numbers of Esperanto speakers varies between a few hundred thousands up to 10 million. These are located all over the world, North America being the continent most sparsely populated by Esperantists.

The Esperanto speakers are quite productive and tens of thousands of books have been published in Esperanto since its birth, every year a world congress is taking place with more than thousand participants in addition to regional activities taking place.

The Esperanto community also covers numerous organizations, and the international Esperanto union (UEA) was founded as early as in 1908. It acts as

an umbrella organization for countless national, regional and special-interest organizations in the Esperanto world. It also publishes a substantial amount of material in Esperanto and has advisory status in UNESCO.

3.2 Introduction to Esperanto

This section will be written as a short tutorial in Esperanto, giving the non-Esperanto speaking reader a quick introduction to make it possible for him/her to follow the examples trough the thesis.

Naturally we will not cover the complete Esperanto grammar here, but there are a list of language elements that must be explained to get a decent overview of a new language. These elements are:

1. The alphabet.
2. Pronounciation rules.
3. The different word groups and their attributes. (Conjugations, declarations etc). The building blocks used to build sentences.
4. Sentence building.

3.2.1 The alphabet and pronounciation guidelines

The Esperanto alphabet is built upon the Latin one and contains the following characters:
a, b, c, ĉ, d, e, f, g, ĝ, h, ĥ, i, j, ĵ, k, l, m, n, o, p, r, s, ŝ, t, u, ŭ, v, z.

A description of the pronunciation rules:

- vowels
  - a pronounced as *a* in *f**a**r* and *s**a**ke*
  - e pronounced as *e* in *r**e**gina*
  - i pronounced as *i* in *r**i**ver* and *l**i**ttle*
  - o pronounced as *a* in *l**a**w*
  - u pronounced as *oo* in *f**oo**d*

- consonants when these differs from standard UK english pronounciation
  - c pronounced as *ts* in *ha**ts***
  - ĉ pronounced as *ch* in ***Ch**ur**ch**ill*
  - g pronounced as *g* in ***g**ood*
  - ĝ pronounced as *g* in ***g**entleman*
  - ĥ pronounced as *ch* in the scottish word *lo**ch*** or the german word *a**ch***
  - ĵ pronounced as *J* in the french name ***J**ean*
  - ŝ pronounced as *sh* in ***sh**e*
  - ŭ pronounced as *w* in ***w**ell*

- diphthongs
  - sc pronounced as *ts* in *ha**ts***
  - ng always pronounced as two seperate sounds
  - nk always pronounced as two seperate sounds
  - rd always pronounced as two seperate sounds
  - rn always pronounced as two seperate sounds
  - rs always pronounced as two seperate sounds
  - rt always pronounced as two seperate sounds

- combinations of consonants
  All the consonants are pronounced fully, unchanged by the presence of its neighboring consonants.

- syllables
  The accent should be put upon the second last syllable. Prefixes should be separated and given a distinct accent.

3.2.2 Word groups

There are 8 different word classes in Esperanto: noun, pronouns, verb, adjective, adverb, prepositions, conjunctions and determinant.

The first four stand out as being large dynamically built classes. Words within these categories are made by adding a specific ending to a word stem. The four last categories are different by being closed sets of prebuilt static words.

A special case is **adverbs** which is a mix between open and closed. The category contains prebuilt fixed words as well as the possibility of building new adverbs by adding the adverb ending to word stems.

- The noun

  The noun is the major building block, representing the participants in a sentence. They are formed by adding the ending **o** to word stems. To mark plurality the character **j** may then be added. To mark the difference between the subject and the object in the sentence, one must finally add the ending **n** to show what noun is in *accusative* case. When not having the **n**-ending it is by default in *nominative* case.

Use of the noun

viro ĵetas pilkojn al mi - a man throws/pitches balls to me

Example 3.1

  There are only two cases in Esperanto; nominative and accusative, they behave as described in the previous paragraph. To mark plurality as opposed to singularity the ending **j** is added. While *viro* means *man*, *viroj* express the meaning *men*.

  Since Esperanto lacks genitive case, they must express gentive through other forms. We achieve this by using the prepositon **de**. This can be compared to the english use of **of**.

How to express genitive

la pilko - the ball
la pilko de la viro - the mans ball

Example 3.2

- The adjective

  The adjective word ending is **a**. When used in an attributive context it will always directly precede or follow the noun it is describing. The case and number must correspond to the noun it describes. The word endings **n** and **j** are also here used to indicate this.

Use of the adjective

la bela birdo - the beautiful bird
floroj flavaj - yellow flowers

Example 3.3

- The determinant

  The definite article in Esperanto is **la** for both singular end plural nouns. There is no indefinite article for any number.

Use of the article

knabo - boy, a boy
la knabo - the boy
pomoj - apples
la pomoj - the apples

Example 3.4

- The pronoun

  This is a closed set, and although they don't follow the noun formation, they appear in the same sentence contexts. The pronouns are listed in table 3.1.

Esperanto	**English**	**type**
kiu	who	relative individual
tiu	this one	indicative individual
ĉiu	every(one)	universal inidividual
iu	someone	indefinite individual
neniu	noone	negative individual
kio	what	relative thing
tio	that (thing)	indicative thing
ĉio	every(thing)	universal thing
io	something	indefinite thing
nenio	nothing	negative thing
kies	whose	relative possesive
ties	that one's	indicative possesive
ĉies	everybody's	universal possesive
ies	somebody's	indefinite possesive
nenies	nobody's	negative possesive
mi	I	personal
vi	you	personal
li	he	personal
ŝi	she	personal
ĝi	it	personal
ni	we	personal
vi	you	personal
ili	they	personal
si	'self'	reflexive and personal

Table 3.1: Pronouns in Esperanto

Pronouns act very much like we are used to from languages such as English and Norwegian. They can play the same role as nouns, that is primarily the role of subject and accusative object.

Possesive pronouns are made by adding the adjective ending **a** to the personal pronouns. Hence we see that possesive pronouns are actually not pronouns in Esperanto. Often called *adjective pronouns*, they are indeed adjective variants of the pronoun, used to describe a feature of the noun. Esperanto has taken the consequence of this and are using adjectives to express possesive pronouns.

The word endings **n** and **j** are also here used to indicate accusative case and plurality. Plural does of course not make sense with many of the personal pronouns. Usually there can only be one **you**. It is easy to be confused by sentences like:

Miaj okeloj - my eyes

But in fact this is an example of adjective use, since this is an exmaple of the possesive variant of 1st person singular and this is made by turning the pronoun int an adjective by adding the character **a**.

Use of the pronoun

Mi estas dormema - I am sleepy
Li amas min - He loves me

Example 3.5

- The verb

  The verbs is only conjugated in tenses, not in person and number. Each tense has its own final letter to be used together with word stems. These are shown in table 3.2 and in table 3.3. There is no infinitive article (corresponding to English *to* and Norwegian *å*) in Esperanto.

Form	Present tense	Past tense	Future tense
indicative	as	is	os
active participle	anta	inta	onta
passiv participle	ata	ita	ota
active gerund	ante	inte	onte
passiv gerund	ate	ite	ote

Table 3.2: Verb modes and word endings in Esperanto

Form	Word ending
infinitive	i
imperative	u
conditional	us

Table 3.3: Verb modes and word endings in Esperanto #2

present	mi estas kaptanta	I am catching
past	mi estis kaptanta	I was catching
future	mi estos kaptanta	I will be catching

Table 3.4: Expressing imperfect.

Use of the verb

esti bela - to be beautiful
Mi estas malsata - I am hungry
Vi estis dormema - You were sleepy
Mi estos klara - I will be ready

Example 3.6

Complex verbal forms are achieved by using the auxiliary verb *esti* + participle. From this you can express secondary active tenses, passive voice, and nuances of conditional and imperative. These complex forms are not used that often, situations that require forms in English can often be expressed by simple forms in Esperanto. The following examples are taken from Hana[8].

- Imperfect (Imperfekto) is expressed by the auxiliary verb esti + active present participle.

- Perfect (perfekto) is expressed by the auxiliary verb esti + active past participle.

present	mi estas kaptinta	I have caught
past	mi estis kaptinta	I had caught
future	mi estos kaptinta	I will have caught

Table 3.5: Expressing perfect.

present	mi estas kaptonta	I am going to catch
past	mi estis kaptonta	I was going to catch
future	mi estos kaptonta	I will be going to catch

Table 3.6: Expressing predicative

imperfect	esti kaptanta	to be (in state of being) catching
perfect	esti kaptinta	to have caught
predicative	esti kaptonta	to be going to catch

Table 3.7: Expressing infinitive complex forms.

- Predicative (predicativo) is expressed by the auxiliary verb esti + active future participle.

- Complex active infinitives are formed from the infinitive of the auxiliary verb esti + active participle.

- More precise forms of conditional or imperative can be expressed by combining of the auxiliary verb esti in simple form conditional/imperative with active participles.

- Passive voice (pasivavoĉo) is expressed by the auxiliary verb esti + passive participle.

- The adverb

  1. Building adverbs dynamically
  2. Fixed adverbs
     Special occasions : **ne - negating adverb**
     This will always precede the verb it is negating.

imperfect	mi estus kaptanta	I would be catching
perfect	mi estus kaptinta	I would have caught
predicative	mi estus kaptonta	I would be going to catch

Table 3.8: Conditional complex forms

imperfect	estu kaptanta	be catching!, You be catching
perfect	estu kaptinta	You have been/were catching
predicative	estu kaptonta	You shall catch

Table 3.9: Imperative complex forms.

present	mi estas kaptata	I am (being) caught
past	mi estis kaptata	I was (being) caught
future	mi estos kaptata	I will be (in state of being) caught

Table 3.10: Expressing passive voice, imperfect.

present	mi estas kaptita	I have been caught
past	mi estis kaptita	I had been caught
future	mi estos kaptita	I will have been caught

Table 3.11: Expressing passive voice, perfect.

present	mi estas kaptota	I am going to be caught
past	mi estis kaptota	I was going to be caught
future	mi estos kaptota	I will be going to be caught

Table 3.12: Expressing passive voice, predicative.

imperfect	esti kaptata	to be (in state of being) caught
perfect	esti kaptita	to have been caught
predicative	esti kaptota	to be in state of going to be caught

Table 3.13: Expressing passive voice, infinitive.

imperfect	mi estus kaptata	I would be caught
perfect	mi estus kaptita	I would have been caught
predicative	mi estus kaptota	I should be caught

Table 3.14: Expressing passive voice, conditional.

imperfect	estu kaptata	Be caught!
perfect	estu kaptita	You have been/were caught
predicative	estu kaptota	You shall/should be caught

Table 3.15: Expressing passive voice, imperative.

- The preposition

  These are used very similar to what we are used to from English. They are used to express such things as movement, relationship between objects and events, time and so forth and so on. Most of them takes noun in the nominative case, but there are exceptions to this rule; when the prepositional phrase expresses movement the noun is in accusative.

  Prepositions can also be used as prefixes and word stems.

Use of prepositions

sur la tablo - on the table

Example 3.7

- Conjunctions

  As we are used to from English there are two kinds of conjunctions, coordinating and subordinating. Also relative pronouns can be used as conjunctions, see table 3.1 on page 22.

Examples of coordinating conjunctions

kaj - and
aŭ - or
sed - but

Example 3.8

ke	that
se	if
ĉar	because
kvankam	although

Table 3.16: Examples of subordinating conjunctions.

- The affix

  This class has two subgroups: prefix and suffix. They can be added to another word stem (any kind except the determinant), prefix in front of it and suffix after it, to make a new word with a new meaning. Almost all of the affixes can be used as a separate word stem and build a word on its own by adding noun, adjective, adverb or verb ending to it.

Use of affixes

word stem + noun ending : hundo - dog
word stem + suffix + noun ending : hundido - puppy
prefix + word stem + noun ending : ekstari - to stand up

Example 3.9

3.2.3 Sentence building

The word order when constructing sentences in Esperanto is free. They can appear in any order, hence it is impossible to gain help from their position when parsing a sentence. The main ambiguity when using this approach in other languages is often distinguishing between the word functions, especially subject and object, but due to the absolute regularity and uniqueness of the words are chosen based on their *function* this does not pose a problem in Esperanto.

This design would also cause a problem in some languages, such as Norwegian, where word positions are in some cases also used to mark this sentence as a question. This problem is removed in Esperanto by the use of the conjunction *ĉu*, which should be used in questions whenever not any of the other relative words can be used.

Constructing questions

Ĉu vi estas malsata? - Are you hungry?
Kie estas la pilko? - Where is the ball?

Example 3.10

Chapter 4

Planning the work

In this chapter we will try to define which techniques we choose to deploy and the tools we will use in our implementation as well as the reasons behind the choices we've made. We will also try to give a brief, but necessary, introduction to a couple of techniques and fields that will be important to our further work (i.e. encoding issues, BNF/EBNF and general linguistic terminology). Finally we will introduce the basic design of our Esperanto parser and the background for this design.

4.1 Choosing a programming language

> High thoughts must have high language.
>
> Aristophanes

Not to be confused with the motive of this thesis, *Writing a parser for Esperanto*, this describes our choice of programming language in which to implement this parser.

No matter what pattern matching technique (see section 4.8 on page 42) we would end up choosing we need to choose a language to write the actions that shall be executed when successful matches are made. The language should satisfy the following requirements:

- Powerful pattern matching.

- Make tweaking easy.
- Suitable for writing large modular systems.

A quick look at the most **likely** alternatives gives us the following list:

- C/C++
  C is a low-level imperative programming language while C++ is mainly C plus object orientation.

  C's major force is the performance speed, but speed is not a big issue for us. Neither do we need the advantages that follow a low level language; it would rather be a hindrance.

- Perl
  Perl is a strong contender. It is an interpreted language with dynamic typing designed with extra focus on string handling, pattern matching and text manipulation.

  It is both fast and has a very powerful implemention of regular expressions. It also works nicely together with bison (yacc). It's main drawbacks are a somewhat more cryptic notation than high level languages such as Java and Ruby[1], and it has a less thorough implemention of object orientation[2].

- Java
  Java is an object oriented language designed with the intention to achieve platform independent code, built-in network support and the possibility to securely execute remote code.

  The advantages of choosing Java includes its widespread use; many people know it and there are many good Java tools out there. It is also a high level language which is a force when doing simulations like a language parser.

  We feel however, that it is neither well suited to be used in combination with tools such as flex and bison or has a strong implementaion of regular expressions. Java is also a very verbose language, attempting to force the programmer to become equally verbose and thorough. Not such a bad thing, but a side effect is that even small tasks grow big.

[1] It should be noted that Ruby code may be written just as cryptic as any Perl code, but Ruby allows for more verbose code. What is regarded cryptic, intuitive or various shades between, is anyway matter of personal opinion.

[2] Perls OO implementation will be rewritten in the future 6.0 version, so at that time this point may have less relevance.

- Ruby
  Ruby is another high level interpreted scripting language largely based on Perl, C and smalltalk. It has native OO support but also functional features inherited from smalltalk. It has dynamic typing like Perl and thorough exception handling as in Java. It is purely OO as Java, but the interface make it easy to make imperative or procedural programs without the explicit use of OO. It has also inhereted Perls powerful pattern matching mechanisms, although being slower. We are also more comfortable with the scope rules in Ruby compared to those in Python. Another argument in favour of Ruby is our existing skills at it.

- Lisp and similar functional languages
  Lisp is, and has been for many years, the most widely used language in AI and NLP. The structure of Lisp languages makes them easy to parse which in turn makes them suitable for metaprogramming. That is yet again a major reason what they have been believed to be suitable for AI and NLP projects.

  The main drawback with Lisp for us is our lack of programming skills in it. It takes time and a lot of effort to achieve an appropriate level of skill in a language.

- Python
  Python is a another interpreted language with much of the same strengths and functionality as Perl.

  We regard Perls regular expressions as more powerful, but Python has a better native OO implementation. Python is also considerably slower. The most significant argument against choosing Python is lack of expert skills and rescent training in it as well as the presence of good (and very similar) alternatives in Perl and Ruby.

So, to sum up why we've chosen Ruby:

1. It has broad functionality for string matching and manipulation, but is at the same time suitable for large modular systems.

2. It is object-oriented. Language analysis is in its nature work on lists of objects of different classes (i.e. "word classes"). That makes the program abstraction more intuitive, at least to me.

3. It is a relatively high-level language, with a terminology close to human speach. Since natural languages are highest level languages[3], it is convenient to use a high level language to do work on them.

4. We like Ruby. Personal preference does matter.

4.2 Alphabet, transcription and encoding

Esperanto uses an alphabet that differs slightly from the English alphabet, though also being a Latin based one, as described in chapter 3. The non-standard letters are *ĉ*, *ĝ*, *ĥ*,*ĵ*, *ŝ* and *ŭ*.

In most Esperanto texts available in electronical format, non-formatted to be fit for parsing, these letters are transcribed after one of several common conventions, some of which are show in table 4.1[4].

Convention	**ĉ**	**ĝ**	**ĥ**	**ĵ**	**ŝ**	**ŭ**
pre-circumflex	^c	^g	^h	^j	^s	^u (sometimes also ~u)
post-circumflex	c^	g^	h^	j^	s^	u^ (sometimes also u~)
post-apostrophe	c’	g’	h’	j’	s’	u’
post-backquote	c‘	g‘	h‘	j‘	s‘	u‘
post-x	cx	gx	hx	jx	sx	ux
post-h	ch	gh	hh	jh	sh	uh

Table 4.1: The most common transcription methods used when writing Esperanto

The most commonly used of these are the pre-circumflex and the post-x. However, common practice as it may be, we thought it would be better if our parser internally uses the complete Esperanto alphabet. This would be best accomplished by adding unicode support. We have, however, scheduled this feature for future versions, and as of today the parser is only meant to support the pre-circumflex convention as the transcribation method used on input texts. The parser itself also represents the special characters after this convention, in the code.

[3]The level of a planned language is determined by how close it is in vocabulary and syntax to a natural language. See (`# TODO: Find link`) for further details.

[4]The content of this table are common and found many places, although readers who visit `http://steve-and-pattie.com/esperantujo/alphabet.html` may notice a strong resemblance.

4.3 BNF and EBNF

BNF and EBNF [7] notations is used to describe the grammar of context-free languages. A definition of BNF notation is shown in example 4.1.

A definition of BNF expressed in itself

```
syntax       ::= { rule }
rule         ::= identifier "::=" expression
expression   ::= term { "|" term }
term         ::= factor { factor }
factor       ::=   identifier |
                   quoted_symbol |
                   "(" expression ")" |
                   "[" expression "]" |
                   "{" expression "}"
identifier   ::= letter { letter | digit }
quoted_symbol ::= """ { any_character } """
```

Example 4.1

The EBNF notation (extended BNF) has three more operators in addition :

- ? : which means that the symbol (or group of symbols in parenthesis) to the left of the operator is optional (it can appear zero or one times)
- * : which means that something can be repeated any number of times (and possibly be skipped altogether)
- + : which means that something can appear one or more times

These extra operators makes a grammar written in EBNF notation much more compact than one written in standard BNF, although the same grammar can always be written in both notations.

4.4 Linguistics terminology

Throughout this thesis we will expect the reader to have a basic knowledge of linguistics, so that when we use terms as *accusative* and *nominative* they are understood. He should be comfortable with analyzing a text and be able to understand parse trees.

4.5 The overall design of our parser

The design of the parser will be as described in figure 4.1 on page 35. It consists of three specialized parsers, a BNF parser that traverse the BNF grammar and builds an object structure, a scanner/morphology parser that reads the sentence and builds a list of objects where each object represents one word in the given sentence and all its attributes and a syntax parser that takes as input the structures from the two other parsers and match the list of symbols against the BNF rules and concludes with a number of possible correct syntax trees.

We will discuss the implementation of the BNF parser in chapter 8, the morphology parser in chapter 6 and the syntax parser in chapter 10.

4.6 The need for a scanner/morphology parser

Why would we then write the scanner as a separate parser? There are generic tools out there (such as PC-Kimmo) that can do the job well.

We ended up with a scanner/morphology parser where the Esperanto morphology is rather hard-coded into it. We would consider this a poor solution in most cases, but in this there was several good arguments in favor of such a solution; Esperanto morphology is so non-complex and transparent, and the need for a good input set into our syntax parser.

The former argument was what made it possible to make such a hard-coded parser, while the latter was the reason why we wanted to do it this way.

There are several generic morphology parsers out there, such as PC-kimmo (see section 2.1), that would only require of me to describe Esperanto's morphology as a set of rules written in their specified syntax. In order to use it as input to our syntax parser i would also have to do some amount of work on PC-kimmos output data.

We concluded that the benefit of writing our own scanner/morphology parser from scratch, would give us a sustainable extra amount of flexibility that would be hard to surpass. It would certainly be worth the loss of the advantages a more generic parser approach would offer.

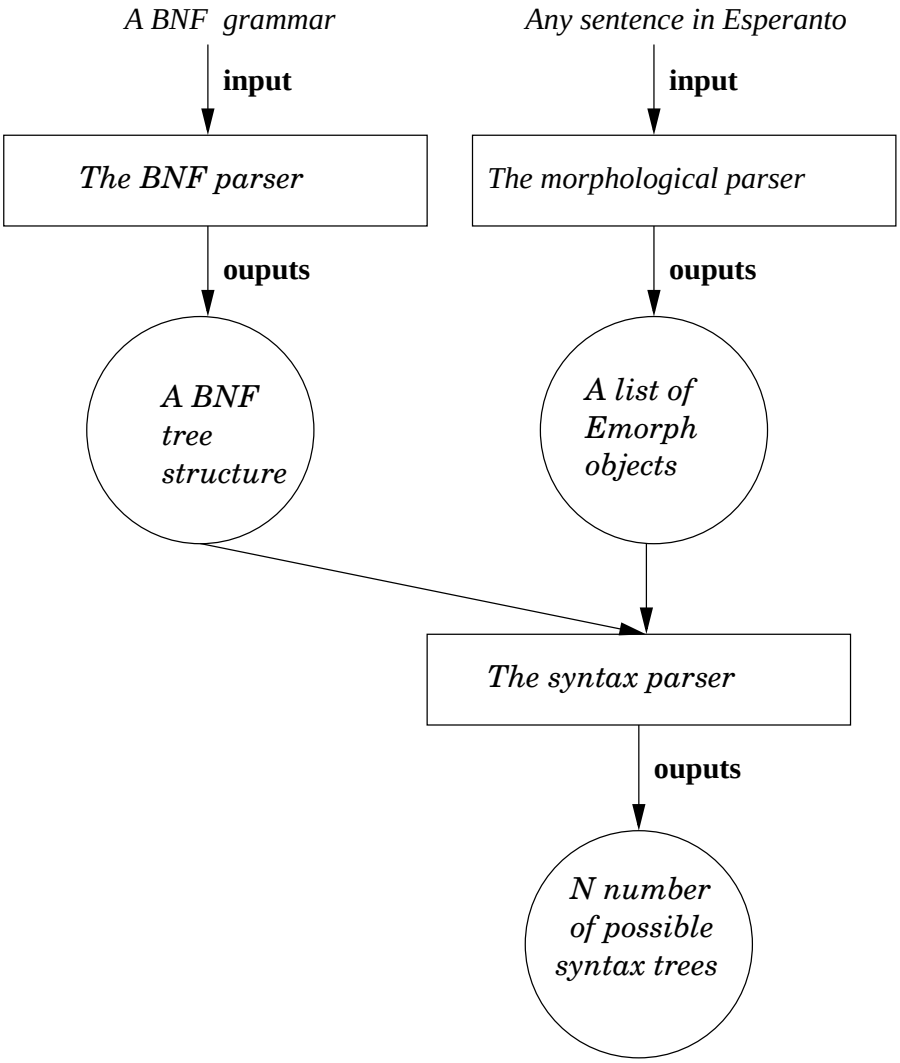

Figure 4.1: The complete parser design

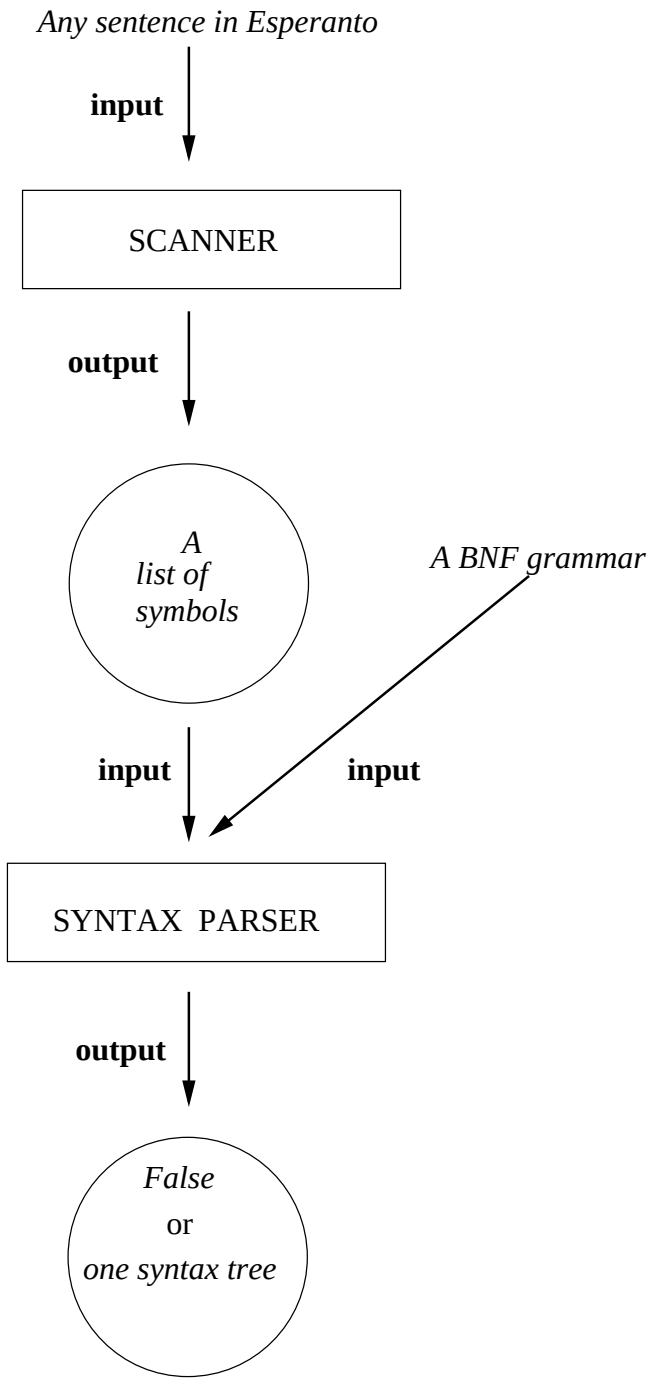

Figure 4.2: Common parser design

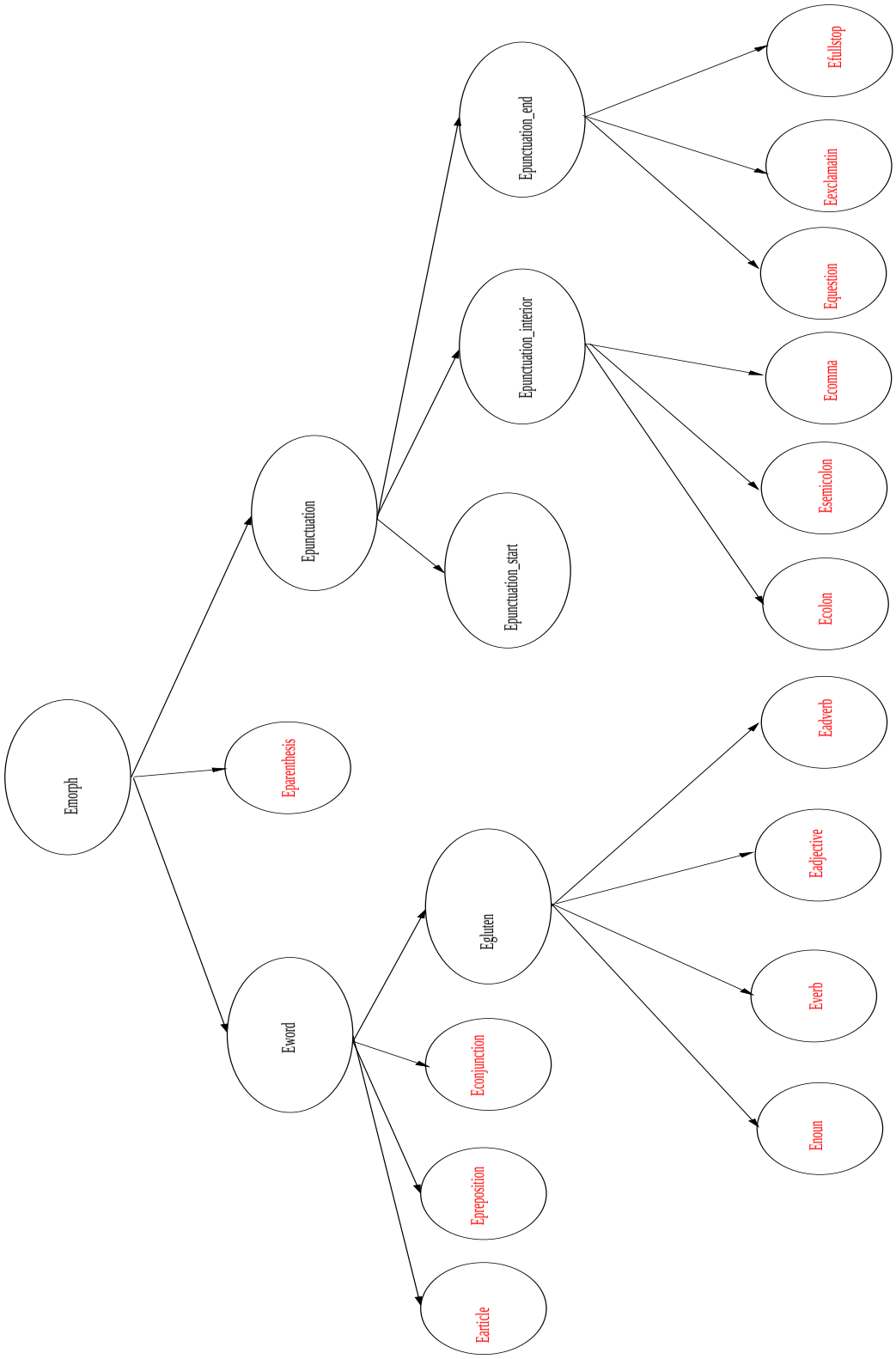

Figure 4.3: The morphological structure

4.6.1 A morphological parser, a morphology parser or a scanner

There might be some confusion between what we need, what we want and what we are actually making as a frontend to our syntax parser. What we need is undeniably a scanner. A program or a module that we can run the wanted Esperanto text through and then have it feed us with one symbol at the time, tagged with all the necessary lexical attributes we need to know for the further parsing.

In addition to this typical scanner behaviour we want our scanner to do more. Not only to tag each lexical symbol with all the lexical attributes needed, but also to do a thorough morphological analysis of each morpheme. Concider the word *protokolojn*. Our scanner will feed our syntax parser with the word, tagged with the lexical attributes *noun*, *accusative case* and *plural number*. A morphological analysis will also reveal that the word is a compound word made up of the prefix *pro*, the root stem *tokol*, the noun-indicating morpheme *o*, a morpheme *n* indicating accusative case and a morpheme *j* indicating plurality. (The scannner would already have discovered the last three morphemes, but it would only be interested in the occurence of them, not in the way they are put together.)

Why is a morphology parser of any interest when we have stated that our syntax parser only needs a scanner?

1. The results from the morphological analysis will indicate the success rate of the scanner. It will be a lot easier to detect errors in the morphological analysis-results than when looking at the output from the scanner alone.

2. If we should try to apply the results from our scanner into another application, for example an Esperanto to Norwegian machine translator, there will unarguably always be necessary with a morphological analysis. And when knowing that in no matter what setting the parser should be used it would also be necessary to do that step, it is only a natural consequence to include it in the parser itself.

Extending the scanner to also perform such a morphological analysis will turn it into a morphology parser as well. We will throughout this thesis use both of the terms *scanner* and *morphology parser*, making the selection based on whether we're discussing the scanner part or the morphological analysis

(or result), but have in mind that it is in fact the same program module we are referring to and that it sometimes can be appropriate to use any of the terms which again may lead to some confusion with the reader.

4.7 The need for a meta-BNF parser

Our meta-BNF will make use of EBNF notation, but why do we need to extend it even further?

It is easy to explain why we want to make use of a EBNF notation; postulating that it takes more than a hundred rules to describe Esperanto, probably closer to a thousand or more, this is really a must. To manually keep track of that many (very similar to each other) rules is a task hard to do correct, and it would be unwise not to use the aids available.

Defending the choice to extend the notation further takes a little more hard work. Consider the sentence:

> *Mi kisis la knabon, ĉirkaŭprenis la knabinon kaj kuris poste al la hejmo.*[5]

The EBNF[6] covering this sentence could be as shown in figure 4.7 on the next page. A correct generated parse tree would then look like the one shown in 4.7 on page 41.

We can see that it correctly defines the sentence to consist of three main clauses, *Mi kisis la knabon*, *[mi] ĉirkaŭprenis la knabinon* and *[mi] kuris poste al la hejmo.*

Now consider the sentence:

> *Mi kisis la knabon, kiun mi amas, ĉirkaŭprenis la knabinon kaj kuris poste al la hejmo.*

It is almost identical with the former sentence, the accusative object in the first main clauser has just been expanded with an apposition.

full_sentence	:= sentence commasign bisentence affpunctterm
sentence	:= mainclause
bisentence	:= bisentencesub? coconjunctionterm sentence
bisentencesub	:= bisentence interpunctterm
mainclause	:= subject verbal object
mainclause	:= verbal adverbial+
verbal	:= **verb**
object	:= anounphrase
coconjunctuon	:= **kaj**
anounphrase	:= **la**? **noun-accusative**
nnounphrase	:= **la**? **noun-nominative**
adverbial	:= **adverb** | prepexpr
prepexpr	:= **preposition** nnounphrase
interpunctterm	:= **comma**
affpunctterm	:= **full stop**

Figure 4.4: EBNF describing the sentence *Mi kisis la knabon, ĉirkaŭprenis la knabinon kaj kuris poste al la hejmo.*

Making an attempt at expanding the EBNF grammar to cover this extension, we could end up with the result shown in figure 4.7 on page 42.

It does look plausible at first sight. A closer inspection will however reveal a big flaw. Remember that we want the output to be three main clauses seperated by a comma and a conjunction.

1. *Mi kisis la knabon, kiun mi amas,*
2. *[mi] cirkaŭprenis la knabinon*
3. *[mi] kuris poste al la hejmo*

The problem arise between the two first main clauses. The EBNF grammar states that a full sentence may consist of any number of main clauses seperated with commas except the last two which is seperated by a coconjunction. Even though there seem to be a comma between the end on the first and the second mainclause (*... amas , cirkaŭprenis ...*), this comma is in fact part

[5]English: *I kissed the boy, hugged the girl and ran afterwards [towards] home.*

[6]Some of the minor EBNF rules from the EBNF we have developed have been omitted to improve readability.

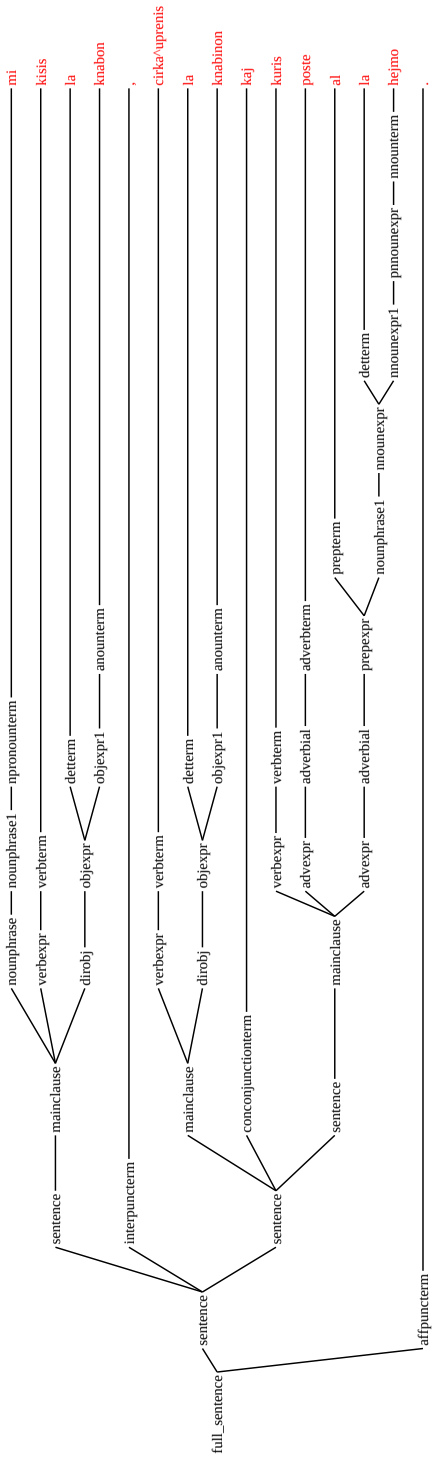

Figure 4.5: Parse tree for the sentence *Mi kisis la knabon, ĉirkaŭprenis la knabinon kaj kuris poste al la hejmo.*

full_sentence	:= sentence interpunctterm bisentence affpunctterm
sentence	:= mainclause
bisentence	:= bisentencesub? coconjunctionterm sentence
bisentencesub	:= bisentence interpunctterm
mainclause	:= subject verbal object
mainclause	:= verbal adverbial+
verbal	:= **verb**
object	:= anounphrase
apposition	:= interpunctterm sentence interpunctterm
coconjunctuon	:= **kaj**
anounphrase	:= **la**? **noun-accusative**
nnounphrase	:= **la**? **noun-nominative**
adverbial	:= **adverb** | prepexpr
prepexpr	:= **preposition** nnounphrase
interpunctterm	:= **comma**
affpunctterm	:= **full stop**

Figure 4.6: BNF meant to describe the sentence *Mi kisis la knabon, kiun mi amas, ĉirkaŭprenis la knabinon kaj kuris poste al la hejmo.*

of the first main clause. We need it in the apposition rule to mark that we've reached the end of the apposition. So if the EBNF grammar above should describe out sentence correctly, we would need to alter the sentence with another comma like this:

> *Mi kisis la knabon, kiun mi amas,, ĉirkaŭprenis la knabinon kaj kuris poste al la hejmo.*

This is clearly not correct Esperanto, so instead we decided that there is a need to extend EBNF in order to make it possible for the comma to belong to more than one rule. Hence the need for a meta-BNF. The development of the meta-BNF is described in chapter 8 on page 61.

4.8 Choosing pattern matching technique

Language parsing is all about pattern matching. A set of grammatical rules are written and then the input texts are checked against these rules in order to be able to state if the sentences in that text are legal sentences in

the described languages and which rules they are described by. These rules are however nothing but descriptions of language patterns organized in a hierarchical structure.

There are several tools or tecniques we can use for this pattern matching.

- BNF
  In short a BNF specification is a set derivation rules describing legal patterns. BNF is thoroughly described in section 4.3 on page 33. The most commonly used tools to match against BNF grammars are Flex and Bison. Flex is a tool for generating programs that perform pattern-matching on text while Bison is a parser generator. They must be used in combination with a more general-purpose programming language, like C or Perl.

- Regular expressions
  A regular expression is a string that describes a pattern. Many programming languages has built-in engines for handling regular expressions.

We discard Flex and Bison as unsuitable alternatives because they don't support multiple solutions[7]. Nevertheless we feel that BNF is the most appealing notation in which to describe the syntax of Esperanto.Therefore we will use BNF to formalise the syntax knowing that we must implement our own BNF traversing engine. The development of the syntax parser is described in chapter 9 on page 67 and chapter 10 on page 73.

We also believe that the regularity and simplicity of the Esperanto morphology makes it easy to define it in regular expressions. As described in the next sections there are several languages that have powerful built-in regular expressions engines, so we believe it will be expedient to use regular expressions in our scanner. The development of the scanner is described in chapter 5 on page 45 and chapter 6 on page 53.

[7]Which we need.

Chapter 5

Morphology analysis

> A writer is someone who writes, and a stinger is something that stings. But fingers don't fing, grocers don't groce, haberdashers don't haberdash, hammers don't ham, and humdingers don't humding.
>
> Unknown

This chapter has a more thorough analysis of Esperanto, while we in the subsequent chapters describe the design and implementation of the morphology parser.

Even at the morphological level we need to deal with some issues. Most critical is the issue about where to deal with conjugational forms and case.

5.1 Nouns

Nouns are one of the major building blocks of human language and is the first elements children will learn. A typical sentence analysis will try to express *Who did what tho whom?*, and that alone indicate the importance of nouns.

5.1.1 Regular expression

Expressed in regular expression we have:

PF - prefix
SF - suffix
ROOT - word root
NE - noun word ending (*o*)

(1) *PF?(ROOT(NE)?)*ROOT(SF)?NE*

There is still a question whether the optional part *(ROOT(NE)?)* can include affixes as well. Then it would be extended to something like

(ROOT(NE)?) — PF?ROOT(SF)?(NE)?

and the complete regular expression into

(2) *PF?(PF?ROOT(SF)?(NE)?)*ROOT(SF)?NE*

An example illustrating the extension above:

a) puppy : hundido
hund-id-o : (ROOT)(SF)(NE)

This one satisfy both (1) and (2). But let us look at another:

b) puppy eyes : hundidokeloj
c) puppy eyes : okeloj por hundido

In this example **b)** is represented by the regular expression // and hence will not satisfy (1), but will satisfy (2). **c)** on the other hand will satisfy (1) because we have split the expression into several words, each which will satisfy (1). [1]

Another matter is that most affixes can be used as stems. If we can make the generalization that syntactically speaking all affixes can be used as stems

(only that it in all cases does not make much sense semantically), we can get a somewhat simplified and generalized solution.

This raises a new question; should we at all be bothered with semantical concerns? We believe this is an important philosophical and practical discussion that deserves to be explored in detail, but also that (regretfully) that task is beyond the scope of this thesis.

However we are forced to make a choice for *this* parser. Since it (at least originally) is not designed for practical use, but more as a tool when exploring ways of parsing Esperanto, we will disfavor questions of semantics when the syntactic road ahead looks clear.

5.1.2 BNF

We are now left with the BNF grammar for nouns shown in table 5.1.

<noun>	:	<nouncom> <nend>
<nouncom>	:	<pre> <stemcomb> <suf>
<stemcomb>	:	<prestem> <mainstem>
<prestem>	:	ϵ
<pre>	:	bo | dis | ...
<pre>	:	ϵ
<suf>	:	aĉ | ad | ...
<suf>	:	ϵ
<nend>	:	o

Table 5.1: A BNF-grammar for nouns

5.2 Verbs

Very similar to nouns. The stem is pretty much identical, but with the noun ending substituted with verb ending. Again we have the case of whether or not to care about that several affixes will actually not appear in a verb compound, since it semantically will make no sense.

The regular expression for verbs will the be :

*PF?(PF?ROOT(SF)?(VE)?)*ROOT(SF)?VE*

The list of word endings(*VE*) for the different conjugation of the verb indicative is as shown in table 5.2 while the remaining verb endings are shown in table 5.2

Ending	temporal mode	Comments
as	present	
os	future	
is	past	

Table 5.2: Verb indicative forms

Ending	temporal mode	Comments
i	infinitive	Ambiguous word ending!
u	imperative	Ambiguous word ending!
us	conditional	

Table 5.3: Remaining verb forms

5.2.1 Other temporal modes

It is worth noting that although Esperanto has modes corresponding to *active participle* and *passive participle*. They are not to be recognized as belonging to the verb class in Esperanto linguistics. They are built with the same stems, the appropriate suffixes to get the desired semantic meaning and finished with the word ending corresponding to the sentence function the word shall have. Whe have shown them in table 5.2.1

Mode	Present	Past	future
Active participle	anta	inta	onta
Passiv participle	ata	ita	ota
Active gerund	ante	inte	onte
Passiv gerund	ate	ite	ote

Table 5.4: The active and passive participles

5.3 Adverbs

This word class is a little different from the two previous presented. Not only can it be built in the same manner as them, the adverb word ending being **e**, but it is at the same time a closed class.

The closed adverbs are divided into several subsets depending on their semantic use.

Given that we keep all the adverbs in an array *ADVERBLIST*, and *ADVE* is the adverb word ending *e*, the regular expression for adverbs will be:

> *PF?(PF?ROOT(SF)?(ADVE)?)*ROOT(SF)?ADVE | ADVERBLIST[1] | .. | ADVERBLIST[n]*

5.4 Adjectives

Adjectives is very similar to adverbs as they can be dynamically built (with the word ending being **a**) and also has a closed set, although this is much smaller than the corresponding adverbial set.

The regular expression for adjectives will almost be identical with the one for adverbs. Given that we keep all the adjectoves in an array *ADJECTLIST* and ADJE is the adjective word ending *a* we get the following:

> *PF?(PF?ROOT(SF)?(ADJE)?)*ROOT(SF)?ADJE | ADJECTLIST[1] | ... | ADJECTLIST[n]*

5.5 Pronouns

Pronouns is a completely closed word class. That makes it less complex to parse. A table lookup is needed for every input word, but this table is not only finite but also very small.

As with verbs we will only need to concern ourselves with some of the pronouns. Looking at table 5.5 we can see that the correlative class *abstract phenomenon* takes on the syntactic form of nouns and will therefore be treated as a nouns by our scanner. Personal pronouns, together with the correlatives

individual and *possession*, does not comply in form neither with nouns or eachother.

If the pronouns are kept in an array *PRONOUNLIST*, the regular expression for pronouns will be as follows:

PRONOUNTLIST[1] | .. | PRONOUNLIST[n]

5.6 Prepositions and Conjunctions

As pronouns prepositions and conjunctions are completely closed word classes. Even though these are finite closed sets they are also a big disturbance, since they overlap.

An example of this ambiguity is *dum*. It can be both conjunctional and prepositional. The scanner will recognice it as both, but the parser will only try out the former. In future version it will be possible to fix in several manners. One will be to try out both possibilities, a solution that would seriously deprave the temporal performance of the parser. A more promising option would be to add a post-morphology parse-check that would add some level of context priority checking.

If the prepositions are kept in an array *preplist* and the conjunctions are kept in an array *conlist*, the regular expressions for prepositions and conjunctions will be as follows:

PREPLIST[1] | .. | PREPLIST[n]
CONLIST[1] | .. | CONLIST[n]

5.7 Numerals

Not surprisingly, this is a noncomplex unit. Numerals will appear in the same contexts as nouns and adjectives; as part of noun phrases. Numerals are described by the following regular expression:

$\backslash d+$

5.8 Correlatives

Esperanto has a number of correlatives which internally have a regular build-up. They take on the form of pronouns, adjectives and adverbs. See table 5.5 for an overview. See under each seperate word class for an analysis of the class.

Meaning and *form*	**Relative**	**Indicative**	**Universal**	**Indefinite**	**Negative**
Individual *pronoun*	kiu	tiu	ĉiu	iu	neniu
Thing *pronoun*	kio	tio	ĉiu	io	nenio
Kind *adjective*	kia	tia	ĉia	ia	nenia
Manner *adverb*	kiel	tiel	ĉiel	iel	neniel
Reason *adverb*	kial	tial	ĉial	ial	nenial
Place *adverb*	kie	tie	ĉie	ie	nenie
Motion *adverb*	kien	tien	ĉien	ien	nenien
Time *adverb*	kiam	tiam	ĉiam	iam	neniam
Amount *adverb*	kiom	tiom	ĉiom	iom	neniom
Possession *pronoun*	kies	ties	ĉies	ies	nenies

Table 5.5: Correlative words and their function in Esperanto

Chapter 6

Building the morphology parser

6.1 Description of the parser

The morphology parser is a small code piece, which when fed with a word or character, returns a list of possible morphological structures for this morpheme.

Relying upon the results from the analysis done in the previous chapter, we build a parser that is dependent upon the use of regular expression. We maintain a collision list, which does not only contain all the possible collisions, such as verb in the infinitive tense collide with personal pronouns in the nominative case, but also all the words from the fixed sets, such as pronouns, prepositions, conjunctions and a number of adverbs. This collisions list is not complete, but it would not be impossible to get somewhere near a complete state.

For each hit we find, we make a new *Emorph* object, or really an instance of one of *Emorph's* subclasses; look at figure 4.3 on page 37 for a complete list of subclasses.

We attach this list of objects to a new *Elex*-object. Each sentence we send into the parser, will therefore come out as a list of *Elex*-objects. An example of this structure is shown in figure 6.2 on page 55.

These *Elex*-objects does at the moment contain nothing but the original word string, the list of Emorph objects and the routine *getMostLikelyMorph*. At the moment this routine always return the Emorph object first in the list, but in a future version we can easily see at least two improvements:

1. It returns the most likely, looking at the words appearance in dictionaries, the rest of the text and other texts.

2. We run a full syntax parse for each Emorph alternative.

The second outline will certainly provide all correct results, but not only will the performance, the execution time, suffer greatly but the number of presented results will most likely increase substantially and make it harder to sort out false hits.

The design of the parser is shown in figure 6.1 on page 54.

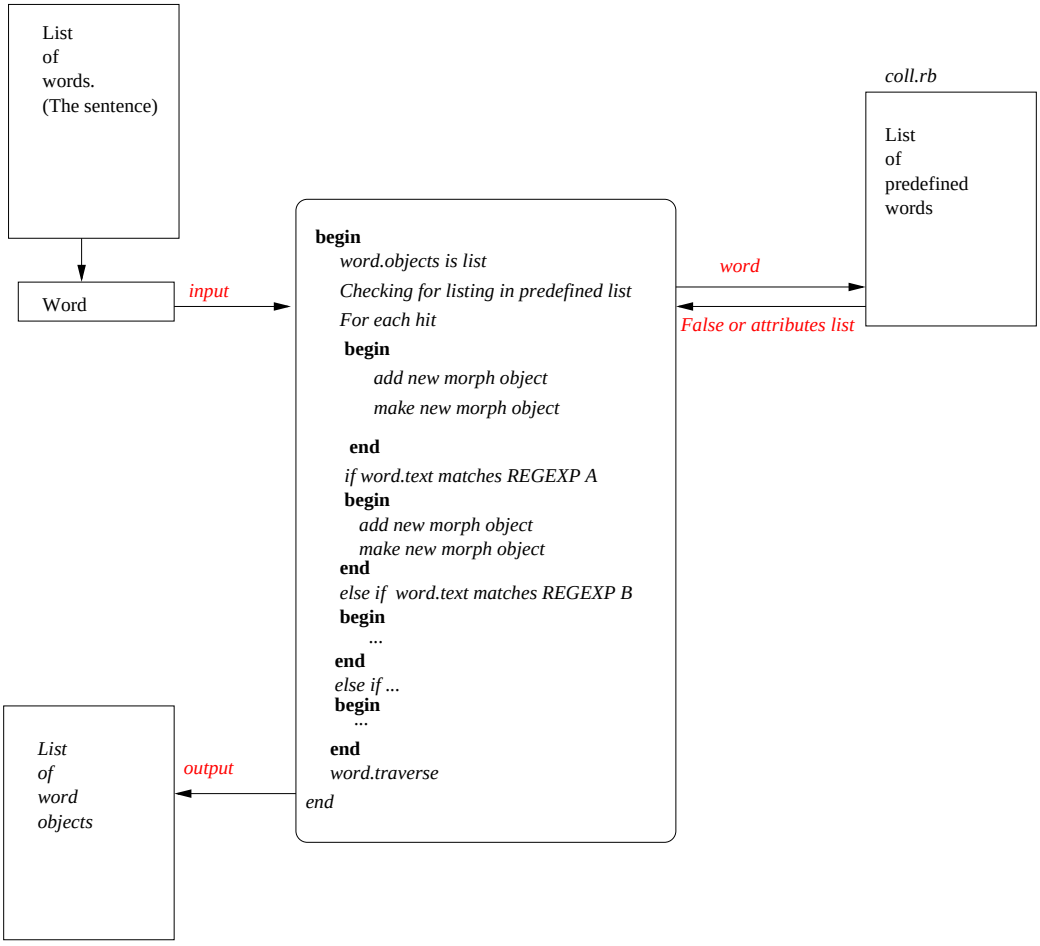

Figure 6.1: Overview of the morphology parser

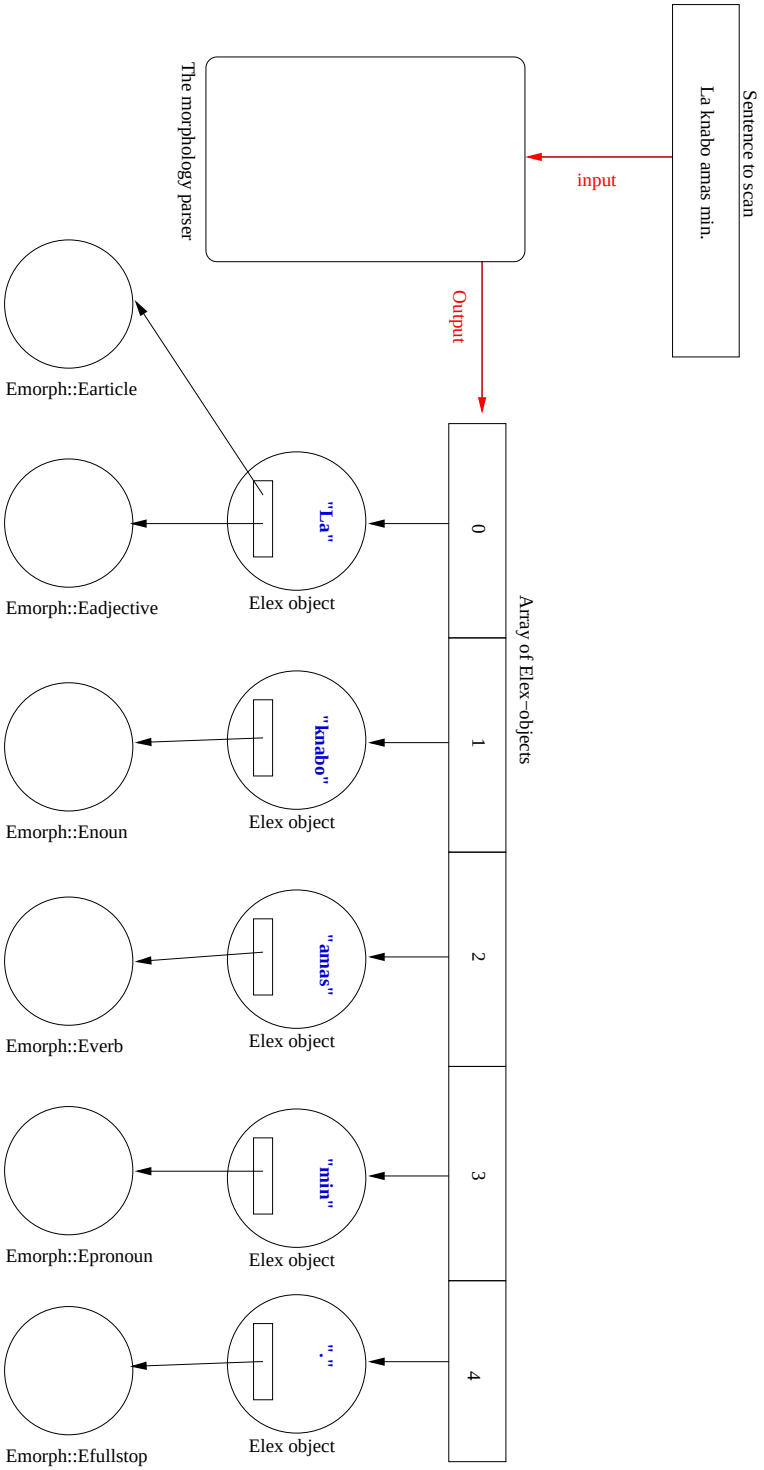

Figure 6.2: Example of the output from the morphology parser (unfinished)

6.2 The non-scanner part of the parser

The parser described in the previous section is doing the job of an ordinary scanner. But it is also worth taking a closer look at the part that turns our scanner into a full-featured parser (see section 4.6 on page 34 for the reason why we have this design).

The first layer of the parser (what we may call the scanner layer) determines the word class and its attributes. The next layer, which in figure 6.1 on page 54 is being entered with *word.traverse* determines the build-up of the word. Not necessary for the parsing itself, but the result is interesting in itself. We can also imagine that in a future version the result from this parsing may be used to determine which one of the possible Emorph objects to choose.

In example 6.1 we see the output from the morphology parser for the word **protokolo**. The #-sign means the parser has found that stem in a dictionary, something which increases the chance for this being the correct build-up. We can therefore conclude that the parser has found four possible builds for **protokolo**, with the third being the most likely. To make it clear, the four possibilities are *prot-okol*, *prot-ok-ol*, *protokol* and *pro-tokol*.

The build-up of the word protokolo

```
protokolo
-------------
      Class:  Noun
      Case:  nominative
      Number: single
      Stems:
            prot#
                  okol
                  ok#
                        ol
            -----------
            protokol#
            -----------
            pro#
                  tokol
            -----------
```

Example 6.1

Chapter 7

Testing the morphology parser

> I don't give a damn for a man that can only spell a word one way.
>
> Mark Twain

When testing the parser, we are interested in three types of results. Scanner performance is most important for the rest of our Esperanto parser, success rate when running a large number of words through it and finally we will look at how well it succeeds at various complexity levels.

7.1 Mass testing

We tested the morphology parser on a set of texts found on the Internet, containing approximately 2000 words. The texts can be found in appendix A on page 99. The results are not entirely unpleasant – 90% of the words were correctly identified. There are, however, a couple of points to be made:

1. Most of the incorrect results occur for one of the following two reasons:
   (a) They are proper nouns but not detected as such.
   (b) The input texts are of too poor quality with lots of spelling errors and broken words.

   This implies that by cleaning the texts somewhat our success score would increase substantially.

2. There is a surprisingly high amount of words that can be parsed more than one way, about 30% in our tests. The reason for that is obviously the fact that we don't require that the word stem must appear in our dictionary.

   A better approach would, perhaps, be a compromise between the two. If a parse tree is qualified from the dictionary, all non-qualifying trees are discarded. This might be something to explore in further work.

3. A few words were classified as more than one word type, but this is due to an inherent ambiguity in Esperanto and therefore to be expected. This ambiguity is already presented in chapter 5 on page 45.

We then tried to do a quick clean-up of some of the texts and run them through the morphology parser again, with quite good results. The score was now 98%.

1. All of the grammatical errors due to erroneous texts was eliminated.

2. The number of proper noun detection errors was unchanged.

3. The number of words with more than one possible parse tree was unchanged.

4. Originally as many as 35-40% of the words was presented with more than one possible word class. But when adding a few simple checks such as that a word stem had to be at least two characters long, this number fell to about 10%.

   Extremely common words such as *ne*, *en*, *ni*, *la*, *de* and *ja* no longer was misclassified as adverbs, verbs or adjectives.

7.1.1 Scanner results

If we only look at the results that are important to the syntax parser, no errors because of ambiguous word class belonging occurred. As already mentioned, we did have a few incidents of this type, but the morphology parser managed to choose the right instance to give the parser every time.

That leaves us with two types of errors that were passed on to the syntax scanner: failing to detect proper nouns and actual grammatical errors in the texts.

We discuss the problem of proper noun detection in-depth in section 11.2.7 on page 91. And if enough resources are allocated to make sure the input texts are grammatically correct it is to be expected that near to all of the errors are eliminated.

7.2 A closer look at some results

The most simple example included is example 7.1. It shows that for the word *hundo*, we can only find one possible build-up, consisting of two morphemes, the stem *hund* and the noun-indicating suffix *o*.

The build-up of the word hundo

```
hundo
-----
      Class:  Noun
      Case:  nominative
      Number: single
      Stems:
            hund#
            -----------
```

Example 7.1

In example 7.2 we show the possible build-ups of the word *Esperanto* itself. The result is a little special beacuse it find two valid results. It finds the combination of the morphemes *esper*, *ant* and *o*, but also encounters the full stem in the dictionary, making this also a valid parse tree. The reason for this is that the word Esperanto is to be considered as a recognised expression, independent of its original build-up.

The build-up of the word esperanto

```
esperanto
---------
      Class:  Noun
      Case:  nominative
      Number: single
      Stems:
            esperanto#
            -----------
            esper#
```

```
        ant
    -----------
```

Example 7.2

The build-up of *ekstari*, shown in example 7.3, reveals another common ambiguity. It successfully parses the word into the morphemes *ek* (prefix indicating the beginning of something), *star* (stem meaning *standing*) and *i* (indicating verb infinitive).

However, we also get a hit on the prefix *eks*, indicating that something is over. Combining this with the *i*, indicating verb infinitive, our parser makes a guess that it could be a stem named *tar*. This stem is not to be found in the parser's dictionary, so it is presented as a less likely result.

The build-up of the word ekstari

```
ekstari
-------
    Class:  Verb
    Tense:  infinitive
    Stems:
        ek#
            star#
        -----------
        eks#
            tar
        -----------
```

Example 7.3

Chapter 8

Creating a BNF grammar

8.1 The background

The BNF grammar for a natural language, for instance English or Norwegian, is a huge and complex rule set consisting of thousands of rules defining a large, intricate web of linguistic do's and don't's.

BNF grammars describing Esperanto will of course be dwarf-like compared to these, since it is highly regular and uniform and less burdened[1] with idioms and other irregular sentence structures.

Nevertheless there are a couple of factors that indicate that the picture isn't that simple.

1. Esperanto was designed to include the same functionality and cover the same specter as any natural language does.

   This alone means a great increase in the number of rules needed to describe a language. The DLT-project (see section 2.3 on page 10) and Siv Sjøgren's parser (see section 2.3 on page 10) avoided this problem by only focusing on subsets of Esperanto.

[1] I am using *burdened* highly ambiguously here. There is no doubt that any form of irregularity poses an annoying hurdle for the parser programmer, but it is nevertheless our opinion that the use of idioms and their like play a big role in defining the beauty of a language. But there is of course more than one type of beauty.

2. The full effect of Esperanto's regular design is focused on the morphology and will therefore be more significant when writing the morphology parser (see chapter 5 on page 45).

   The sentence structure on the other hand allows for much of the same freedom as many well-known natural languages, so even without all the idioms and peculiar exceptions found in these we are looking at a large number of rules.

As described in section 4.7 on page 39 this size makes it convenient to write a separate BNF parser. Then we can write our grammar in a meta-BNF style, and let this BNF parser:

1. Expand the meta BNF rules into correct BNF rules.
2. Group these rules into an object structure convenient to traverse for our syntax parser.

8.2 Analyzing the language

Before we can describe our meta-BNF syntax, we must describe the BNF notation and its limitations in order to make it obvious why we need the extensions we put into our meta-BNF.

Recall the explanation for designing our own meta-BNF from section 4.7 on page 39. Let us have another look at the example sentence causing trouble.

> *Mi kisis la knabon, kiun mi amas, ĉirkaŭprenis la knabinon kaj kuris poste al la hejmo.*

The problem with the EBNF grammar we made was that we need the second comma to appear in two different rules. It is both needed to show that the apposition (*, kiun mi amas,*) has ended and to separate the two first main clauses (*Mi kisis ka knabon, kiun mi amas,* and *ĉirkaŭprenis la knabinon*).

Taking the EBNF grammar from table 4.7 on page 42 and modifying it so it actually covers this sentence it would become as shown in figure 8.2 on the facing page.

The mystery is revealed in the *apposition* rule. We need that second comma in the *apposition* rule, so we can have a way to determine when we've reached

full_sentence	:= sentence interpunctterm bisentence affpunctterm
sentence	:= mainclause
bisentence	:= bisentencesub? coconjunctionterm sentence
bisentencesub	:= bisentence interpunctterm
mainclause	:= subject verbal object
mainclause	:= verbal adverbial+
verbal	:= **verb**
object	:= anounphrase
apposition	:= interpunctterm sentence interpuncterm#
coconjunctuon	:= **kaj**
anounphrase	:= **la**? **noun-accusative**
nnounphrase	:= **la**? **noun-nominative**
adverbial	:= **adverb** | prepexpr
prepexpr	:= **preposition** nnounphrase
interpunctterm	:= **comma**
affpunctterm	:= **full stop**

Figure 8.1: Meta-BNF describing the sentence *Mi kisis la knabon, kiun mi amas, kaj kuris poste al la hejmo*

the end of the apposition. But it is equally needed in the *full_sentence* rule, to separate between the sub-sentences of a full sentence.

That is why we have introduced the #-operator as shown in that rule. It signals that we need that match, but once we have it, we can leave the word as unreadq. We have introduced a lookahead functionality to the EBNF notation.

8.3 Parsing the meta-BNF

The techniques used when parsing the meta-BNF are quite simple. The rule set is scanned and each rule is split into rule name (left side) and rule body (right side). For each rule name there is created an Etructure instance which in turn contains an array reference to all the possible right sides for that rule.

There is also maintained a lookup table where each time a rule x occures in the rule body to a rule y, then y is registered in the hash with x as key. This will make it easier to disregard unnecessary rules when doing the full syntax parse later.

8.4 BNF rules naming convention

As described in 8.1 on page 61, it is likely that our BNF grammar will consist of hundreds of rules, if not more. Naming all of these rules so that they all will be human-readable, or rather human-friendly, and still maintain a relationship with their syntactical role is near to impossible in our opinion. And a parse tree for even a short and simple sentence will include several cryptic named rules. Take for example the sentence in figure 11.5 on page 84, *Bela knabino kaj knabo estis tie*. Even the parse tree for this fairly simple sentence consists of more than 15 unique rules.

The reason for this is that in order to describe a syntax element such as a noun phrase, which is a well known unit in a parse tree, we must construct more than five other subrules that in various combinations give a fully description of how a noun phrase can be constructed.

But this illustrates the fact that the main elements that we are likely to find the most interesting will always be close to the root of the parse trees. So we have tried to give these rules names that are identical to their function in the sentence, while the subrules diverge from this layout, often including abbreviations and numbers. But we cannot get away from the fact that it may require some amount of concentration and focus to fully decrypt and appreciate the parse trees.

It could have been tempting to omit all the subrules when drawing the parse trees, only showing the main rules that most of us are allready familiar with. But then we would apparently end up with identical parse trees, since the many of the variations is in the analysis of the sentence elements. So we keep them all.

8.5 Building a BNF structure

In order to make it possible to check an input text against the formal grammar of Esperanto, we need to maintain a parsable structure of the grammar in memory. We did that by building up a tree structure of it.

In figure 8.3 we show the graph for an extremely simple grammar, shown in figure 8.2.

sentence	:=	subject verbal
sentence	:=	verbal subject
subject	:=	**nnoun**
verbal	:=	**verb**

Figure 8.2: A simple BNF grammar

8.6 Matching against our terminals

At this point we will have a list containing *Eterminal* objects, each referring to a terminal from the BNF. When matching a morphologically parsed text against our BNF, how should we match the output from the morphological parsing to our *Eterminal* objects?

The way this parser is designed one will have to build a separate morphology parser for each language applied and hook this onto the parser. Therefore we have chosen to demand that one also supply a list containing the matches, as shown in the example in figure 8.1.

Terminals	**Parsed morphemes**
COMMA	Ecomma
PREP	Epreposition
ART	Earticle
...	...

Table 8.1: Matching the parsed morphemes against the BNF terminals

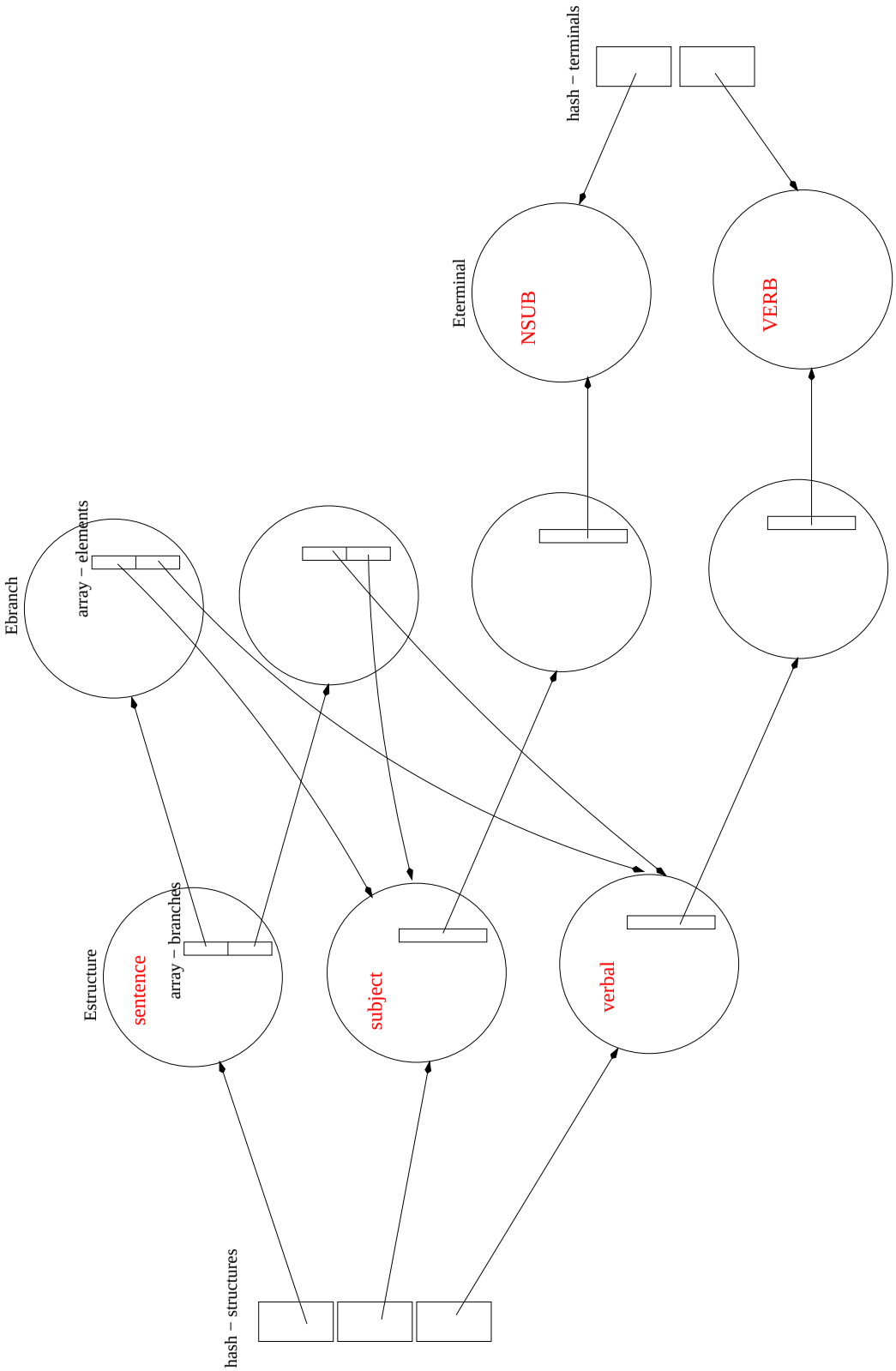

Figure 8.3: Data structure of the parsed BNF grammar

Chapter 9

Syntax analysis

This chapter will introduce some of the problems that will arise in the syntax parsing stage. While we in section 9.1 will discuss the problem of parsing Esperanto, we will in section 9.2 on the following page summarize important syntax features that will have great impact on the parsing solution.

9.1 Problem definition

Doing a full syntax parsing of any given text[1] in Esperanto is a formidable task. We will argue that it is just as complex and difficult as in any other West European language. Some arguments supporting this claim are:

- The word order in Esperanto differs very little from most languages of West European origin.
- Esperanto has the same amount of possible word classes as these other[2] languages and these word classes span the same linguistic width.
- The sentences in Esperanto can be as long and complex as in any other languages, covering the same semantic width.

There are arguments supporting the opposite view, that the task of a full syntax parsing is indeed easier accomplished for Esperanto.

[1] From now on when we refer to such a text we will mean a sentence.
[2] I.e. languages of West European origin.

- Esperanto lacks the use of idioms (see chapter 8 on page 61 for a further explanation of this) that most other natural languages have.
- Esperanto has a small vocabulary.
- Esperanto's vocabulary follows a very regular pattern.

Nevertheless we will argue that even if we agree with with these facts, they are also irrelevant. Most idioms that we are aware of follow the stated syntax rules nicely and don't need any new special rules to be parsed. It is on the semantic level they fail to make sense given the standard rules (or give a completely different meaning than intended) and therefore it is in the semantic parsing stage they must be dealt with. A smaller and much less complex vocabulary does indeed make a huge difference for a parser, but only for the scanner part.

9.2 Sentence syntax in Esperanto at a glance

Let us recap the most essential grammatical rules in sentence building in Esperanto.

1. Free word order but it is common to use SVO[3].

   It is important to notice that by word order we do not refer to the actual literal words, but to their functions (syntax elements) such as *verbal*, *subject*, *object* and *adverbial*. So when we say that the adverbial can be placed anywhere in the sentence, and the adverbial is a prepositional phrase, for example *sur la tablo*, it does not mean that we can put these three words wherever we feel like in the sentence. They must be grouped together, but as long as we preserve them as a phrase, keeping the adverbial intact, we can move that adverbial around.

2. All questions must begin with one of the question words *kie*, *kio*, *kia* or *ĉu*.

[3]SVO is an example of a much used way to classify the sentence structure in languages. English and Norwegian are good examples of SVO languages, the normal sentence structure is Subject Verbal Object. Gaelic is an example of a VSO language while Hindi is an example of a SOV language, to mention a few.

3. The sentences in Esperanto may be as complex as the ones we are used to from languages such as English and Norwegian. For instance the subject in the following sentence is a verb infinitive with a predicate.

   Esti bela estas malgrava.
   - To be beautiful is unimportant.

   A consequence of this complexity is that since some syntax elements (subject, adverbial etc) can be built by proper combinations of other syntax elements, even with complete sentences, the length and depth is in theory infinite.

   La knabo, kiom mi amas, malamas min.
   - The boy, whom I love, hates me.

   La knabo, kiom mi, la juna knabino, amas, malamas min.
   - The boy, whom I, the young girl, love, hates me.

4. There may exist more than one valid way to parse a given sentence. This fact may not be that easy to discover, since the ambiguities may lie hidden in the deeper layers.

   Another way to describe layers may be the height of a given tree branch. Consider the sentence *La bela knabo, kiom mi amas, malamas min.* Layer one will be *La bela knabo, kiom mi amas* [the subject], *malamas* [verbal] and *min* [direct object]. The third layer will consists of *la bela knabo* [nounphrase1], *,kiom mi amas* [relapposition] and *,* [commaterm]. These layers become quite apparent when looking at the parse tree displayed in figure 9.1 on the following page.

Considering the points above, how do they influence the way we need to write our parser?

1. Free word order combined with infinite length and depth will make the BNF grammar huge, with an enormous amount of possible combinations. The parsing technique must be suitable for such a huge grammar.

2. The usual behavior for a parser is to find one valid parse tree and then stop. Our parser must be able to (efficiently) keep looking until every possible combination has been investigated.

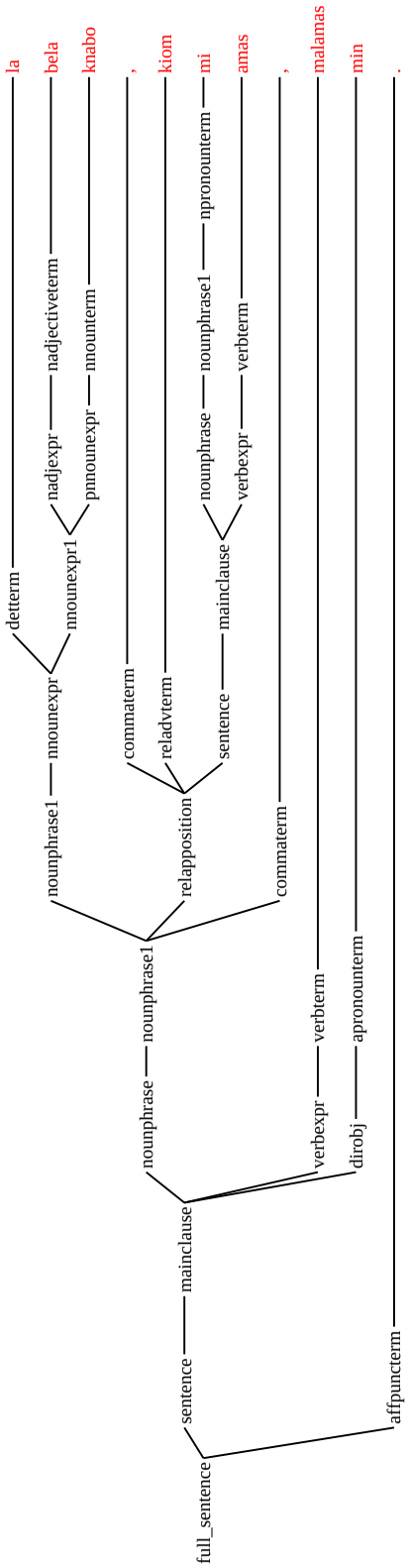

Figure 9.1: Parse tree for the sentence *La bela knabo, kiom mi amas, malamas min.*

3. The previous two points make it clear that the parser must implement well-placed cutoffs, otherwise its performance will be tol slow to be worthwhile.

4. An interesting point is that the question words might at first look like a simplifying factor but in reality is contributing to making it harder. The reader often thinks "Ah, they make it so much more easy to spot interrogating sentences versus descriptive sentences." True, but beside the point. Why do they exist, why do we have to begin all interrogating sentences with them? Because since Esperanto has free word order we can not use word order to show sentence type. So really the question words only make it possible to have free word order and that is not a simplifying feature from a parsers point of view.

Chapter 10

Implementation of the syntax parser

We will discuss issues around matching parsed morphemes to the BNF terminals in section 10.1.1 and choice of parsing technique in section 10.1.1 on the following page.

10.1 Building a morphological structure

Each word in Esperanto corresponds to at least one of the classes defined for word classes, all subclasses of the class *Emorph*[1]. In addition we have defined classes for other types of symbols used, such as comma and parenthesis. An overview of these classes is shown in figure 4.3 on page 37.

10.1.1 Matching our parsed morphemes to the BNF terminals

When we compare the output from the morphology parser to what we can expect from the BNF structure, we see a mismatch. The BNF will make a clear distinction between nouns and adjectives in the nominative and accusative cases. There will also be a similar case with single and plural number, but this does not have any effect on our syntax trees. When doing the pattern matching we end up with a minor dilemma. In order to match *NNOUN*(a

[1]Emorph is shortfor Esperanto-morpheme.

noun in nominative case) correctly to an *Enoun* object from the morphological parsing, we must execute some code doing checks to see whether the *Enoun* object has the case attribute set to "nominative".

The problem that arises is that the simple list containing matches between BNF terminals and morpheme objects (shown in table 8.1) is too simple. It only makes a connection between the name of a terminal and the name of a class defined in the morphology parser.

To solve this problem we have several alternative solutions :

1. To be able to use such a simple name matching list we must introduce less general classes in the morphology parser. The *Enoun* class must be exchanged with at least two classes *Enounnom* and *Enounacc*, possibly four, also introducing the number aspect. This trade has been shown in figure 10.1. The same arguments can be used on the present class *Eadjective*.

2. Another way to solve this problem is to keep the *Enoun* class unchanged and instead extend the matching list from figure 8.1. The list would then contain code to be evaluated to determine a match instead of a name to name connection.
   We see this in figure 10.2 on page 76. To see if a certain word matches NNOUN, the corresponding code must be evaluated (where *ob* refers to the object containing the word) and found true.

We landed on the second alternative. The main benefit by choosing this solution is that it will introduce a frame work suitable for extending. That means more flexibility and a lot easier to maintain. It will also ensure the further existence of parser/scanner independence. It will be possible to choose a scanner that returns other symbols than the BNF grammar knows, since the parser will look up the symbols in the symbol list and get the proper "translation".

Parsing techniques

The options were LL(k) and LR(k). LL requires that the grammar is not only context-free but also that there is no left-side recursion, something that we have not.

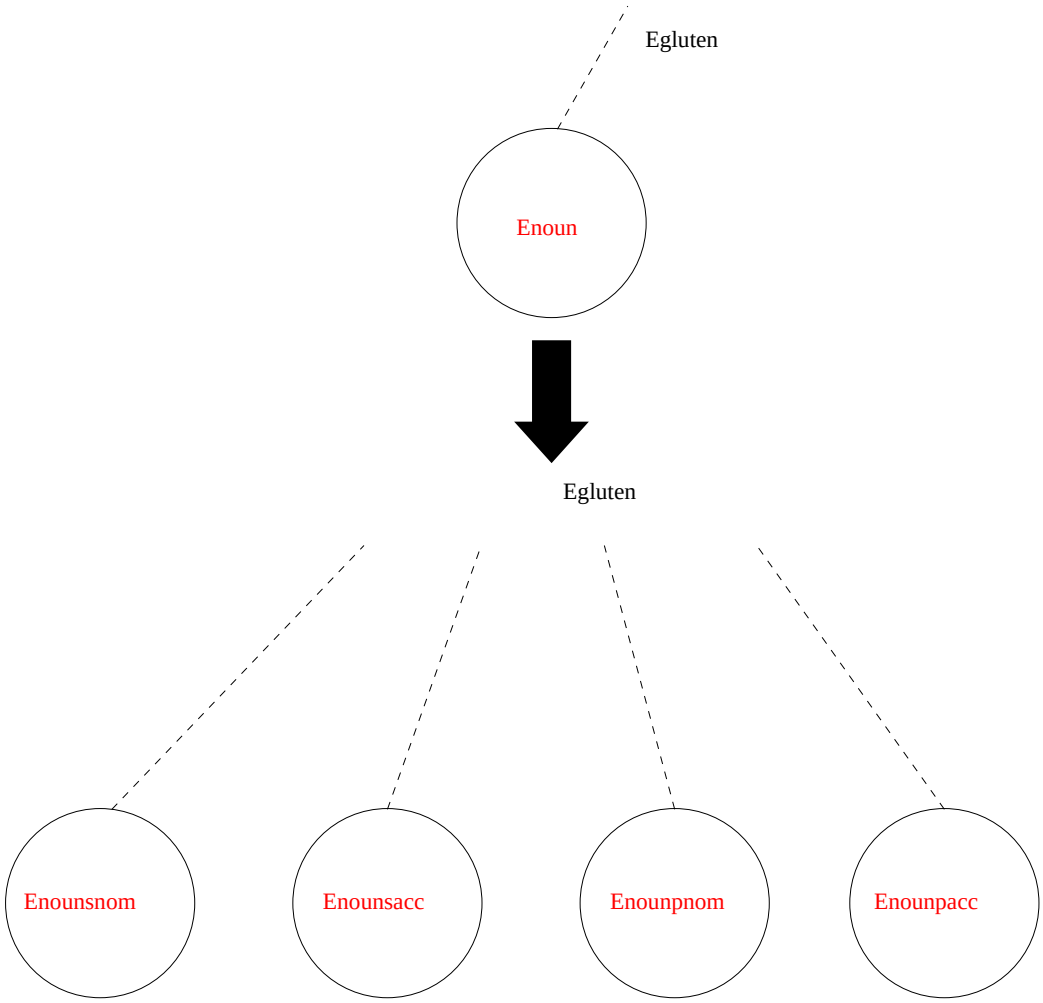

Figure 10.1: Another possible way to group noun occurrences

COMMA	begin ob.type == Ecomma end
PREP	begin ob.type == Epreposition end
ART	begin ob.type == Earticle end
NNOUN	begin ob.type == Enoun ob.case == "nominative" end
...	...

Figure 10.2: Extended matching the parsed morphemes against the BNF terminals

LR is not that rigid. It still requires a context-free grammar though. Nevertheless we believe it still would be possible to use LR, with modifications. When encountering an ambiguity, that is, when a lookahead gives more than one possible tree (or rather branch), we close our eyes to it and fork a new tree (or *trees* with results > 2), an exact copy to the one we're working on, making sure the trees uses different lookaheads. And in this manner we continue as if the ambiguity never happened.

Chapter 11

Testing the syntax parser

In this chapter, we test the parser and analyze the results.

11.1 Mass testing

We tested our parser on the text corpus defined in appendix A on page 99.

The initial success rate was 72%. A few of the failing sentences were caused by faulty scanner information from the morphology parser, but looking back at the testing of this module in chapter 7 on page 57, we were able to get its success rate near to perfect.

Tests show that the texts contain examples on syntax errors, but not that many. This mean that the failing rate of 28% is mostly caused by an imperfect BNF grammar. We believe this is due to two factors:

1. It takes time to write a formal grammar describing a language like Esperanto. Given more time, we would have come closer to achieving it.

2. Such a formal grammar is not only huge but also very complex. It is, in our opinion, possible to get it correct, but this requires a lot of time and work. To improve the grammar, one would need to improve the readability, so that errors would not only be easier to detect but also to eliminate.

11.2 A closer look at some problematic situations

In the following subsections we will try to present and discuss in depth some of the results. We are looking for two types of weaknesses: not only weaknesses and deficiencies in our parser and BNF grammar, but also ambiguities in Esperanto's grammar.

11.2.1 Undesirable valid parse trees

Let us take another look at the sentence *La bela knabo, kiom mi amas, malamas min.* We remember that the correct parse tree is the one shown in figure 11.1 on the facing page. But our parser also offers three more results, shown here in figure 11.2 on page 80, figure 11.3 on page 81 and figure 11.4 on page 82.

All of these are valid syntax trees according to the BNF grammar we have written. When comparing the trees, it is obvious that the two last trees are not desirable results. But nevertheless they are considered valid. Is this wrong?

These two parse trees are considered to include only a noun phrase. In reality you will often encounter sentences that only consist of one word, or more precisely, one sentence element. The following example consists only of the subject, a noun phrase.

> Question: *Kiu estas tie?* (Who is here?)
> Answer: *Tommy.*

Such passages are quite common, especially spoken, but also written. But is it really a legal syntax structure? It is not an uncommon view in several languages that a legal sentence must at a minimum contain an act, that is a verb. So that the grammatical correct answer in the example above is really *Tommy estas [tie].*, but since the presence of the verb is not necessary for the semantic meaning, it is simply left out. We have not been able to verify it, but if we assume that this is the case in Esperanto. We can also say that an implicit verb is not accepted and drop this possibility from the BNF grammar, which of course would mean fewer unwanted results.

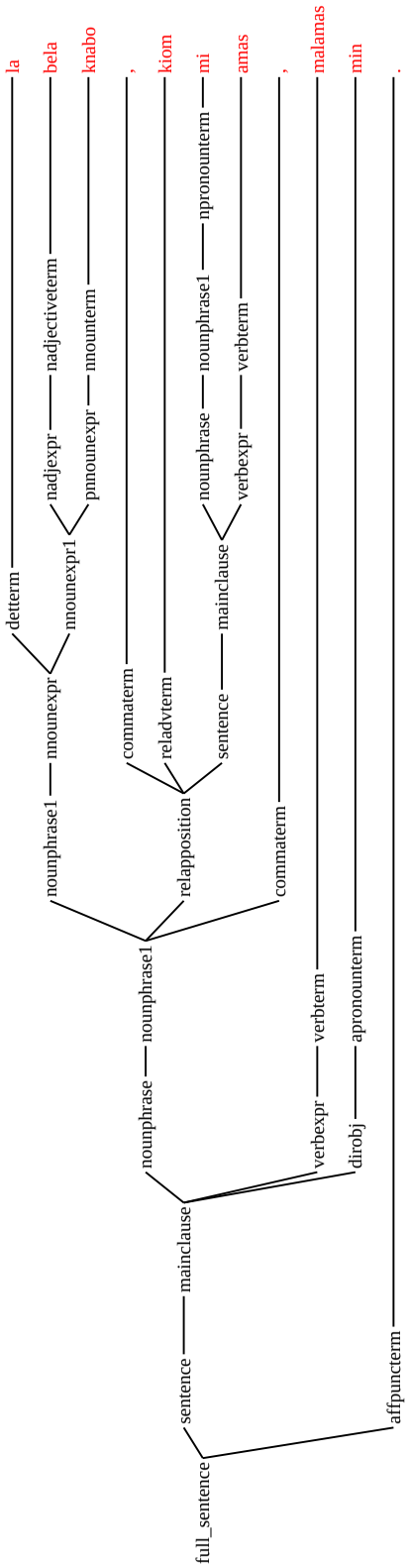

Figure 11.1: The first parse tree for the sentence *La bela knabo, kiom mi amas, malamas min.*

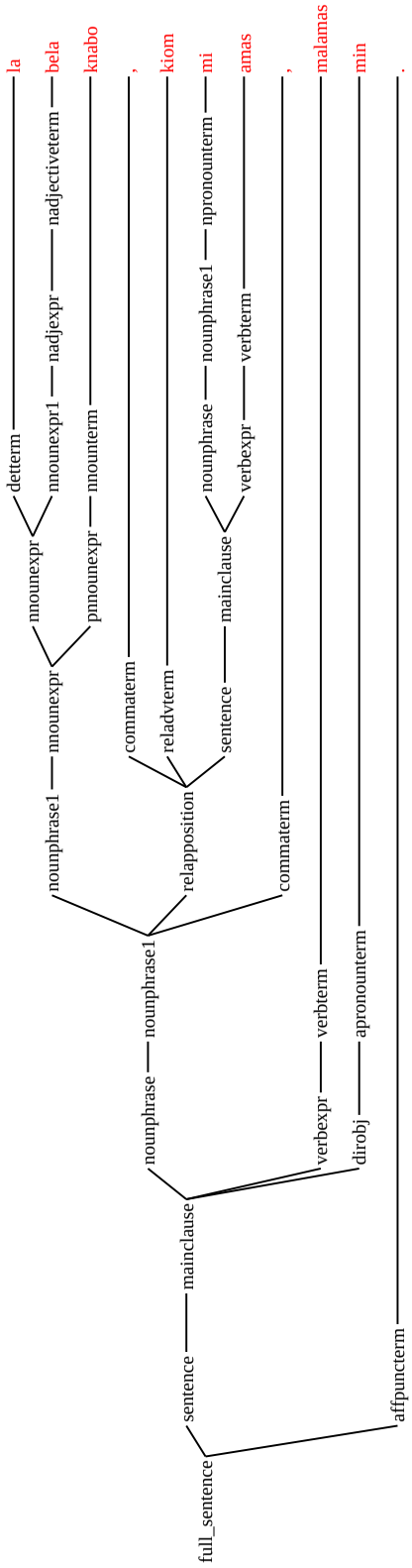

Figure 11.2: The second parse tree for the sentence *La bela knabo, kiom mi amas, malamas min.*

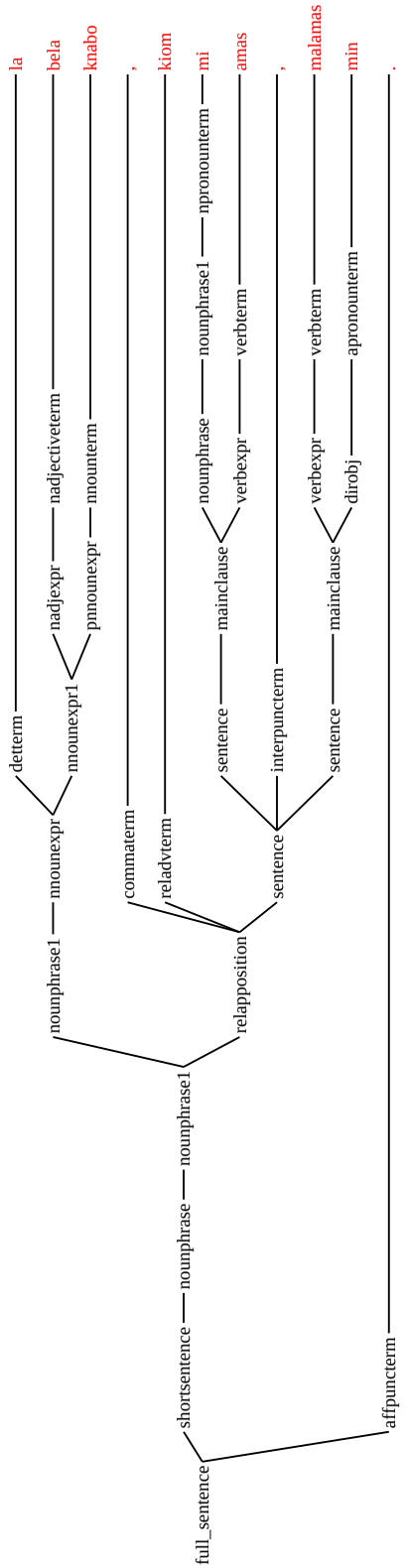

Figure 11.3: The third parse tree for the sentence *La bela knabo, kiom mi amas, malamas min.*

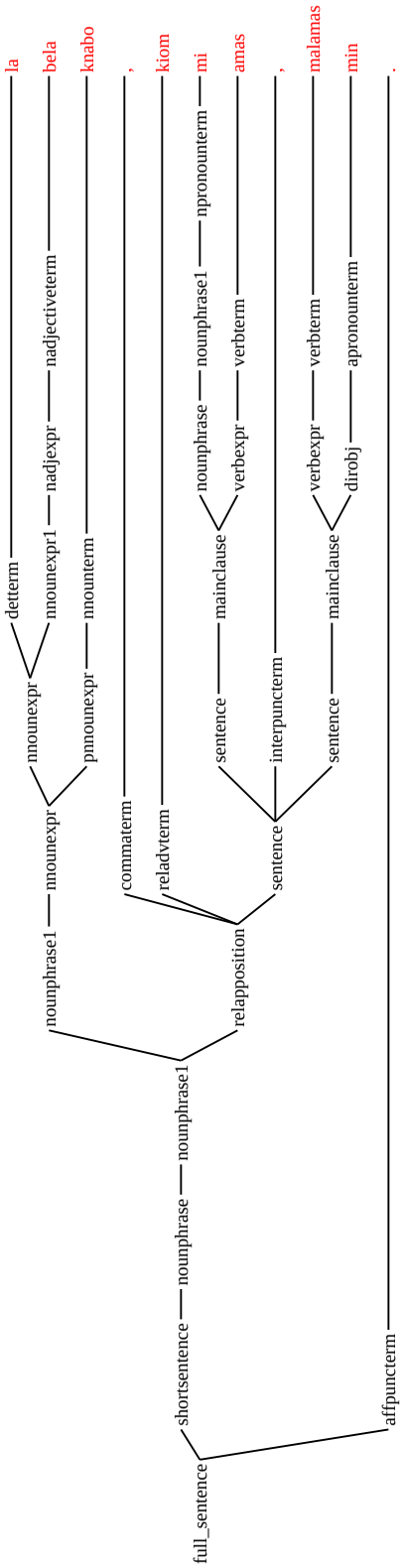

Figure 11.4: The fourth parse tree for the sentence *La bela knabo, kiom mi amas, malamas min.*

11.2.2 Ambiguity in noun phrases

There are examples of ambiguities in Esperanto that our parser fails to recognize. Consider the following sentence:

> *Bela knabino kaj knabo estis tie.*
> - A beautiful girl and boy were here.

This sentence generate the two parse trees in figure 11.5 on the following page and 11.6 on page 85.

It should be able to recognize the constellation *(bela (knabino kaj knabo))* in a third alternative tree, that is that both the boy and girl was beautiful. This reflects of course an inherent ambiguity in Esperanto, that we can not know if the adjective *bela* is in apposition to *knabino* or *knabino kaj knabo*. But there is little doubt that the BNF grammar should be expanded to detect this possibility.

Let us also have a closer look at the two parse trees it does present. They are nearly identical, the main syntax elements are completely identical. But let us inspect the subject, noun phrase, of the sentence, *bela knabino kaj knabo*. The main substructure for this element is *nounphrase [bela knabino] conjunction [kaj] nounphrase [knabo]*. Again this is a branch we can agree with. Inspecting the first noun phrase closer, we see that branch (1) defines this as a *nnounexpr* followed by *pnnounexpr*. Branch (2) on the other hand defines it as an *nadjexpr* followed by a *pnnounexpr*. Take a look at table 11.1 to get a short explanation of these rules.

Rule	Explanation
nnounexpr	A noun expression in nominative case. **Simplified** this means a noun or an adjective.
pnnounexpr	Either a pronoun or a noun in the nominative case.
nadjexpr	An adjective in the nominative case.

Table 11.1: Simplified definitions of a few BNF rules

Branch (2) strikes us immediately as a sane parse tree.

But if nnounexpr can be followed by pnnounexpr and they both have the potential of being a noun, this also gives us the possibility of the noun phrase combination *noun noun*. When is this combination valid? The following sentence is a good example.

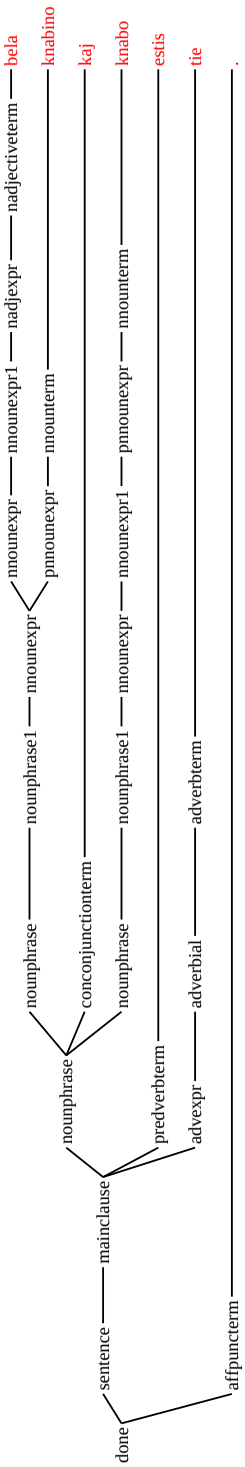

Figure 11.5: Parse tree number one for the sentence *Bela knabino kaj knabo estis tie.*

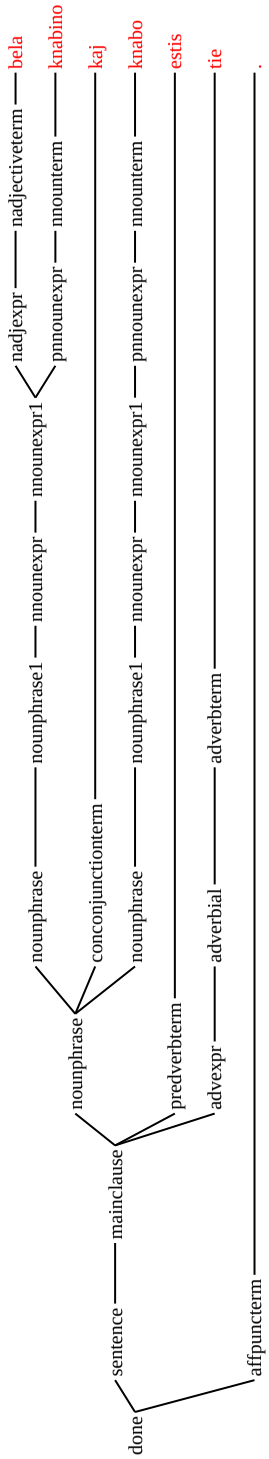

Figure 11.6: Parse tree number two for the sentence *Bela knabino kaj knabo estis tie.*

La instruisto John eliris.
- The teacher John went out.

The conclusion is that it is a valid situation when one of the nouns is a proper noun. Our grammar however does not distinguish proper nouns from other nouns in the nnounexpr rule.

That leads us to the following conclusion:

1. Our BNF grammar should be rewritten to reflect this special situation. See section 11.2.7 on page 91 for a more in-depth discussion about detecting proper nouns in a text.

2. Improving the parser's ability to detect proper nouns. This is discussed further in section 11.2.7 on page 91.

11.2.3 The need for a precedence in the meta-BNF

However, we also see from the table 11.1 on page 83 that nnounexpr can be an adjective. And when we also know that a noun phrase *may* consist of a single nnounexpr we get the possibility that *bela* could be a stand-alone subject. Take a look at the following sentence to see an example of such a sentence being syntactic correct:

La bela estis tie.
- The pretty [one] was here.

This also leads us to the conclusion that an adjective may be linked to an adjective, such as in the following sentence:

La malgranda bela estis tie.
- The little pretty [one] was here.

Another sentence fragment to look at is the following:

La nova, malgranda, ĝentila kaj bela knabo.
- The new, small, polite, and beautiful boy.

This short fragment generates no fewer than ten possible parse trees. This is due to the fact that every adjective can be linked to the adjectives and nouns next to it. So a noun phrase containing a list of adjectives and nouns will necessarily have many permutations and therefore many parse trees.

Our theory is that it is possible to implement a *rightmost precedence* for this situation, causing all the elements to get linked up to the last element, whether it is a noun or an adjective. Which again would mean that eight or nine of the then possible parse trees would be eliminated.

11.2.4 Efficiency

Another important issue is the performance of the parser. Bottom-up parsers are robust and reliable but at the expense of efficiency. Our parser has also a design cut out for unusual poor time performance, because we never stop until every last possibility has been explored. So while we ordinarily would stop parsing in the fourth pass if we found a valid parse tree at that time, we now keep on parsing until there are no more new matches to be found. In table 11.2 we present our parser's performance rate for the sentence *La bela knabo, kiom mi amas, malamas min.*

Pass	Accumulated number of iterations	Number of new matches found
1	491	10
2	1632	23
3	4430	33
4	10107	27
5	18980	16
6	29690	50
7	47344	80
8	74020	10
9	100700	0

Table 11.2: Iterations done when parsing the sentence *La bela knabo, kiom mi amas, malamas min.*

Over a hundred thousands iterations for just this one sentence seem like unnecessary inefficient, event though the actual runtime was less than six seconds. If further work shall be done on this project, something should be done to improve the performance of the parser.

11.2.5 Level-based parsing

One interesting solution is to introduce a level-based grammar. That can reduce the number of iterations through the parsing algorithm significantly.

In a level-based BNF grammar the rules are written so that they can be grouped into sections where the rules in a given section never refer to a rule in a section above their own. This mean that one can first apply all the rules in section one on a text. When no more matches are found, the rules in section one are put aside and the rules of section two are applied on the text. Then continue until the rules of the last section have been applied on the text and hopefully a parse tree has been found.

Let us illustrate this by the following example. We define a grammar for the small language *Bltest* in example 11.1.

BNF grammar forBltest

```
a = a b
a = b
b = c d d
b = d d
c = ‘‘habba’’
c = ‘‘zut’’
d = ‘‘foo’’
d = ‘‘bar’’
```

Example 11.1

Looking at table 11.3 on the facing page we can see that applied on the given sentence «*habba foo foo bar foo*» a standard bottom-up parser will need 3240 iterations. This is a parser implemented without any lookaheads at all, so there are room for improvements.

Level-based BNF grammar for Bltest

```
Level 3 : a = a b
Level 3 : a = b
Level 2 : b = c d d
Level 2 : b = d d
Level 1 : c = ‘‘habba’’
Level 1 : c = ‘‘zut’’
Level 1 : d = ‘‘foo’’
Level 1 : d= ‘‘bar’’
```

Example 11.2

Pass	Accumulated number of iterations	Number of new matches found
1	40	5
2	240	10
3	640	13
4	1136	18
5	1744	22
6	2432	16
7	2912	8
8	3120	6
9	3304	4
10	3240	0

Table 11.3: Iterations done when parsing the test sentence *habba foo foo bar foo* with a regular bottom-up parser.

In example 11.2 we have divided that same grammar into 3 levels. From table 11.4 we can see that the parser would now need 640 iterations on the first pass + 50 iterations on the second pass + 46 iterations on the third pass = 736 iterations.

Level	Pass	Accumulated number of iterations	Number of new matches found
1	1	20	5
	2	120	10
	3	320	10
	4	520	5
	5	620	1
	6	640	0
2	7	650	4
	8	680	2
	9	690	0
3	10	700	4
	11	720	4
	12	736	0

Table 11.4: Iterations done when parsing the test sentence *habba foo foo bar foo* with a level-based bottom-up parser.

11.2.6 Level-based parsing implemented in this project

We have actually implemented one aspect of level-based parsing. By separating all our terminals into a homogeneous environment we have achieved a grammar with two levels. The first one containing simple rules only referring to terminals, the second level containing rules with complex right-hand sides, though only referring to other rules, defined in either of the two levels.

We gained two things by this move.

1. The parsing algorithm got less complex, since we moved all the terminals into a level of their own.
2. The time performance improved substantially with this alteration alone. For sentences with more than 7 words, there was up to a 30% time reduction.

To write, or rewrite, the BNF grammar into a proper level-based grammar, is not an easy task. Many rules must be completely rewritten to avoid direct dependencies on other rules. Many assisting rules would need to be added to accomplish this. The direct readability of the grammar would diminish.

Modifying the parser so that it will be able to handle level-based grammars properly is a minor task, just a matter of adding a couple of tests and an additional loop in the algorithm.

The meta-BNF parser would need more work. It must be able to read the rules and after some algorithm detect and sort the dependencies so that the most efficient level-splitting is achieved.

11.2.7 Ambiguity between word classes

The implication of exceptions in the morphology

Looking back at section 6.1 on page 53 we recall that when there is uncertainty about which word class a word should be classified as, the morphology parser returns a list of all possibilities. As described in that section, the present way of dealing with this is that the syntax parser chooses the alternative first listed. Another possibility would be to make a choice based on the context. The most robust solution however, would be to run a full syntax parse for all the alternatives. This would also fit better into this projects

framework, since we have already stated that we are interested in *all* possible parse trees.

It is worth noting that when we run tests in section 7.1 on page 57, we never encountered this problem.

Proper nouns

A special case of ambiguity between word classes is detecting proper nouns. In Esperanto it will ideally follow the structure of a common noun, that is ending on -o. If that was attainable there would technically be no argument left for operating with the term *proper noun* since the subclass has completely merged together with its superclass.

Looking further than this and also picturing the stage where applications such as a translator is, we can see that it in many cases would be most useful to detect proper nouns so to know that these must be dealt with differently from ordinary nouns.

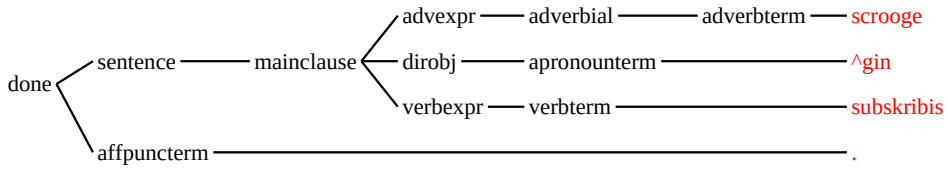

Figure 11.7: Parse tree of the sentence «Scrooge ĝin subskribis.»

Error in proper noun detection

```
input : Scrooge ^gin subskribis.

Matched rules :
==================
--- Word nr :          0
--- Word :             scrooge
--- Nr of rules :      6 adverbterm(2) adverbial(2) advexpr(2) mainclause(4)
                       sentence(4) done(-1)
------------------------------
--- Word nr :          1
--- Word :             ^gi
--- Nr of rules :      5 apronounterm(3) dirobj(3) mainclause(4) sentence(4)
                       done(-1)
------------------------------
```

```
--- Word nr :          2
--- Word :             subskribis
--- Nr of rules :      5 verbterm(4) verbexpr(4) mainclause(4) sentence(4)
                       done(-1)
------------------------------
--- Word nr :          3
--- Word :             .
--- Nr of rules :      3 fullstopterm(-1) punctuationterm(-1)
                       punctuationterm(-1)
------------------------------
```

Example 11.3

It is tempting to conclude that the above example has no errors; it finding a complete sentence structure that looks very plausible. However, this specific parse tree make no semantic sense, since "Scrooge" has mistakenly been taken for an adverb and not a proper noun.

Below (in figure 11.8) we can see how the parse tree from figure 11.7 example 11.3 should look.

Our problem in this specific sentence is that the proper noun is at the beginning of the sentence. As already described we determine the likelihood of a proper noun by looking for capital words. This approach only works for words not occurring as a first word in a sentence.

One approach to a solution would be to introduce a second pass in our morphology parser. (Or rather a first pass and degrade the current pass0o to second pass.) The first pass would then scan the text for words likely to be proper nouns and maintain them in a list. During the second pass, which would do the job our one pass does now, we would also check the morphemes against this list.

Another useful, although less automated, approach will be to give users the possibility to submit such a list as input. This could for instance be desirable when names only appear once in a given text.

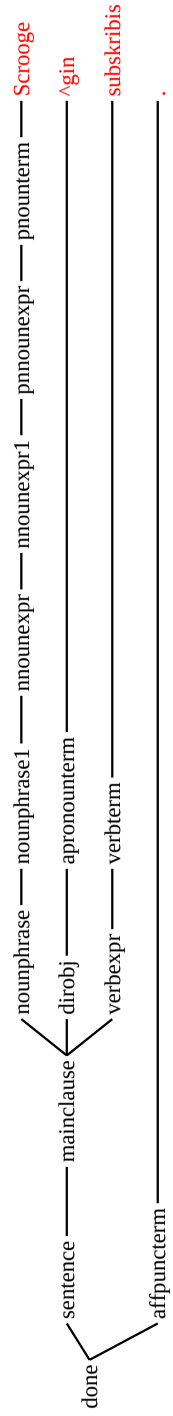

Figure 11.8: Corrected parse tree of the sentence «Scrooge ĝin subskribis.»

Chapter 12

Conclusion and further work

This thesis is based on a theory that suggests that the structure of Esperanto makes it suitable to be expressed in a formal grammar. It aims to find out whether this theory holds, or if Esperanto is as difficult to parse as ethnic languages are considered to be.

To test this, we have implemented a full morphology parser and a syntax parser. A meta BNF syntax is also introduced as well as meta BNF to EBNF converter. As much Esperanto as possible is then defined by such a meta BNF grammar.

We have had partial success. The morphology parser and the syntax parser both work as hoped and expected. The BNF grammar is, however, not complete, but we have concluded that this is, although huge, a possible task given enough resources. We do question whether this is desirable, since we conclude that it is not possible to write an unambiguous grammar for Esperanto, simply because Esperanto itself is not unambiguous.

A morphology parser was developed with a scanner module. This parser uses a combination of LR parsing and regular expressions to traverse each word in a text, checking the results against a dictionary. The morphology of Esperanto is hardcoded, so it is not applicable on other languages.

The thesis suggests and implements an extension to EBNF, giving it a lookahead functionality. The development of this metaBNF was crucial to the success of this project.

The thesis includes a grammar for Esperanto written in meta-BNF, although this does not describe the complete Esperanto language as of today.

The syntax parser is based on the design of a conventional bottom-up parser, but includes some support for levelbased parsing and explores ways of improving the performance of a standard bottom-up parser.

12.1 Suggestions for further work

Possible future areas to explore and improve might be:

- The BNF should include support for nonletter symbols.
- The BNF should include support for direct and indirect speech.
- The parser should handle Unicode-encoded texts.
- Better proper noun detection would improve the system.
- Rewrite the BNF grammar into a level-based grammar.

12.2 Where Esperanto goes wrong

It might be prententious of us, but we think it is useful to supply a short list of, in our opinion, unfortunate elements in Esperanto. These are features that don't work well in an automatic parsing situation.

If we look beyond our scope of automatic parsing, we could have written a much longer list, but language politics is not the purpose of this thesis.

1. Pronouns are essentially nouns when considered from a strictly syntax point of view. This would lead to the following conclusions:
   (a) They should follow the standard for contructing nouns, by using the word ending *o*. This would
      i. prevent the current ambiguity between personal pronouns and the verb infinitive.

ii. prevent the current ambiguity between the relative pronouns and the verb imperative. This would force a modification of some words, among them the word *kio* which already represents the meaning *what*, while *kiu* at the present represents *who*. We would think this could best be achieved by applying the use of an affix to derive the *individual* class from the *thing* class.

(b) The pronouns should be inflected for both case and number. This would cut the number of pronouns in half and make the morphology more regular.

2. All the correlatives should take on the proper morphological form based on their function, just as for the pronouns, see table 5.5 on page 51. The correlative classes needing change would be *individual*, *manner*, *reason*, *motion*, *time*, *amount* and *possession*.

3. The overlap between conjuctions and prepositions is an obscuring feature, and it certainly makes the task of automatic parsing a lot more difficult[1].

4. Affixes as roots. The morphological parsing of words would be easier if affixes could not appear as stems as well.

[1]Or a lot less easy.

Appendix A

Texts used for parsing

A.1 La kamelo kaj la arabo

This small text has been copied from [11].

> Arabo sidis en sia domo en la urbo. Apud domo trans la strato li vidis kamelon. La kamelo iris trans la straton ĝis la pordo, kaj diris al la arabo, Frato, mi ne estas varma, mi colas meti nur la nazon en vian varman domon. La arabo skuis la kapon, sed la kamelo metis la nazon tra la pordo en la ĉambron. La kamelo komencis puŝi sian tutan vizaĝon en la domon. Baldaŭ li havis la kapon ĝis la kolo en la domo. Post la kapo iris la kolo en ĝin, kaj baldaŭ la tuta korpo estis en la domo. La arabo estis kolera, ĉar li ne volis havi tutan kamelon en sia domo. Li kuris al la kamelo, kaptis lin, tenis lin, kaj diris, Frato, vi volis meti nur la nazon en mian domon. La ĉambro ne estas granda sed ĝi estas la mia, kaj mi preferas sidi en ĝi. Via diro estas bona, diris la kamelo, via domo ne estas granda, sed ĝi estas varma, kaj mi ŝxatas stari en ĝi. Mi preferas stari kaj kuŝi en ĝi, kaj mi donos al vi mian arbon trans la strato. Ĉu vi ne volas iri sub la arbon? Kaj la kamelo pusis la arabon de lia domo en la straton de la urbo. La kamelo nun trovis sin en varma ĉambro, sed la juna arabo staris trans la strato kaj ne estis varma.

A.2

This small text has been copied from [6].

Tra lando de indianoj

Ankoraŭ ne estis alveninta la tago, en kiu ni startus nian penetradon en direkto de la vojpinto. Kolonelo Vanique prokrastis la viziton, timante pro niaj vivoj. ĝis tiam ja pluraj ĵurnalistoj aperis tie - ĉiuj venintaj en aviadilo al la kampadejo - por aranĝi enketon inter la ekspediciuloj -, sed revenis al la civilizita mondo la saman tagon, publikigante en siaj ĵurnaloj fantaziajn raportojn pri siaj "aventuroj". Kio, tamen, plej konfuzis la ekspedician ĉefon, estis la ĉeesto, en tiu soleca regiono, de virino bonmaniera, de malfortika eksteraĵo, kaj eĉ pretendanta pasi Mortriveron kaj interniĝi ĝis la Sierra do Roncador (Sera do Ronkador). Promesinte al kolonelo Vanique ke ni prenos sur nin respondecon pri ĉio kio povus al ni okazi, li certigis al ni ke post kelkaj tagoj li mem nin akompanos. Ni utiligis tiun tempon por konatiĝi kun la vivo de la kampadejo kaj speciale kun la vivo en la praarbaro. Tiucele ni faris oftajn ekskursojn en la ĉirkaŭaĵo. En la budo de la radiotelegrafisto Alencar (Alenkar) ni rimarkis belegajn orkideojn, kiujn Meri ne ĉesis admiri. Ilia posedanto volis donaci al ni kelkajn, sed ni decidis ne akcepti tion kaj persone ekiri al la serĉo de orkideoj en la arbaro. Jen kiel ni fariĝis ĉasantoj de orkideoj. "Orkideo" estas magia vorto en la tuta mondo. En la salonoj ĝi signifas plejan distingitecon, kiel donaco, la plej delikatan atentemon, en la florvendejoj prezojn plej altajn. ĝia nomo estas universala. Nur la portugala faras escepton. "Parásita" oni ĝin nomas simple. Kaj kiam oni aŭdas tiun nomon el la buŝo de loĝanto de Matogroso, la fabela planto perdas sian tutan ĉarmon, reduktiĝante al simpla parazito, kiu kreskas kroĉita al la trunkoj kaj branĉoj de la arboj, donante, iam kaj tiam, florojn violkolorajn, blankajn aŭ flavajn, nek pli nek malpli belajn ol la plantoj ceteraj. Tiom pli mirinda prezentiĝas antaŭ la okuloj de la vizitanto de tiu regiono la unua orkideo, kiun li sukcesis mallevi de sur arbo: ĝiaj violkoloraj folioj etendiĝas kviete, dum la centro de la floro similas skulptaĵon elpensitan de frenezulo; ĝia parfumo elradias ebriigan dolĉon, rememorigante tempojn romantikajn. Li komparas tiun admirindan kreaĵon kun la orkideo

de urba florvendejo, enkorpigita al florvazo kaj sen ia odoro, perceptante distancon egan inter ambaŭ fratinoj. La mirinda naturo plenigas liajn okulojn, pulmojn, kaj la koro batas furioze. Jen la momento, kiam la homo forgesas siajn brakojn, sangantajn pro la arbdornojn, sian laciĝon, kaŭzitan de la fortostreĉo, kaj la pantalonon disŝiritan. Kaj jen estas ankaŭ la momento, kiam - se li havas fibron de aventuremo - li fariĝas orkideĉasanto. La orkideo kreskas, kaj sur la trunko mem, kaj sur la branĉoj plej altaj de la arbo, kvankam estas vero ke la plantoj malaltaj facile estas detruataj de la bestoj, kio klarigas ilian abundecon en alteco malfacile atingebla. Oni devas serĉi la orkideon kun la kapo supren levita. Kaj oni ne perdu tempon senutile: ni serĉu ĝin prefere sur arboj sulkŝelaj. Tie plej facile alkroĉiĝas la filigrane maldikaj radikoj, kiuj disetendiĝas ĉirkaŭ la trunko. La du folioj karnecaj kaj verdaj, kronantaj la ovalan akvujon, ankaŭ verdan, ne estas konfuzeblaj. Por trovi unu el tiuj plantoj necesas foje trairi centon da metroj de densarbaro. Kaj tio ne estas afero facila.

A.3 Other texts

And a few other texts that we won't include in full-text here:

- Kristnaska Kanto, Charles Dickens [15]
- La Venecia komercisto, William Shakespear

Appendix B

Lang file

```
# $Id: lang.rb,v 1.2 2005/01/24 17:55:09 benteaa Exp $
#
# Copyright (C) 2003 by Bente Christine Aasgaard <benteaa@ifi.uio.no>
#
#

$terminals = {}

$terminals['NNOUN'] =
  '
    begin
      true if ob.instance_of? Enoun and ob.casus == "nominative"
    end
  '

$terminals['PNOUN'] =
  '
    begin
      true if ob.instance_of? Eword
    end
  '

$terminals['NPRONOUN'] =
  '
    begin
      true if ob.instance_of? Epronoun and ob.casus == "nominative" and
        (ob.kind == "personal"  or ob.kind == "reflecsive")
    end
  '

$terminals['APRONOUN'] =
  '
```

```
    begin
      true if ob.instance_of? Epronoun and ob.casus == "accusative" and
        (ob.kind == "personal" or ob.kind == "reflecsive")
    end
  '

$terminals['RELPRONOUN'] =
  '
    begin
      true if ob.instance_of? Epronoun and ob.kind == "relative"
    end
  '

$terminals['NRELPRONOUN'] =
  '
    begin
      true if ob.instance_of? Epronoun and ob.kind == "relative" and
        ob.casus == "nominative"
    end
  '

$terminals['ARELPRONOUN'] =
  '
    begin
      true if ob.instance_of? Epronoun and ob.kind == "relative" and
        ob.casus == "accusative"
    end
  '

$terminals['POSSPRONOUN'] =
  '
    begin
      true if ob.instance_of? Epronoun and ob.kind == "possesive"
    end
  '

$terminals['NADJECTIVE'] =
  '
    begin
      true if ob.instance_of? Eadjective and ob.casus == "nominative"
    end
  '

$terminals['AADJECTIVE'] =
  '
    begin
      true if ob.instance_of? Eadjective and ob.casus == "accusative"
    end
  '
```

```
$terminals['ANOUN'] =
  '
    begin
      true if ob.instance_of? Enoun and ob.casus == "accusative"
    end
  '


$terminals['VERB'] =
  '
    begin
      true if ob.instance_of? Everb and ob.vtype == "indicative"
    end
  '


$terminals['VERBPRED'] =
  '
    begin
      true if ob.instance_of? Everb and ob.vtype == "predicative"
    end
  '

$terminals['VERBINF'] =
  '
    begin
      true if ob.instance_of? Everb and ob.vtype == "infinitive"
    end
  '

$terminals['VERBIMP'] =
  '
    begin
      true if ob.instance_of? Everb and ob.vtype == "imperative"
    end
  '

$terminals['ADVERB'] =
  '
    begin
      true if ob.instance_of? Eadverb
    end
  '
$terminals['JEADVERB'] =
  '
    begin
      true if ob.instance_of? Epreposition and ob.word == "je"
    end
```

```
  '

$terminals['MADVERB'] =
  '
    begin
      true if ob.instance_of? Eadverb and ob.word == "kiel"
    end
  '


$terminals['RELADVERB'] =
  '
    begin
      true if ob.instance_of? Eadverb and ob.kind == "relative"
    end
  '

$terminals['PLIADVERB'] =
  '
    begin
      true if ob.instance_of? Eadverb and ob.word == "pli"
    end
  '

$terminals['OLADVERB'] =
  '
    begin
      true if ob.instance_of? Eadverb and ob.word == "ol"
    end
  '

$terminals['DET'] =
  '
    begin
      true if ob.instance_of? Earticle
    end
  '


$terminals['CONCONJUNCTION'] =  # This is not good enough
  '
    begin
      true if ob.instance_of? Econjunction and ob.kind == "co"
    end
  '

$terminals['SUBCONJUNCTION'] =  # This is not good enough
  '
    begin
```

```
      true if ob.instance_of? Econjunction and ob.kind == "sub"
    end
  '

$terminals['KECONJUNCTION'] =
  '
    begin
      true if ob.instance_of? Econjunction  and ob.word == "ke"
    end
  '

$terminals['CUCONJUNCTION'] =
  '
    begin
      true if ob.instance_of? Econjunction  and ob.word == "^cu"
    end
  '

$terminals['PREP'] =
  '
    begin
      true if ob.instance_of? Epreposition
    end
  '

$terminals['PUNCTUATION'] =
  '
    begin
      true if ob.kind_of? Epunctuation
    end
  '

$terminals['FULLSTOP'] =
  '
    begin
      true if ob.instance_of? Efullstop
    end
  '

$terminals['EXCLAMATION'] =
  '
    begin
      true if ob.instance_of? Eexclamation
    end
  '


$terminals['QUESTION'] =
  '
```

```
    begin
      true if ob.instance_of? Equestion
    end
  '

$terminals['COMMA'] =
  '
    begin
      true if ob.instance_of? Ecomma
    end
  '

$terminals['INTERPUNCTUATION'] =
  '
    begin
      true if ob.kind_of? Epunctuation_interior
    end
  '

$terminals['EXPRESSION'] =
  '
    begin
      true if ob.kind_of? Eexpress
    end
  '


$terminals['AFFPUNCTUATION'] =
  '
    begin
      true if ob.instance_of? Efullstop or ob.instance_of? Eexclamation
    end
  '

def getBinding(ob)
  return binding
end


def matchTerminal (term, ob)
  unless $terminals.has_key?(term)
    puts "\nERROR : terminal #{term} isn't defined.\n"
    return false
  end
  # puts $terminals[term]
  puts ob.casus if term == "NOUN"
  return Kernel.eval($terminals[term], getBinding(ob))
end
```

Appendix C

Meta-BNF

```
%
%
full_sentence = sentence affpuncterm
full_sentence = bisentence affpuncterm
full_sentence = impsentence exclamationterm
full_sentence = expressterm exclamationterm
full_sentence = cuterm sentence questionterm
full_sentence = sentence commaterm bisentence affpuncterm
full_sentence = shortsentence affpuncterm
%
%
shortsentence = advexpr
shortsentence = nounphrase
shortsentence = shortsentence conconjunctionterm shortsentence
%
bisentencesub = bisentence commaterm
bisentence = bisentencesub? conconjunctionterm sentence
%
% Sentence
%
sentence = sentence interpuncterm subconjunctionterm? sentence
sentence = mainclause conconjunctionterm subconjunctionterm? sentence
sentence = mainclause
%
% Imperative sentences
%
impsentence = impverbterm
%
% Main clauses
%
mainclause = advexpr? nounphrase? advexpr? dirobj? verbexpr advexpr? dirobj? advexpr?
mainclause = advexpr? dirobj? advexpr? nounphrase? verbexpr advexpr?
```

```
mainclause = advexpr? verbexpr nounphrase? advexpr?
mainclause = advexpr? nounphrase? advexpr? predverbterm predexpr? advexpr?
%
% Predicative
%
predexpr = nounphrase
predexpr = degadj1
%
% Noun phrase
%
nounphrase = nounphrase nounphrasesub? conconjunctionterm nounphrase
nounphrasesub = commaterm nounphrase
nounphrase = nounphrase1
nounphrase1 = nounphrase1 napposition
nounphrase1 = nounphrase1 relapposition commaterm?
nounphrase1 = npronounterm
nounphrase1 = nnounexpr
%
% Noun expression
%
nnounexpr = nnounexpr prepexpr
nnounexpr = nnounexpr pnnounexpr
nnounexpr = detterm nnounexpr1
nnounexpr = nnounexpr1
nnounexpr = nrelpronterm
nnounexpr = possesiveterm nnounexpr1
nnounexpr = nrelpronterm nnounexpr1
nnounexpr1 = nadjexpr pnnounexpr
nnounexpr1 = nadjexpr
nnounexpr1 = pnnounexpr nadjexpr
nnounexpr1 = pnnounexpr
pnnounexpr = nnounterm
pnnounexpr = pnounterm
%
% Adjective
%
nadjexpr = adverbterm nadjexpr
nadjexpr = nadjectiveterm nadjexpr
nadjexpr = nadjectiveterm
aadjexpr = aadjectiveterm aadjexpr
aadjexpr = aadjectiveterm
degadj1 = pliadvterm nadjectiveterm
degadj2 = oladvterm nadjectiveterm
degadj1 = degadj1 degadj2
%
% Apposition
%
apposition = napposition
apposition = aapposition
```

```
napposition = commaterm nounphrase napposition
napposition = commaterm nounphrase #punctuationterm
napposition = madverbterm nounphrase #punctuationterm
aapposition = commaterm dirobj aapposition
aapposition = commaterm dirobj #punctuationterm
relapposition = commaterm relpronterm sentence
relapposition = commaterm reladvterm sentence
%
%
% Direct object
%
commaobj = commaterm dirobj
dirobj = dirobj commaobj? conconjunctionterm dirobj
dirobj = infverb? objexpr
dirobj = apronounterm
dirobj = infverb reldirobj?
dirobj = reldirobj
reldirobj = commaterm keconjterm sentence
objexpr = objexpr prepexpr
objexpr = objexpr apposition
objexpr = possesiveterm objexpr1
objexpr = arelpronterm objexpr1?
objexpr = detterm objexpr1
objexpr = objexpr1
objexpr1 = aadjexpr anounterm
objexpr1 =  anounterm
%
% Prepositional expression
%
prepexpr = prepterm nounphrase1
prepexpr = prepterm dirobj
prepexpr = prepterm infverb
%
% Verbal
verbexpr = verbterm
%
% Infverb
infverb = infverbterm
infverb = infverbterm dirobj
infverb = dirobj infverbterm
infverb = infverbterm objexpr advexpr
infverb = infverbterm advexpr
%
%
% Adverbial
% advexpr = helpproduction, to make it possible for several
%           adverbials to come after one another
%
advexpr = adverbial advexpr
```

```
advexpr = adverbial
adverbial = prepexpr
adverbial = jeadverbterm
adverbial = adverbterm
%
%
%
%
%
% Simple terminal rules
%
nnounterm = NNOUN
anounterm = ANOUN
pnounterm = PNOUN
npronounterm = NPRONOUN
relpronterm = RELPRONOUN
nrelpronterm = NRELPRONOUN
arelpronterm = ARELPRONOUN
apronounterm = APRONOUN
detterm = DET
nadjectiveterm = NADJECTIVE
aadjectiveterm = AADJECTIVE
verbterm = VERB
predverbterm = VERBPRED
infverbterm = VERBINF
impverbterm = VERBIMP
adverbterm = ADVERB
reladvterm = RELADVERB
madverbterm = MADVERB
jeadverbterm = JEADVERB
prepterm = PREP
conconjunctionterm = CONCONJUNCTION
subconjunctionterm = SUBCONJUNCTION
punctuationterm = PUNCTUATION
commaterm = COMMA
affpuncterm = AFFPUNCTUATION
interpuncterm = INTERPUNCTUATION
fullstopterm = FULLSTOP
exclamationterm =  EXCLAMATION
questionterm =  QUESTION
keconjterm = KECONJUNCTION
possesiveterm = POSSPRONOUN
pliadvterm = PLIADVERB
oladvterm = OLADVERB
cuterm = CUCONJUNCTION
expressterm = EXPRESSION
```

Appendix D

Expanded BNF

```
Number of left side rules : 67
Number of right side rules : 268
Number of terminals : 35

aadjectiveterm  [1]
        0 [1] AADJECTIVE
aadjexpr        [2]
        0 [2] aadjectiveterm aadjexpr
        1 [1] aadjectiveterm
aapposition     [2]
        0 [3] commaterm dirobj aapposition
        1 [2] commaterm dirobj
adverbial       [3]
        0 [1] prepexpr
        1 [1] jeadverbterm
        2 [1] adverbterm
adverbterm      [1]
        0 [1] ADVERB
advexpr         [2]
        0 [2] adverbial advexpr
        1 [1] adverbial
affpuncterm     [1]
        0 [1] AFFPUNCTUATION
anounterm       [1]
        0 [1] ANOUN
apposition      [2]
        0 [1] napposition
        1 [1] aapposition
apronounterm    [1]
        0 [1] APRONOUN
arelpronterm    [1]
        0 [1] ARELPRONOUN
```

```
bisentence      [2]
        0 [3] bisentencesub conconjunctionterm sentence
        1 [2] conconjunctionterm sentence
bisentencesub   [1]
        0 [2] bisentence commaterm
commaobj        [1]
        0 [2] commaterm dirobj
commaterm       [1]
        0 [1] COMMA
conconjunctionterm      [1]
        0 [1] CONCONJUNCTION
cuterm  [1]
        0 [1] CUCONJUNCTION
degadj1         [2]
        0 [2] pliadvterm nadjectiveterm
        1 [2] degadj1 degadj2
degadj2         [1]
        0 [2] oladvterm nadjectiveterm
detterm         [1]
        0 [1] DET
dirobj  [8]
        0 [4] dirobj commaobj conconjunctionterm dirobj
        1 [3] dirobj conconjunctionterm dirobj
        2 [2] infverb objexpr
        3 [1] objexpr
        4 [1] apronounterm
        5 [2] infverb reldirobj
        6 [1] infverb
        7 [1] reldirobj
exclamationterm         [1]
        0 [1] EXCLAMATION
expressterm     [1]
        0 [1] EXPRESSION
full_sentence   [7]
        0 [2] sentence affpuncterm
        1 [2] bisentence affpuncterm
        2 [2] impsentence exclamationterm
        3 [2] expressterm exclamationterm
        4 [3] cuterm sentence questionterm
        5 [4] sentence commaterm bisentence affpuncterm
        6 [2] shortsentence affpuncterm
fullstopterm    [1]
        0 [1] FULLSTOP
impsentence     [1]
        0 [1] impverbterm
impverbterm     [1]
        0 [1] VERBIMP
infverb         [5]
        0 [1] infverbterm
```

```
        1 [2] infverbterm dirobj
        2 [2] dirobj infverbterm
        3 [3] infverbterm objexpr advexpr
        4 [2] infverbterm advexpr
infverbterm     [1]
        0 [1] VERBINF
interpuncterm   [1]
        0 [1] INTERPUNCTUATION
jeadverbterm    [1]
        0 [1] JEADVERB
keconjterm      [1]
        0 [1] KECONJUNCTION
madverbterm     [1]
        0 [1] MADVERB
mainclause      [140]
        0 [2] advexpr verbexpr
        1 [2] nounphrase verbexpr
        2 [3] advexpr nounphrase verbexpr
        3 [3] advexpr advexpr verbexpr
        4 [3] nounphrase advexpr verbexpr
        5 [4] advexpr nounphrase advexpr verbexpr
        6 [2] dirobj verbexpr
        7 [3] advexpr dirobj verbexpr
        8 [3] nounphrase dirobj verbexpr
        9 [4] advexpr nounphrase dirobj verbexpr
        10 [4] advexpr advexpr dirobj verbexpr
        11 [4] nounphrase advexpr dirobj verbexpr
        12 [5] advexpr nounphrase advexpr dirobj verbexpr
        13 [2] verbexpr advexpr
        14 [3] advexpr verbexpr advexpr
        15 [3] nounphrase verbexpr advexpr
        16 [4] advexpr nounphrase verbexpr advexpr
        17 [4] advexpr advexpr verbexpr advexpr
        18 [4] nounphrase advexpr verbexpr advexpr
        19 [5] advexpr nounphrase advexpr verbexpr advexpr
        20 [3] dirobj verbexpr advexpr
        21 [4] advexpr dirobj verbexpr advexpr
        22 [4] nounphrase dirobj verbexpr advexpr
        23 [5] advexpr nounphrase dirobj verbexpr advexpr
        24 [5] advexpr advexpr dirobj verbexpr advexpr
        25 [5] nounphrase advexpr dirobj verbexpr advexpr
        26 [6] advexpr nounphrase advexpr dirobj verbexpr advexpr
        27 [2] verbexpr dirobj
        28 [3] advexpr verbexpr dirobj
        29 [3] nounphrase verbexpr dirobj
        30 [4] advexpr nounphrase verbexpr dirobj
        31 [4] advexpr advexpr verbexpr dirobj
        32 [4] nounphrase advexpr verbexpr dirobj
        33 [5] advexpr nounphrase advexpr verbexpr dirobj
```

```
34 [3] dirobj verbexpr dirobj
35 [4] advexpr dirobj verbexpr dirobj
36 [4] nounphrase dirobj verbexpr dirobj
37 [5] advexpr nounphrase dirobj verbexpr dirobj
38 [5] advexpr advexpr dirobj verbexpr dirobj
39 [5] nounphrase advexpr dirobj verbexpr dirobj
40 [6] advexpr nounphrase advexpr dirobj verbexpr dirobj
41 [3] verbexpr advexpr dirobj
42 [4] advexpr verbexpr advexpr dirobj
43 [4] nounphrase verbexpr advexpr dirobj
44 [5] advexpr nounphrase verbexpr advexpr dirobj
45 [5] advexpr advexpr verbexpr advexpr dirobj
46 [5] nounphrase advexpr verbexpr advexpr dirobj
47 [6] advexpr nounphrase advexpr verbexpr advexpr dirobj
48 [4] dirobj verbexpr advexpr dirobj
49 [5] advexpr dirobj verbexpr advexpr dirobj
50 [5] nounphrase dirobj verbexpr advexpr dirobj
51 [6] advexpr nounphrase dirobj verbexpr advexpr dirobj
52 [6] advexpr advexpr dirobj verbexpr advexpr dirobj
53 [6] nounphrase advexpr dirobj verbexpr advexpr dirobj
54 [7] advexpr nounphrase advexpr dirobj verbexpr advexpr dirobj
55 [3] verbexpr advexpr advexpr
56 [4] advexpr verbexpr advexpr advexpr
57 [4] nounphrase verbexpr advexpr advexpr
58 [5] advexpr nounphrase verbexpr advexpr advexpr
59 [5] advexpr advexpr verbexpr advexpr advexpr
60 [5] nounphrase advexpr verbexpr advexpr advexpr
61 [6] advexpr nounphrase advexpr verbexpr advexpr advexpr
62 [4] dirobj verbexpr advexpr advexpr
63 [5] advexpr dirobj verbexpr advexpr advexpr
64 [5] nounphrase dirobj verbexpr advexpr advexpr
65 [6] advexpr nounphrase dirobj verbexpr advexpr advexpr
66 [6] advexpr advexpr dirobj verbexpr advexpr advexpr
67 [6] nounphrase advexpr dirobj verbexpr advexpr advexpr
68 [7] advexpr nounphrase advexpr dirobj verbexpr advexpr advexpr
69 [3] verbexpr dirobj advexpr
70 [4] advexpr verbexpr dirobj advexpr
71 [4] nounphrase verbexpr dirobj advexpr
72 [5] advexpr nounphrase verbexpr dirobj advexpr
73 [5] advexpr advexpr verbexpr dirobj advexpr
74 [5] nounphrase advexpr verbexpr dirobj advexpr
75 [6] advexpr nounphrase advexpr verbexpr dirobj advexpr
76 [4] dirobj verbexpr dirobj advexpr
77 [5] advexpr dirobj verbexpr dirobj advexpr
78 [5] nounphrase dirobj verbexpr dirobj advexpr
79 [6] advexpr nounphrase dirobj verbexpr dirobj advexpr
80 [6] advexpr advexpr dirobj verbexpr dirobj advexpr
81 [6] nounphrase advexpr dirobj verbexpr dirobj advexpr
82 [7] advexpr nounphrase advexpr dirobj verbexpr dirobj advexpr
```

```
83 [4] verbexpr advexpr dirobj advexpr
84 [5] advexpr verbexpr advexpr dirobj advexpr
85 [5] nounphrase verbexpr advexpr dirobj advexpr
86 [6] advexpr nounphrase verbexpr advexpr dirobj advexpr
87 [6] advexpr advexpr verbexpr advexpr dirobj advexpr
88 [6] nounphrase advexpr verbexpr advexpr dirobj advexpr
89 [7] advexpr nounphrase advexpr verbexpr advexpr dirobj advexpr
90 [5] dirobj verbexpr advexpr dirobj advexpr
91 [6] advexpr dirobj verbexpr advexpr dirobj advexpr
92 [6] nounphrase dirobj verbexpr advexpr dirobj advexpr
93 [7] advexpr nounphrase dirobj verbexpr advexpr dirobj advexpr
94 [7] advexpr advexpr dirobj verbexpr advexpr dirobj advexpr
95 [7] nounphrase advexpr dirobj verbexpr advexpr dirobj advexpr
96 [8] advexpr nounphrase advexpr dirobj verbexpr advexpr dirobj advexpr
97 [1] verbexpr
98 [3] dirobj advexpr verbexpr
99 [4] advexpr dirobj advexpr verbexpr
100 [3] dirobj nounphrase verbexpr
101 [4] advexpr dirobj nounphrase verbexpr
102 [4] advexpr advexpr nounphrase verbexpr
103 [4] dirobj advexpr nounphrase verbexpr
104 [5] advexpr dirobj advexpr nounphrase verbexpr
105 [4] dirobj advexpr verbexpr advexpr
106 [5] advexpr dirobj advexpr verbexpr advexpr
107 [4] dirobj nounphrase verbexpr advexpr
108 [5] advexpr dirobj nounphrase verbexpr advexpr
109 [5] advexpr advexpr nounphrase verbexpr advexpr
110 [5] dirobj advexpr nounphrase verbexpr advexpr
111 [6] advexpr dirobj advexpr nounphrase verbexpr advexpr
112 [1] verbexpr
113 [2] verbexpr nounphrase
114 [3] advexpr verbexpr nounphrase
115 [3] verbexpr nounphrase advexpr
116 [4] advexpr verbexpr nounphrase advexpr
117 [1] verbexpr
118 [2] predverbterm predexpr
119 [3] advexpr predverbterm predexpr
120 [3] nounphrase predverbterm predexpr
121 [4] advexpr nounphrase predverbterm predexpr
122 [4] advexpr advexpr predverbterm predexpr
123 [4] nounphrase advexpr predverbterm predexpr
124 [5] advexpr nounphrase advexpr predverbterm predexpr
125 [2] predverbterm advexpr
126 [3] advexpr predverbterm advexpr
127 [3] nounphrase predverbterm advexpr
128 [4] advexpr nounphrase predverbterm advexpr
129 [4] advexpr advexpr predverbterm advexpr
130 [4] nounphrase advexpr predverbterm advexpr
131 [5] advexpr nounphrase advexpr predverbterm advexpr
```

```
        132 [3] predverbterm predexpr advexpr
        133 [4] advexpr predverbterm predexpr advexpr
        134 [4] nounphrase predverbterm predexpr advexpr
        135 [5] advexpr nounphrase predverbterm predexpr advexpr
        136 [5] advexpr advexpr predverbterm predexpr advexpr
        137 [5] nounphrase advexpr predverbterm predexpr advexpr
        138 [6] advexpr nounphrase advexpr predverbterm predexpr advexpr
        139 [1] predverbterm
nadjectiveterm  [1]
        0 [1] NADJECTIVE
nadjexpr        [3]
        0 [2] adverbterm nadjexpr
        1 [2] nadjectiveterm nadjexpr
        2 [1] nadjectiveterm
napposition     [3]
        0 [3] commaterm nounphrase napposition
        1 [2] commaterm nounphrase
        2 [2] madverbterm nounphrase
nnounexpr       [7]
        0 [2] nnounexpr prepexpr
        1 [2] nnounexpr pnnounexpr
        2 [2] detterm nnounexpr1
        3 [1] nnounexpr1
        4 [1] nrelpronterm
        5 [2] possesiveterm nnounexpr1
        6 [2] nrelpronterm nnounexpr1
nnounexpr1      [4]
        0 [2] nadjexpr pnnounexpr
        1 [1] nadjexpr
        2 [2] pnnounexpr nadjexpr
        3 [1] pnnounexpr
nnounterm       [1]
        0 [1] NNOUN
nounphrase      [3]
        0 [4] nounphrase nounphrasesub conconjunctionterm nounphrase
        1 [3] nounphrase conconjunctionterm nounphrase
        2 [1] nounphrase1
nounphrase1     [5]
        0 [2] nounphrase1 napposition
        1 [3] nounphrase1 relapposition commaterm
        2 [2] nounphrase1 relapposition
        3 [1] npronounterm
        4 [1] nnounexpr
nounphrasesub   [1]
        0 [2] commaterm nounphrase
npronounterm    [1]
        0 [1] NPRONOUN
nrelpronterm    [1]
        0 [1] NRELPRONOUN
```

```
objexpr         [7]
        0 [2] objexpr prepexpr
        1 [2] objexpr apposition
        2 [2] possesiveterm objexpr1
        3 [2] arelpronterm objexpr1
        4 [1] arelpronterm
        5 [2] detterm objexpr1
        6 [1] objexpr1
objexpr1        [2]
        0 [2] aadjexpr anounterm
        1 [1] anounterm
oladvterm       [1]
        0 [1] OLADVERB
pliadvterm      [1]
        0 [1] PLIADVERB
pnnounexpr      [2]
        0 [1] nnounterm
        1 [1] pnounterm
pnounterm       [1]
        0 [1] PNOUN
possesiveterm   [1]
        0 [1] POSSPRONOUN
predexpr        [2]
        0 [1] nounphrase
        1 [1] degadj1
predverbterm    [1]
        0 [1] VERBPRED
prepexpr        [3]
        0 [2] prepterm nounphrase1
        1 [2] prepterm dirobj
        2 [2] prepterm infverb
prepterm        [1]
        0 [1] PREP
punctuationterm         [1]
        0 [1] PUNCTUATION
questionterm    [1]
        0 [1] QUESTION
reladvterm      [1]
        0 [1] RELADVERB
relapposition   [2]
        0 [3] commaterm relpronterm sentence
        1 [3] commaterm reladvterm sentence
reldirobj       [1]
        0 [3] commaterm keconjterm sentence
relpronterm     [1]
        0 [1] RELPRONOUN
sentence        [5]
        0 [4] sentence interpuncterm subconjunctionterm sentence
        1 [3] sentence interpuncterm sentence
```

```
        2 [4] mainclause conconjunctionterm subconjunctionterm sentence
        3 [3] mainclause conconjunctionterm sentence
        4 [1] mainclause
shortsentence   [3]
        0 [1] advexpr
        1 [1] nounphrase
        2 [3] shortsentence conconjunctionterm shortsentence
subconjunctionterm      [1]
        0 [1] SUBCONJUNCTION
verbexpr        [1]
        0 [1] verbterm
verbterm        [1]
        0 [1] VERB
```

Bibliography

[1] **Trends in Linguistics, studies and Monographs No 42 : Interlinguistics - Aspects of the Science of Planned Languages**
Klaus Schubert
Mouton de Gruyter, 1989

[2] **Flerspråklig Informasjonssenter**
`http://www.esperanto.net`

[3] **Pilot Implementation of a Bilingual Knowledge Bank. In Proc. of the 13th International Conf. on Computational Linguistics**
V. Sadler and R. Vendelman
Helsinki, 1990, 449-451

[4] **Compilers**
Principles, Techniques and Tools
Alfred V. Aho, Ravi Seti and Jeffrey D. Ullman

[5] **Homepage of EOparser**
`http://www.germane-software.com/software/Utilities/EOParse/`

[6] **Swedish Esperanto Association** `http://www.esperanto.se/`

[7] **The definition of EBNF**
`http://www.cl.cam.ac.uk/~mgk25/iso-ebnf.html`

[8] **Two Level Morphology of Esperanto** Jirí Hanaa
`http://www.ling.ohio-state.edu/~hana/esr/thesis.pdf`

[9] **The Esperanto alphabet**
`http://steve-and-pattie.com/esperantujo/alphabet.html`

[10] **Machine Translation**
John Hutchins and Evgenii Lovtskii

[11] **A Complete Grammar of Esperanto**
Ivy Kellerman Reed
http://www.gutenberg.org/etext/7787

[12] **Modern Irish**
Micheal O Siadhail
Cambridge University Press
ISBN 0521425190

[13] **En syntaks for datamaskinell analyse av esperanto**
Siv Sjögren

[14] **Two-level morphology: A general computational model for word-form recognition and production** Kimmo Koskenniemi
University of Helsinki, Department of General Linguistics
ISBN 9514532015

[15] **Kristnaska kanto**
Charles Dickens
Lingve redaktis Margaret MUNROW
Tyresö: Inko, 2000
ISBN 91-7303-071-6

[16] **The Handbook of Linguistics**
Mark Aronoff (Editor), Janie Rees-Miller (Editor)
Blackwell Publishers
ISBN 1405102527