

A CHR_G Model of the Ancient Egyptian Grammar

Thomas Hecksher
Sune T. B. Nielsen
Alexis Pigeon

Supervisor : Henning Christiansen

Student project report
Computer Science Dept., Roskilde University, Denmark

December 20, 2002

Abstract

This report describes and discusses the experiment of writing a grammar of ancient Egyptian, with focus on the the modelling aspect and the formalism used. A grammar or in other words a program was developed using the Constraint Handling Rules (CHR) based grammar formalism CHR-Grammars (CHRG, developed by Henning Christiansen). When the report was written, its writers were new to the field of Natural Language Processing (NLP) and at the same time the entering upon a relatively unexplored area: the application of NLP to ancient Egyptian. Especially with respect to the application of CHRG something new is done, since the rather new formalism have never before been used for NLP.

Put simple the goal was to create an extendible and general model of ancient Egyptian grammar. It should be able to verify whether a list of 'signs' (i.e. representations of hieroglyphic signs in the style: A1, B15, B20) can be recognised as a correct sentence according to the model of the ancient Egyptian grammar formulated in the program.

Contents

1	Introduction	5
1.1	Motivation	5
1.2	Reading Guidelines	5
1.2.1	Notational Conventions	6
1.3	First Part of the Problem: The Ancient Egyptian Language	8
1.3.1	History and Different Stages of the Ancient Egyptian Language	8
1.3.2	Middle Egyptian	9
1.3.3	Static Corpus	9
1.4	Second Part of the Problem: The Grammar and the Tools	10
1.4.1	Field of Focus	10
1.4.2	Conceptual Requirements	11
1.5	Methodology	13
1.5.1	The Formalism	13
1.5.2	Writing Our Grammar	14
1.6	Literature	14
1.6.1	Ancient Egyptian	14
1.6.2	Prolog, CHR _G , NLP etc.	15
1.7	Overview of the Report	15
2	CHR Grammars	16
2.1	Introduction to CHR _G by example	17
2.2	Other Interesting Features of CHR _G	18
2.2.1	Context sensitive rules	18
2.2.2	Simplification	19
2.2.3	Guards	19
2.2.4	Optional grammar symbols in head of rules	19
3	Ancient Egyptian grammar and the interesting modelling problems	21
3.1	The signs of ancient Egyptian	21
3.2	Direction of writing	22
3.3	Arrangements of signs	23

3.4	No clear separation of words	23
3.5	No punctuation marks	23
3.6	Different ways of spelling the same word	24
3.7	Omission of a subject in a sentence	24
3.8	Structure of ancient Egyptian sentences	25
4	Modelling the Language	26
4.1	Introduction	26
4.2	Overview of the Program	26
4.3	The Lexicon	27
4.3.1	Spelling the words	27
4.3.2	Word classes	28
4.4	The Morphological Rules	29
4.4.1	Recognition of inflections	29
4.4.2	Morphology	30
4.5	Constituent Properties Determination Rules	30
4.6	The Syntax Rules	32
4.7	Conjunction	32
4.8	The Ambiguity caused by the C15 sign	34
4.9	The Implied Subject	35
5	Experimental Results	39
5.1	Sentence Recognition	39
5.2	Word Properties Classification Rules	40
5.3	Syntax Rules	41
6	Discussion	42
6.1	NPPP vs. NP(di)	42
6.1.1	Pros And Cons Of Each Version	44
6.2	Conceptual Requirements Of The Program	44
6.2.1	Next Step Implementations	44
6.2.2	Exceptions Handling	45
6.2.3	Readability	46
6.2.4	Generality of our Solutions	46
6.2.5	Noise Reduction	47
6.2.6	Scaling Up	48
6.3	Conceptual Requirements Of The Formalism	48
6.3.1	Readability	48
6.3.2	Limitations to CHRGE	49
7	Conclusion	50
7.1	Perspective	51
8	Dictionary	53

Chapter 1

Introduction

1.1 Motivation

The writing of this report started with an interest in, but only little knowledge of, Natural Language Processing (**NLP**). What we wanted to know from the beginning, is contained in the following questions:

What are the general problems in modelling any natural language?

What would be the key design decisions when creating a general model of a natural language grammar?

As our starting point, we wanted to do a program handling ancient Egyptian grammar. We would like to point out that it is clear from the above questions the interest in the ancient Egyptian language has not been leading in the process or the formulation of the focus of the report. But nevertheless this has turned out to pose some quite interesting problems, which we would possibly not have come across if we had chosen Danish, English, French or German which would have been the most obvious candidates for a living language to model¹.

Together with the interest in NLP, we also initially also wanted to get (more) familiar with logic programming (i.e. Prolog).

1.2 Reading Guidelines

The goal of this report is somehow to reflect the process, and not least, the result of the process of getting into the field of logic programming and NLP, which in our case mainly consists of a combination of four areas: Logic

¹The project group consists of one native French speaking person and two native Danish speaking.

Programming, Constraint Programming, and to some extent Linguistics – especially syntax – and Egyptology. This report will not presuppose any knowledge on Egyptology, grammar or linguistics, while at least some basic notion of Prolog and constraint programming can be helpful. We picture that readers of this report might be people on the same educational level as ourselves, with initial interest in NLP and Prolog programming.

Throughout the report we use a lot of illustrative examples taken from English. We have done so in order to make our points clearer than they would be using Egyptian examples.

In the end of the report we have compiled in a small dictionary explanations and definitions of advanced terms that are used in the report. Abbreviations will be explained the first time they are used.


Terminology: 'Grammar'

A word that is very central to this report is the word 'grammar'. It has a long history and a variety of different uses, which we will not go through here. But we will mention and define the ways we understand the term.

'Grammar' can mean an already existing entity or it can mean a description of an existing entity. Firstly it can be a **description** of something already existing (or something that ought to exist), according to a traditional distinction² between on the one hand descriptive grammar which is a description of (the use of) a language and on the other hand prescriptive or normative grammar describing the correct use. That is so to say a distinction between 'how-it-is' and 'how-it-ought-to-be'. We are of course seeing ourselves as doing descriptive grammar. In the following, this is the understanding of the term we refer to when saying 'our grammar' (used interchangeably with 'our program', 'the code' and the like).

Secondly 'grammar' can be something already existing: "*A level of structural organization which can be studied independently of phonology and semantics, generally divided into the branches of **syntax** and **morphology**.*"³ (accentuation as in original). This is the meaning of the term we use when we refer to 'the ancient Egyptian grammar'. This meaning is underlined by the use of the term 'model' which will be seen a lot throughout the report.

1.2.1 Notational Conventions

Needless to say that a  key does not exist on any keyboard. In order to compute hieroglyphs, we had to choose a way to code each sign.

²Taken from [Cry92], p. 138.

³Cf. [Cry92] p. 138.

There is no final agreement upon standard for representing hieroglyphs by names spelled with letters and numbers (e.g. F34, B93, A11 etc.). We do not know the reasons and discussions lying behind this fact, but we observe that there are many different notations (cf. [Ned02]) and that most of those we have encountered within our reach are very similar, not least in the way of categorising the hieroglyphs.

We came across two representations:

- One established by Sir Alan Gardiner [Gar27] in 1927.
- Another established quite more recently, by Mark Collier and Bill Manley [CM98].

In both representations, a hieroglyph is represented by a combination of a letter and a number. The letter classifies a sign according to what it represents : a human being, an animal, a plant, etc. The number indexes the sign into this picture category.

The former seems to be a standard most Egyptologists agree upon. Many works, like *Hieroglyphica* [GHvdP93] or the HieroTeX fonts [Ros02] are based on this representation. Moreover, among the representations, it appears to be the most comprehensive one.

However, we chose the latter, since we have built our grammar upon the rather simple description and the extensive mapping between signs, meanings, translations and transliterations used in '*How to read Egyptian hieroglyphs*' [CM98].

The result is that our representation may not appear the most familiar to some Egyptologists. But the computed representation of the hieroglyphs could be changed easily by a simple conversion in the code if needed.

The most extensive one we have come across is Hieroglyphica ([GHvdP93]) and it seems to be a development of Gardiners sign list (Cf. [Gar27] pp. 438 ff.). Throughout this report we simply use the hieroglyphic signs, while in the program we use the standard of Collier and Manley (in [CM98])⁴.

To this might be added that it is very common in literature concerning ancient Egyptian to use transliteration (a notational system where each hieroglyphic sign is represented by a western-like letter. This has not at all been relevant to our work and we have left it out of account.

⁴Since we actually are not using many signs due to our small lexicon, we regard the comments in the program code suffice to explain each sign. In some examples we have found it useful though to also add this notation. For full reference on this system we point to [CM98]

1.3 First Part of the Problem: The Ancient Egyptian Language

In order to get started with the perhaps most unfamiliar part of this report and in order to narrow down our focus regarding this part, we start out by giving a brief overview of some general facts about the ancient Egyptian language and its history.

1.3.1 History and Different Stages of the Ancient Egyptian Language

Being one of the earliest scripts - dating back to at least 3000 B.C., Hieroglyphic writing has not been used for about 2000 years⁵. Ancient Egyptian is now regarded as a dead language, which means that no group of people have spoken it as their primary language⁶. Until 1799, where the so-called Rosetta stone was found by French soldiers during the foundation of the fortress in Rosetta⁷, few of the known hieroglyphic texts were understood. The Rosetta stone (now residing in the British Museum, in London) is a stone with the same, relatively long text written in three scripts: Greek, Demotic⁸ and Hieroglyphic. A considerable part of the hieroglyphic text is missing, but still – through the demotic text – the Rosetta stone was a, if not **the**, central key to the understanding the hieroglyphic writing.

Talking about 'ancient Egyptian' or 'ancient Egypt' is not very precise and we may not need very much accuracy, since we are only looking at the writing system. But since there were different systems of writing and since also the hieroglyphic writing was subject to changes and not least to provide a minimum of background knowledge, we find it worth mentioning that the history of the ancient Egyptian language is roughly divided into several periods. Some reservations are of course necessary⁹, the most important of which we see as the fact that written and spoken language has developed differently and in different tempi. Sir Gardiner ([Gar27]) mentions the following periods, also counting in the different Dynasties:

Old Egyptian (Dynasties I-VIII) 3180 to 2240 B.C.

Middle Egyptian (Dynasties IX-XI) 2240 to 1990 B.C.

Late Egyptian (Dynasties XVIII-XXIV) 1573 to 715 B.C.

⁵The latest text being from A.D. 394 (Cf. [Gar27], p. 1.).

⁶For a similar definition, see [Cry92].

⁷Cf. [Gar27], p. 12.

⁸Demotic writing refers to one of the three kinds of script that were developed in ancient Egypt. "Out of hieroglyphic sprang a more cursive writing known to us as **hieratic**, and out of hieratic again there emerged, towards 700 B.C., a very rapid script [...] known as **demotic**." [Gar27], p. 9.

⁹Cf. [Gar27], p. 5.

Demotic (Dynasties XXV-?) 715 B.C. to A.D. 470.
Coptic A.D. 400 to 640.¹⁰

1.3.2 Middle Egyptian

Our focus is Middle Egyptian solely, since this stage this is a good point to start from when learning ancient Egyptian and since most works written as introductions to ancient Egyptian, are dealing with Middle Egyptian. Middle Egyptian is defined by [Gar27] as the "*...idiom employed in the stories and other literary compositions of the Middle Kingdom (Dynasties IX-XIII, roughly from 2240 to 1740 B.C.) as well as public and private monumental inscriptions of that period and also far down to the Eighteenth Dynasty (1573-1314 B.C.)*."¹¹ As nearly indicated in the passage cited here, one of the reasons that Middle Egyptian is recommended as starting point, is that there is a considerable amount of different texts that were written in this idiom. This makes it more appealing to laymen to choose between reading literary, magical, religious, historical, mathematical and legal texts than having to read e.g. business documents. But a far more important reason for starting out with Middle Egyptian and not one of the other stages is given by [Gar27] "*it is more consistently spelt than other phases of the language, and it is in this phase that the inflections of the verb are best displayed in writing*."¹² Throughout the report we use the term 'ancient Egyptian' instead of the more precise 'Middle Egyptian Hieroglyphic writing'.

1.3.3 Static Corpus

One definition of 'grammar' (the chomskyan approach) finds that a grammar is a finite formulation of a language where an infinite number of sentences are possible, given that sentences can be of any length. But since the ancient Egyptian language is a dead language, we in fact have a final number of sentences (even if we count texts not yet found). But still we must believe that when spoken, it was possible to formulate an infinite number of sentences and therefore we must formulate our grammar in such a way that this is possible. From an NLP point of view ancient Egyptian language is interesting because it contains a great potential for ambiguities (see dictionary) as well as it contains a lot of irregularities and exceptions from those basic rules that it is possible to formulate. Considered that the text corpus is completely static, one might propose using machine learning for trying to cover the grammar which is not very easily formalisable. But machine

¹⁰Those informations are from [Gar27], p. 5.

¹¹[Gar27], p. 1

¹²[Gar27], p. 2.

learning is not at all in focus in this report. On the contrary our focus is to cover some basic rules of the grammar and solving some of the problems involved in this.

1.4 Second Part of the Problem: The Grammar and the Tools

1.4.1 Field of Focus

In this report we want to examine the difficulties in handling some of the widespread ambiguity, one has to deal with when formalising the grammar of the ancient Egyptian language. We regard this language as a very good example of some of the general problems connected to formalising a language in order to recognise or manipulate it in a computer, since the problem of ambiguities is unavoidable when modelling any natural language.

We have entered upon new grounds in two ways: Of course we were all more or less new to the field of NLP and two thirds of the group-members were new to logic programming.

But also we are entering two fields that are not only new to us: Firstly CHR_G (the name of the formalism we have used, explanation in the CHR_G chapter, p. 16) is rather new and still in development and we are the first to use it seriously for NLP purposes (according to its creator, H. Christiansen).

Secondly the application of NLP to ancient Egyptian seems like an area in which not many have tried themselves before. We have not managed to trace any preceding attempts to handle Egyptian grammar in a computer in a way that resembles our project ¹³. Our guess is that this is because it is regarded as impossible to make a functioning grammar that in any way covers all the more or less strange and consequent rules that constitute this grammar. But it has neither been our aim to make a full description of the ancient Egyptian grammar nor to decide whether it is possible or not.

We have our focus on the methods and tools involved in formalising a language in NLP and we have narrowed the range of our focus to syntax, and with the exception of a few morphological rules we are not looking at other aspects like semantics (see dictionary), phonology and the like. Neither are we dealing with the fact that hieroglyphs are pictures and that one aspect of handling them could be picture recognition. We simply use a notation for naming hieroglyphs. For a list of computational hieroglyphic projects see the Egyptological Computing page of [Hie02]. Those are mostly writing

¹³The work closest related to our project, which we managed to find, is the Thesis of Serge Rosmorduc [Ros96].

tools and so-called translation software, which is most often phonographic 'translation' into the nearest letters in a western alphabet or sign-by-sign lookup.

In a short formulation we wanted to make an extendible and general model of ancient Egyptian grammar. It should be able to verify whether a list of 'signs' (i.e. tokens like A1, B15, B20) is a correct sentence according to the grammar formulated in the program.

In the following we will, via a set of 'conceptual requirements', go into detail about what we mean by this.

1.4.2 Conceptual Requirements

In this section we put forth a set of conceptual requirements to the program and the tools used for writing it. These requirements are in no way revolutionary, since they are assumedly set up for almost any program written. But by stressing some and not others, we can use these requirements as a formulation and shaping of our focus. Moreover, in the Discussion chapter, they are used for evaluating and measuring our work and tools.

Extendibility and Readability

Because we want to make an extendable model of the ancient Egyptian grammar, we set up two closely related requirements: Extendibility and Readability.

Regarding the extendability it is nearly impossible to make sensible measurement of how extendable a program is. How does one create a program, which is extendable? Or, are not all programs extendable? As we see it one way of making a program easy to extend is to use general design patterns as well as including a thorough documentation of the program. By general design patterns, we mean using common solutions to common NLP problems, where possible.

Because we do not want to make a model that handles the full ancient Egyptian grammar, we assume that the only users would be Egyptologists who are willing to add the features needed in order to make it fit their specific needs and so to say finish the program. This again means that we suppose that the user of our program is going to modify the existing source code. Though we might assume that linguistically oriented Egyptologists are used to rather formal descriptions of the language, we cannot suppose them to have extensive programming experience. For that reason, we have endeavoured to create a program, where the grammar rules can be read directly from the source code. In this way extendibility and readability can be said to be very closely connected properties and as we see it extendibility presupposes readability, simply because it is difficult to extend (i.e. write to) a program that is difficult to read.

Whereas the extendibility is mostly dependent on the writing of the program (one might say its 'design'), the readability depends both on the formalism and on the program(mer).

Furthermore, if an Egyptologist were to find this program any interesting, he ought firstly to be able to express the model of the grammar in such a way that it is possible to spot which grammatical theories it confines to and which not. This makes it a favourable feature that the formalism used is able to express the grammar theory in a way that is at least comparable and at the most desirable equivalent to traditional formulations of the grammar.

As already mentioned our goal has not been to create a program that is in any way finished so that it could be used 'out-of-the-box'. We have been focusing on the modelling aspect solely and decided from the very beginning that we did not want to make any graphical interface, reading from or writing to files, interaction between different programming languages or whatever necessary to make a program to be used by anyone. We repeat this in order to stress how important it is that the formalism we are using allows to have readable source code. We assume that an Egyptologist with special interest in the language might be quite used to reading very formal descriptions of the language. This also means that we regard the execution speed and the performance when analyzing big text corpses, as being less important.

Robustness

One requirement that is not really relevant for the actual implementation, but easily could be in an extended version, is robustness (see dictionary). Robustness in this context is the ability to handle incomplete texts or sentences. Prolog programs often return a 'No' if the input is not completely recognised. But there are cases where an incomplete input could be relevant and the user still wants an output of partial recognised sentences or texts, such as e. g. if one wanted to get the program to try to guess missing parts or if one wanted to handle actual signs by means of picture recognition.

In our case such a robustness is more a property of the formalism than anything else; in CHR_G we have a highly robust parser.

Though it might not be really relevant for the final outcome, robustness is an important feature when developing the program since it allows testing a grammar that is not yet working. We have used this a lot during our work with the grammar.

Ambiguity

In the writing of our grammar a more useful and interesting feature of both the formalism and the program is that ambiguities are handled in a controllable way. In itself ambiguity is not a problem when using constraint programming, but producing a lot of possible interpretations (both partial and full parse trees) is not desirable since that will result in a lot of 'noise'. By the term 'noise' we mean output that is the result of local ambiguities, but cannot be part of a sentence.

A requirement that we find relevant to set up is one that relies both on the programmer and his tools: that the final program is able to recognise sentences but not produce too much noise. Noise is not only making the output less usable, but it can also be a symptom of inaccurate programming. Some interesting aspects of of this the conception of noise is discussed in the Discussion chapter, see page 42.

1.5 Methodology

Once at a meeting our supervisor characterised our project as putting ancient Egyptian grammar and CHR_G in a bottle, shaking and then seeing what the outcome was. This is a good picture of the project in many ways. Mainly because at the starting point we were new to both CHR_G and ancient Egyptian grammar, hence we had absolutely no idea, if it was possible at all. In the next sections we describe our method, when choosing a formalism to make our grammar, how to choose what part of the ancient Egyptian grammar that should be included in our model and how.

1.5.1 The Formalism

When trying to model the ancient Egyptian grammar in a computer, one could choose from different grammar tools (e.g. Prolog-embedding typed feature structure grammar (PETFSG))¹⁴, but since we didn't know any formalisms, we went for the one presented to us by our supervisor; CHR_G. This has some disadvantages. One could argue that we should have looked for the most appropriate for our needs. Furthermore, we are not able to compare the formalism with any other formalisms. On the other hand, the creator of the formalism were at our disposal as supervisor, making it easy to get information and learn *dirty tricks*. Albeit we do not have any strong arguments for choosing the formalism, it has turned out to be a highly suitable tool for our conceptual requirements (see section 1.4.2, p. 11).

¹⁴We just know it by name (<http://stp.ling.uu.se/~matsd/petfsg/>)

1.5.2 Writing Our Grammar

How should one choose what should be included in the grammar, and what should not? There is to our knowledge no clear-cut method to that challenge. Our approach has been to start out with a very basic sentence: "I buried the old", which consists of a verb, a noun (the subject), and a noun (direct object). When we were able to make the program recognize this sentence, we elaborated the program to make the rules more general, in order to cover more and more advanced input sentences. How far we got making the model as general as possible (the number of iterations), depended on how much time we had on our hands, in combination with how far we got in understanding the ancient Egyptian grammar and got able to use the CHR_G formalism. We have used this method of developing from examples, making them more and more general in order to fit other possible sentences, mainly because we were not ancient Egyptian grammar experts in any way. By using examples from the books concerning hieroglyphs, we felt that we had our backs clear regarding the reliability of grammar (see next section). Also, and especially in the testing, we have tried to refrain from interpolating from English examples, i.e. assuming that just because it happens in English, it happens also in ancient Egyptian.

1.6 Literature

1.6.1 Ancient Egyptian

As our main reference to the ancient Egyptian language we have used "*How to Read Egyptian Hieroglyphs*" by Mark Collier and Bill Manley from 1999 ([CM98]). It presents a very simplified grammar, but helpful as an introduction: "*The focus of this book is on reading actual monuments, rather than struggling through a morass of grammar.*"¹⁵

For further and more advanced questions we have consulted Sir Alan Gardiner's "*Egyptian Grammar - Being an Introduction to the Study of Hieroglyphs*". This is an impressive work and though it was originally published in 1927 this book is still one of the main (if not **the** main) reference on Middle Egyptian. But of course some of the knowledge is outdated and this especially concerns the conception of the verb where new theories have been introduced by among others, J. P. Allen¹⁶.

¹⁵[CM98], p. 144.

¹⁶We have not found it necessary to consult these theories firstly because the part of the grammar we deal with is so simple and secondly because we built much on [CM98], who supports the modern 'verbalist' point of view.

1.6.2 Prolog, CHRg, NLP etc.

As introduction to NLP we have used '*Natural Language Processing for Prolog Programmers*' by Michael A. Covington ([Cov94]) - a useful introduction to the field of NLP. For us, it has been useful as an introduction to the basic notions and analyses, the conventions and the NLP way of thinking. This is a book we would recommend but since it deals mostly with examples from English syntax and teaches only DCG (see dictionary), a great part of the details have been irrelevant to our work.

As introduction to Prolog programming we have used the classical Ivan Bratko '*Prolog, Programming for Artificial Intelligence*' ([Bra90]) (the first half of the book).

As reference on CHRg we have mainly used the "*User's Guide to CHR Grammars (CHRgS)*" at Henning Christiansen's website ([Chr02b]). For a more advanced treatment, see the article '*CHR Grammars*' ([Chr02a]).

1.7 Overview of the Report

This report contains seven chapters: The first chapter is the present one, the introduction. The next is a short introduction to CHRg in order to present our main tool to the reader, in very short terms. Chapter 3 is not really an introduction ancient Egyptian grammar, but rather to some basic characteristics with focus on the things related to our modelling project; especially the use of signs and the parts of the grammar that are covered by our program.

Chapter 4, "Modelling the Language", is to be regarded as the main chapter, in which we start out with an overview of our program and then move on to describing the four sections that we divided it into. In this chapter we also explain each of the problems we have focused on and the solutions we chose. The next Chapter "Experimental Results" is a short chapter concerning mainly the output of the program. After this the Discussion chapter (Chapter 6) discusses problems and solutions, tools and program on a more general level. The report is then ended with a concluding chapter summing up and generalising our points.

Chapter 2

CHR Grammars

This chapter is largely based on the CHR_G website [Chr02b]. More exhaustive information about CHR_G, particularly about features that are out of the focus of our program is to be found there and in [Chr02a].

In this chapter we assume that our reader is to some extent familiar with the Prolog way of 'thinking' and, to a minor extent, with CHR. We introduce CHR_G by shortly defining it and mentioning the main features that fits our needs, and then we move on to the more gentle explanation from examples. For definitions of confusing terms, see the dictionary at the end of the report (see page 53).

CHR_G stands for Constraints Handling Rules (or **CHR**) Grammars and is a logical grammar system based on CHR, analogously to the way Definite Clause Grammars (**DCG**) are defined and implemented on top of Prolog. Thus like DCG, CHR_G are a grammatical interface to CHR.

Contrary to the classical DCG, CHR_G work bottom-up and have some very useful features not found DCG:

- CHR_G provide grammar rules based on propagation, simplification, and simpagation rules of CHR.
- Ambiguity is not a problem in CHR_G since all different interpretations (i.e. all different parses) are allowed to exist at the same time. In CHR terminology: they are "*...evaluated in parallel in the same constraint store*"¹. As mentioned earlier CHR_G is quite robust to incomplete sentences. This is mainly because of its bottom-up approach.

¹Cf. [Chr02b]

2.1 Introduction to CHR_G by example

Here is a very short example of how CHR_G could apply to NLP (the numbers are for reference only and should not be understood as a part of the programs):

```
1 handler test.  
  
2 grammar_symbols np/0, verb/0, sentence/0.  
  
3 np, verb, np ::> sentence.  
4 [peter] ::> np.  
5 [mary] ::> np.  
6 [likes] ::> verb.  
  
7 end_of_CHRG_source.
```

The `handler` (1) and the `end_of_CHRG_source` (7) can be considered as mnemonics that just have to be there in any CHR_G program.

The grammar symbols are defined in the beginning of the CHR_G program with a tag `grammar_symbols` (2) and specifies what entities the parser should look for.

As any other basically formed CHR_G rule, the first rule (3) consists of a head (`np`, `verb`, `np`), an arrow (`::>`) and a body (`sentence`). This rule states that, according to the English grammar, a sequence made up of a noun phrase, a verb and a noun phrase, could be recognized as a valid sentence.

The next three rules (4-6) aim at propagating a token, or a list of tokens, to a grammar symbol.

All the grammar symbols behave like Prolog predicates.

Input

The program could be run by calling CHR_G with the `parse` command, which would be like this:

```
parse([peter, likes, mary]).
```

Output

The output would in its essence look as follows:

```

<0> peter <1> likes <2> mary <3>
all(0,3)
np(0,1)
verb(1,2)
np(2,3)
sentence(0,3)

```

Note that in a real version of this simple program we would together with the above have a list of the less interesting `token(0,1, peter)`, `token(1,2, likes)` and `token(2,3, mary)`. The first two lines of the output are not very interesting as results, but it should be mentioned that the numbers between <'s and >'s are boundaries marking the positions of items found. These numbers are also the first two parameters of the predicates listed (line 3-6).

Rules and Lexical Information

In the above example, all the arities are zero, but it is possible to attach attributes to the grammar symbols, thus enabling in our example to define the gender or the number of a noun phrase, or the tense of a verb.

This could be done as follows:

```

[peter] ::> np(singular, masculine).
[mary]  ::> np(singular, feminine).
[likes] ::> verb(singular, present).

```

As in any Prolog predicate, attributes starting with a lower-case letter are considered as constants, whereas those starting with an upper-case letter are considered as undefined variables. Moreover, the `_` notation can be used to match any value of an attribute, as in the following example:

```

np(_, _), verb(Tense), np(_, _) ::> sentence(Tense).

```

2.2 Other Interesting Features of CHR_G

In this section, we describe the interesting features of CHR_G we used. An exhaustive description of CHR_G syntax and all its possibilities is provided at the CHR_G website [Chr02b].

2.2.1 Context sensitive rules

One of the most important features of CHR_G is the easy way to implement context sensitive rules, by using the operators `-\` and `/-`. The syntax of those is: `<right-context> -\ <head> /- <right-context>`.

As an example, we can use the right context of a noun phrase to try to identify it as a subject, by adding this rule to our previous program:

```
np /- verb ::> subject.
```

This context sensitive rule states that if a verb is placed on the right-hand side of a noun phrase, the latter can be considered as a subject.

Operators `-\ and /- can be used in the same rule, if there is any need of checking both left and right contexts.`

2.2.2 Simplification

The operator `::>` propagates one or several grammar symbols into another one. This means that a new constraint is added to the constraint store. Instead of adding a new constraint, it could be useful, even necessary, to replace a constraint by another. This is done by using simplification instead of propagation. The operator for simplification is `<:>`.

For example, we can use simplification instead of propagation in our little program:

```
np, verb, np <:> sentence.
```

Then, the three constraints `np`, `verb` and `np` will be removed from the constraints store, and `sentence` will remain in it.

2.2.3 Guards

CHRG allow the use of guards. A guard must be placed before the body of the rule, between the arrow and the guard sign `|`.

A given rule:

```
np(Name1), [likes], np(Name2) ::> Name1 == Name2 |  
narcissist.
```

applies only if the condition `Name1 == Name2` is satisfied, i.e. the value of `Name1` equals the value of `Name2`.

Of course, a guard can consist of more than one condition. In that case, they have to be separated by commas.

2.2.4 Optional grammar symbols in head of rules

It may occur that the difference between two rules resides only in one part of the head, that is present in one and not in the other. In order to avoid writing the almost identical rule twice, hence improving the readability of

the program, the `optional()` operator can be used.

Then these two rules

```
np, vp, np, pp ::> sentence.  
np, vp, pp ::> sentence.
```

can be gathered into a single one using the `optional()` operator:

```
np, vp, optional(np), pp::> sentence.
```

Actually, the CHR_G compiler will expand the latter rule into the two former ones. Then the `optional()` operator could be considered as *syntactic sugar*, making a CHR_G program quite more readable.

The `optional()` operator can be used only in the core of the head, i.e. not in the context parts.

Chapter 3

Ancient Egyptian grammar and the interesting modelling problems

The purpose of this section is to introduce the grammar of ancient Egyptian (comparing to English where possible). It is not meant to be a reference; the focus is on the problems we came across and handled, as described in the next chapter. For a complete introduction of ancient Egyptian grammar, we refer to 'How to read Egyptian Hieroglyphs' [CM98] and to 'Egyptian Grammar' [Gar27]. All the explanations here are based on these books.

3.1 The signs of ancient Egyptian

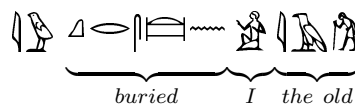


Figure 3.1: The sentence "I buried the old" written in ancient Egyptian.

The hieroglyphs as shown in figure 3.3 are the atoms of Ancient Egyptian, like letters are the atoms of English. However, there are *only* 26 letters, whereas egyptologists have counted about 6900 hieroglyphs¹. Even so, the amount of signs is not the biggest difference. Letters in English only fall into two categories: consonants and vowels. Ancient Egyptian is more complicated.

¹According to *Hieroglyphica* [GHvdP93].

As shown in figure 3.2, we can split hieroglyphs into two main categories: picture signs (called **ideograms** or semograms) and sound signs (called **phonograms**).

An ideogram can either be used as a **logogram** or a **determinative**. A logogram has pictorial meaning, i.e. the reader should interpret the sign as the object, act or relation that it depicts. A determinative placed at the end of a word is never pronounced. It only helps the reader to get a general idea about the meaning of a word. For example, the sign \sphericalangle represents human legs, indicating the concept of *motion*.

As for the phonograms, they are divided into 3 categories, according to the number of consonants they consist of (1, 2 or 3).

The tricky part is that the same sign can fall into both categories (ideogram and phonogram) - not at the same time, of course, but in different contexts.

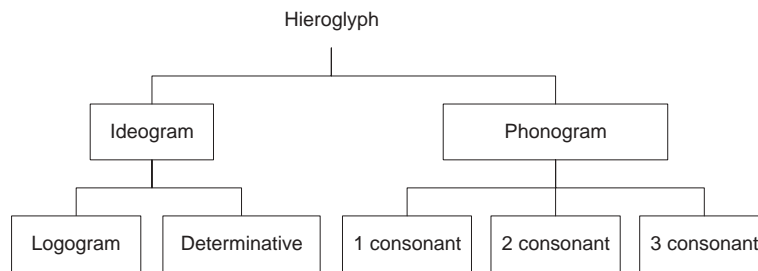


Figure 3.2: The different types of hieroglyphs

Like in Hebrew, the sounds written in the ancient Egyptian language are only the consonants. The spelling of a word may consist of a number of ideograms and a number of phonograms, often ended by a determinative. Furthermore, the 'rebus principle' is used. The meaning of a word can be used as sound, e.g. a bee and a leaf could be 'belief' (not an example from Egyptian). Here an ideogram is being used as a phonogram, and is called *phonetic complementing*.




3.2 Direction of writing

Hieroglyphs were used in a more decorative manner than letters are in English. They often formed a fundamental part of the aesthetic value of a monument. Although we present hieroglyphs in a left-to-right in this report, one should know that hieroglyphs may be also arranged right to left, or in columns, in order to make the writings look better. If in columns,

hieroglyphs are always written top-down. To differentiate the two horizontal arrangements, the reader should have a look to the direction signs are facing. They normally look to the beginning of the line.

3.3 Arrangements of signs

Hieroglyphs are not arranged one after the other, like letters are in English, but in balanced groups in order to fit the available space, in a more pleasing way. The rule for reading groups is simple: read the top sign(s) before the bottom one(s).

Let's have a look at the name 'Senbi': . According to  facing left, we know that the word should be read in a left-to-right order. When combined the horizontal precedes the vertical reading. So, if we lay down the word, it would be spelled like that: 

3.4 No clear separation of words

In English each word is delimited by a space to make the text legible. The sentence: *This is not very legible!* makes that point clear. But that is not the only reason for separating the words. It might be the case that the delimiters could be placed at different positions, hence causing structural ambiguity.

In Ancient Egyptian there is no clear definition to when one word ends and another begins. All words are written next to each other, without any spacing. However, in some situations, we can get some hints on where a word stops or not.

First, we have seen that an ideogram can be interpreted as a determinative. In that case, it will end the word it is attached to. But as it could also be a logogram, and then not necessarily end a word, we only have an indication of a possible word ending.

Second, as mentioned in 3.3, Egyptians often grouped signs. An useful information is that these groups can belong to only one word. Therefore, a word ending can be situated only before or after a group of signs.

But still, there are no general and always applicable rules to determine where a word begins and ends.

3.5 No punctuation marks

In English and other languages written nowadays, punctuation marks are used to separate words into sentences, clauses, conjunctions and phrases in order to clarify meaning. Who likes whom in the following sentence?

”Peter likes Magda and Mary and Martha and Meryl like Paul.”

Different interpretations are valid, or at least two:

”Peter likes Magda, and Mary and Martha and Meryl like Paul.”

”Peter likes Magda and Mary, and Martha and Meryl like Paul.”

This is an example of the consequence of removing the punctuation marks from an English sentence, but that does not automatically make it a problem in Ancient Egyptian, since it might be the case that such situations are avoided simply by rephrasing.

3.6 Different ways of spelling the same word

Words in hieroglyphic writing do not have one single correct spelling but are rather 'elastic', in order to fit a lack or a surplus of space. As a matter of fact, a word can be contracted or expanded through the inclusion or omission of sound complements. A sound complement is a 1-consonant sign that accompanies a 2-(or 3-)consonant sign.

For example, let's consider the word $\begin{matrix} \square \\ \ominus \end{matrix} \Lambda$, meaning *to go out*. It consists of 2 phonograms and a determinative :

- \square is a 2-consonant, representing a plan of a house, is pronounced *pr*
- \ominus is a 1-consonant, representing a mouth, is pronounced *r*
- Λ is a determinative, as mentioned before, indicating motion.

However, the word is pronounced *pr*, and not *prrr*, due to the phenomenon called 'phonetic complementing'. Indeed \ominus is only written to jog out the memory of the reader about the *r* of \square . Therefore, both spellings $\square \Lambda$ and $\begin{matrix} \square \\ \ominus \end{matrix} \Lambda$ are correct.

Moreover, the order of the signs can change, in order to get a more satisfactory visual effect.

3.7 Omission of a subject in a sentence

In the Ancient Egyptian language, the subject of a verb can be omitted if the sequence $\begin{matrix} \text{𓂏} \\ \text{𓂏} \end{matrix}$ is present just before the verb. Actually, these two symbols invoke a sense of involvement in the assessment or presentation of what is written.

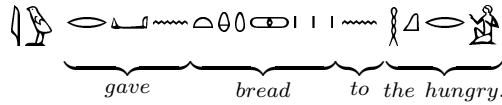




Figure 3.3: The sentence "(I) gave bread to the hungry" written in ancient Egyptian.

In self-presentation inscriptions,  gives a sense of looking back at one's life. In other contexts, the perfect ("someone has done something") also suits, particularly in recorded speech.

It is important to note that the presence of  does NOT imply that the subject is omitted. It is then just a way to emphasize the subject.

The ambiguity about the possible presence of a subject in Egyptian sentences was one of the most important issue when modelling ancient Egyptian grammar (see section 4.9 page 35).

3.8 Structure of ancient Egyptian sentences

Like any language, ancient Egyptian follows some rules concerning the usual structure of the sentences. If we only consider grammatical elements such as subjects, verbs and objects, an ancient Egyptian sentence would be structured as follows ²:

1. The verb, with its tense, number and gender inflexions.
2. The subject, if not omitted.
3. The object(s), direct, indirect, or both, according to the transitivity of the verb.

²In so-called verbal sentences (contrasted to non-verbal sentences) which are the ones concerned in our grammar, "...the normal word order is: 1. verb, 2. subject, 3. object, 4. adverb or adverbial phrase..." [Gar27], p. 34

Chapter 4

Modelling the Language

4.1 Introduction

This chapter has two main purposes.

The first is to provide a documentation of our program, or put differently: to describe our representation of ancient Egyptian grammar in detail. Here we deal with the implementation, basing on the structure of the program.

The second purpose - which is overlapping with the first - is to present the main problems we have chosen to express in our implementation.

Our goal has not been to solve all the problems that we have come across, but the ones we found most interesting and most essential in order to design a working model. Thus this chapter is treating the main problem - the design of an extendable model of some basic aspects of Ancient Egyptian grammar - and describing the solution we have come up with.

We start out by giving an overview of the grammar by listing the different layers our program consists of, together with an explanation of our model of the syntactic structure of Ancient Egyptian sentences. Each of the layers is then explained in detail.

The three following sections deal with some specific problems we came across. Each one is explained, as well as the solutions we propose.

4.2 Overview of the Program

We have divided the program into four layers, although they are not that strictly separated in practice:

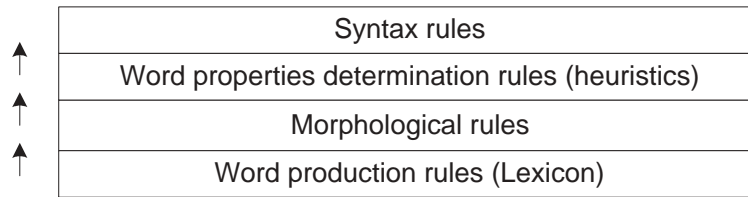


Figure 4.1: The layers of the program

- Syntax rules, which defines the permitted structural position of words and phrases.
- Constituent properties determination rules, which defines how our knowledge of ancient Egyptian grammar is used to classify properties of constituents (i.e. words or whole phrases).
- Morphological rules, which classifies different representations of the same word.
- Lexicon rules, which defines the word production rules.

As the parser used in CHR_G works bottom-up, we'll now present these different layers starting from the lexicon, until the syntax rules.

4.3 The Lexicon

As in every grammar we have in our program a list of terminal symbols, i.e. a lexicon. Each entry of the lexicon is a CHR_G rule, and defines both the spelling and the class of a word.

The head of such a rule consists of the spelling of a given word in its stem form (that is its simplest form, without any tense, gender or number inflection), whereas the body is a grammar symbol representing the class the word belongs to.

4.3.1 Spelling the words

According to CHR_G syntax, the words are spelled as Prolog lists of hieroglyphs, under their computed representation. This representation uses for each hieroglyph an unique combination of a letter and a number (for more information, see page 6).

As an example, the verb *to bury*, spelled $\overline{\text{Ⲁ}} \overline{\text{ⲓ}} \overline{\text{ⲛ}} \overline{\text{ⲛ}}$ in Ancient Egyptian, would be represented as follows:

[c11, a38, e28, f19]

As seen in section 3.6, a given word can be spelled in different ways. However, in order not to spend too much time on building up an complete exhaustive lexicon, we chose to define only one (sometimes two) entry for each word. The chosen spellings are the one defined in *How to read Egyptian hieroglyphs* [CM98] page 151, *Egyptian-English vocabulary*.

Moreover, we do not include information about the grouping arrangements (see 3.3) in the entries of the lexicon, since arrangements are only a matter of visual effect, and have no consequence on the spelling of the words.

4.3.2 Word classes

Important information is added on the lexical level. As the most basic information, the words are classified into word classes, by propagation rules.

This is the entry of the lexicon dealing with the verb 'to bury':

[c11, a38, e28, f19] ::> v(sing, infinitive, 1).

This rule classifies the word spelled [c11, a38, e28, f19] as a verb, using the grammar symbol **v**. As words are spelled in their stem form, the entry of the lexicon concerning 'to bury' also specifies that this verb is in singular (1st attribute : **sing**) and not conjugated (2nd attribute : **infinitive**). Moreover, the lexicon is the layer where we are the most likely to attach transitivity to verbs. Indeed, it is the only stage where we know the meaning of a verb, and subsequently its transitivity. The transitivity of a verb can have three values :

- 0** : The **intransitive** verbs do not expect any object (e.g. "*I live*")
- 1** : The **transitive** verbs expect only one kind of objects, either direct or indirect (e.g. "*I buried the old*" or "*I spoke to the pharaoh*")
- 2** : The **di-(or bi-)transitive** verbs expect both a direct and indirect object (e.g. "*I gave bread to the hungry*")

In our case, the verb 'to bury' is transitive and only expects a direct object (3rd attribute : 1).

The other word classes are represented as follows:

- **n** is the grammar symbol for nouns
- **pron** for pronouns

- p for prepositions

The two former grammar symbols have arity 1. The only attribute is the number. Most of the entries concerning nouns have this attribute defined as `sing`, due to the fact that most of the Ancient Egyptian words have a stem form in singular. A few exceptions exist, like the word *bread* which always appears in plural. In that case, the attribute is `plur`.

The grammar symbol for prepositions has no attribute.

4.4 The Morphological Rules

As in many languages, words in Ancient Egyptian can be inflected. This is applicable for verbs, which agree in number and gender with the subject, and also have tense inflections. Moreover, nouns can be inflected in number. These inflections are often represented by one or several signs following the stem form of the word they are attached to. For example, ' ' ' (D3) after a noun would indicate a plural inflection, and ~~~~ (C15), the past tense if it follows a verb. Then, the morphological rules will attach this information to the related word.

4.4.1 Recognition of inflections

The first step is to recognize these inflections. In the two examples mentioned, the inflection consists of a single sign. It would generate a lot of noise if we used a rule like:

```
[c15] ::> verb_inflection(past).
```

because any occurrence of ~~~~ would be interpreted as a past inflection, all the more is that this sign is quite often used in Ancient Egyptian language.

The use of context-sensitive rules can partially solve the problem:

```
v(Num,infinitive,Trans) -\ [c15] ::> verb_inflection(past).
```

Then, an occurrence of ~~~~ would be interpreted as a past inflection only if it directly follows a verb not yet inflected in tense (2nd attribute : `inifinitive`), regardless its number or its transitivity.

In the same way, ' ' ' will be recognised as a plural inflection only if it follows a noun in singular :

```
n(sing) -\ [d3] ::> noun_inflection(plur).
```

Although these rules handle terminal symbols, such as [c15] or [d3], we do not include them in the lexicon. Indeed, they also use context-sensitivity including non-terminal symbols. This is the reason why one could argue that the morphological rules layer overlaps a bit the lexicon.

4.4.2 Morphology

Once an inflection has been recognised, this information has to be attached to the word. This is done by the morphological rules, which handle only non-terminal grammar symbols.

For example, the tense of a verb is defined as follows :

```
v(Num, infinitive, Trans), verb_inflection(Tense)
::> v(Num, Tense, Trans).
```

This rule propagates a verb (in its stem form) and its inflection into an inflected verb.

Nouns are inflected in the same way :

```
n(sing), noun_inflection(Num) ::> n(Num).
```

It may look useless to define the noun inflection as a single-attribute grammar symbol, since the only value of this attribute is `plur`. Actually, it is the only value *for the moment*. But as we keep in mind our program could be reused and extended, the attribute will be likely to have values like `feminine` or `masculine`, in order to implement the gender of nouns (implying a 2nd attribute for the `n` grammar symbol).

4.5 Constituent Properties Determination Rules

After being tested by the morphological rules, nouns and verbs have been inflected. And they now have to become **phrases**, i.e. noun phrases and verb phrases. As to cover both words and phrases we use the term 'constituent' for both, since a sentence *consists* of word and/or phrases (dependent on the point of view). Making phrases out of words is done through these rules:

```
v(Num, Tense, Trans) ::> vp(Num, Tense, Trans).
n(Num) ::> np(Num, Role, Type).
pron(Num) ::> np(Num, Role, Type).
```

For the verb, nothing really changes. It keeps its three attributes, namely `Num` for the number, `Tense` for the tense and `Trans` for the transitivity. One

could argue that our definition of a verb phrase differs to the traditional linguistic one, which also includes the objects in addition to the verb as part of the verb phrase. However, this definition is applicable to modern languages such as English, in which the objects usually follows the verb. As seen in 3.8, in ancient Egyptian, the subject should stand between the verb and the objects, thus making impossible to gather the latter into a traditional verb phrase. We therefore decided to define a verb phrase as a verb, as well as its inflexions.

Things are slightly different for the noun. Noun phrases have three attributes: the number inherited from the noun, the role (subject or object) and the type (direct or indirect) in case of an object. Note that `Role` and `Type` are undefined when the rule applies.

Transformation from a pronoun to a noun phrase is done identically.

The next step is to determine the properties of such phrases. All the necessary information about verb phrases are known from the lexicon and the morphological rules. But most of the properties of the noun phrases can only be deduced from the context of the sentence.

Let us start with the subject. In ancient Egyptian, it follows the verb. Then, the rule

```
vp(Num, _, _) -\ np(Num, Role, Type)
  ::> var(Role), var(Type) | np(Num, subj, subj_type).
```

will identify an undefined `np` as a possible subject if it follows the verb, and if they agree in number. The `var()` predicates in the guard are built-in Prolog predicates. They succeed if the argument is an undefined Prolog variable. By using this guard, we want to avoid this rule to apply to any `np`. If the guard wasn't there, the program would loop endless, since this rule would create such an `np` that it would apply again, and so on...

Usually, after a subject, we ought to find the direct object, if of course the verb is (di)transitive:

```
vp(Num1, _, Trans), np(Num1, subj, subj_type)
  -\ np(Num2, Role, Type)
  ::> var(Role), var(Type), Trans \== 0
  | np(Num2, obj, dir).
```

And as the subject can be omitted¹, this look-alike rule could be useful:

```
vp(Num1, _, Trans)
  -\ np(Num2, Role, Type)
  ::> var(Role), var(Type), Trans \== 0
  | np(Num2, obj, dir).
```

¹see 4.9 *The Implied Subject*, page 35.

In the determination of the subject, the guards make the rules not to apply infinitely. Moreover, the last condition `Trans \== 0` prevents these rules from applying if the left context is an intransitive verb.

Last but not least, we have to determine whether a noun phrase is (or at least can be) an indirect object. This can be easily stated we assume an indirect object is always preceded a preposition:

```
p -\ n(Num) ::> np(Num, obj, ind).
```

Note that we proceed here from a noun, and not a noun phrase. This has been done because the only knowledge needed in order to recognise an indirect object, is that a preposition is present.

4.6 The Syntax Rules

This is the last stage before recognising a sentence. The couple of rules that handle valid sentence structures are explained below, in *4.9 The Implied Subject*, page 35. However, these rules manipulate two new grammar symbols that will be detailed here.

The first grammar symbol is `pp`. It stands for *prepositional phrase*. According to English linguistics, a prepositional phrase consists of an indirect object with its preposition. Subsequently, the rule to generate a `pp` is rather simple:

```
p, np(Num, obj, ind) ::> pp.
```

The other grammar symbol has no equivalent in NLP. Its name, `nppp`, comes from the rule:

```
np(Num, obj, dir), pp ::> nppp.
```

An `nppp` can be considered as a combination of a direct object and a prepositional phrase.

The full use of both `pp` and `nppp` will be detailed in the next section.

4.7 Conjunction

One of the aspects that makes ancient Egyptian hard to read, understand and model is the fact that conjunctions are not explicitly written. In other words, the word *"and"* doesn't exist in ancient Egyptian. Therefore, conjoined words or phrases are just written one after the other.

The way we chose to define conjunctions for noun phrases and prepositional phrases² was to state that two similar phrases standing next to each other are conjoined. It would be then grammatically correct to consider them as a single phrase. This statement is implemented by the following rules:

```
pp, pp ::> pp.
nppp, nppp ::> nppp.
np(Num1, Role, Type), np(Num2, Role, Type)
  ::> np(Num3, Role, Type).
```

The two first rules, handling `pp` and `nppp` conjunctions, do not raise any difficulties. The third rule enables subjects (`np(Num, subj, subj_type)`) and direct objects (`np(Num, obj, dir)`) conjunctions.

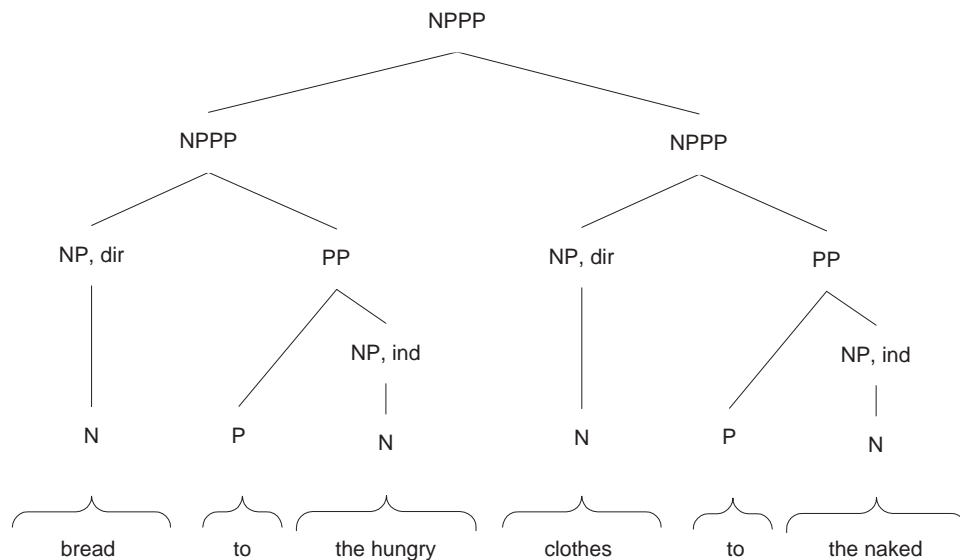


Figure 4.2: Conjunction tree made out of the sentence "I gave bread to the hungry and clothes to the naked"

The figure 4.2 illustrates the way we implemented conjunctions of `nppp`.

²We never came across conjunctions of other phrases, such as verb phrases.

4.8 The Ambiguity caused by the C15 sign

In ancient Egyptian we find three main uses of the sign $\overline{\text{C15}}$:

Firstly it can be used as a word by itself. Here it is a preposition that is translated as 'to' (e.g. in 'I gave bread **to** the hungry').

Secondly $\overline{\text{C15}}$ has what one can interpret as two morphological functions. On the one hand, it can be used in a way that is very similar to the English past tense suffix for verbs: '-ed' as in 'walked'. When being the last sign in a verb, it marks that the verb is past tense. On the other hand, it can be used as a suffix for plural nouns (although the $\overline{\text{C15}}$ sign is more often used). It behaves then like the English '-s' in 'cats'.

There would be absolutely no point in having three rules stating that $\overline{\text{C15}}$ can be interpreted as any of the three possibilities, since it would create too much noise. To reduce considerably this noise, we have to use context-sensitive rules.

$\overline{\text{C15}}$ can be interpreted as a tense inflection if its left context is a verb in the infinitive form :

```
v(Num,infinitive,Trans) -\ [c15] ::> verb_inflection(past).
```

or as a noun inflection (thus indicating plural) if it follows a noun in singular :

```
n(sing) -\ [c15] ::> noun_inflection(plur).
```

We can also interpret $\overline{\text{C15}}$ as a preposition if it precedes a noun :

```
[c15] /- n(Num) ::> p.
```


In most cases, because of the context sensitivity, the parser will come up with only one interpretation. But at least in two situations, $\overline{\text{C15}}$ still remains ambiguous, namely if it is between :

- a verb and a noun; then it could be interpreted both as a tense inflection and a preposition.
- two nouns, if the former is in singular; in that case, it could be a plural inflection for the former noun, and a preposition for the latter one.


This remaining ambiguity is handled by the syntax rules (see page 32), which state what kind of objects (direct and/or indirect, or none) is expected depending on the transitivity of the verb. Therefore, although two interpretations appear in the constraint store, only one will lead to a valid sentence structure.

One should note that we managed to make up some examples which could lead to two different valid sentence structures. However, while making them, we assumed some possibilities in Ancient Egyptian language without having any confirmation about their validity. Moreover, we never came across such examples in the books we used as Egyptology reference ([Gar27] and [CM98]).

4.9 The Implied Subject

As explained previously (see 3.7 page 24), the subject of a sentence can be omitted if  precede the verb.

There is no way (by for example inflections) to distinguish the syntactic role of a noun phrase standing on its own. The noun phrase immediately following a verb can be its subject, or the direct object if the subject is omitted.

Moreover, even if indirect objects are preceded by a preposition, it is not obvious to recognize them, since the symbol of some prepositions can also be interpreted as an inflexion for the previous noun (as an example, see the ambiguity caused by , page 34).

Finally, it is important to remember that the conjunction ("*and*") is never explicitly written in Ancient Egyptian. Then the sequence **Verb NP1 NP2** can be interpreted in different ways, such as :

- NP1 as the subject and NP2 as the direct object
- NP1 **and** NP2 as the subject
- NP1 **and** NP2 as the direct object, and the subject would have been omitted

This brings a lot of ambiguity when trying to determine the syntactic role of a noun phrase standing right next to a verb.

The first step in solving this problem is to define the transitivity (i.e. number of objects) of the verbs. This is done as soon as a verb is recognised, in the lexicon. The transitivity of a verb can have three values : **0**, **1** or **2**.

Intransitive verbs does not raise any problems. If any noun phrase follows the verb, then it is its subject.

Transitive verbs is also not a problem if the verb expects an indirect object. Indeed, indirect objects are prefixed by prepositions, thus making them recognizable if they directly follow the verb. If they don't, i.e. if the subject is not omitted, some prepositions may bring some ambiguity (see page 34). However, any noun phrase between the verb and the first indirect object would be the subject of the verb.

But if the verb expects a direct object and is followed by more than one noun phrase, there is no clue to guess if the subject has been omitted or not.

Ditransitive verbs raises the same problem.

The absence of $\langle \rangle$ before the verb may be helpful. Indeed, it would imply that the subject has NOT been omitted, and then we'll be sure that at least the first noun phrase that follows the verb is its subject.

For that purpose, we defined an entry in the lexicon for the $\langle \rangle$, as well as the new grammar symbol `leafchick`³:

```
[c20, b1] ::> leafchick.
```

This enables us to define different rules that match a valid sentence structure. As for sentences based on a ditransitive verb, the first possible structure is :

```
vp(Num1, Tense, 2), np(Num1, subj, subj_type), nppp
::> s(Tense).
```

But if the $\langle \rangle$ is present before the verb, the subject can be omitted, thus providing a perfect example of the use of the `optional()` operator :

```
leafchick, vp(Num1, Tense, 2), optional(np(Num1, subj,
subj_type)), nppp ::> s(Tense).
```

The rules dealing with transitive verbs look almost identical, the only differences being the value of the 3rd attribute of the `vp` grammar symbol

³Since the common English names for these signs are 'leaf' and 'chick' :)

(transitivity 1 instead of 2) and the grammar symbol `nppp` changed to `np(Num2, obj, dir)` or `pp`, depending on the kind of object the verb expects (namely direct or indirect).

The following figure illustrates the way we structure a sentence including an instance of an `nppp`:

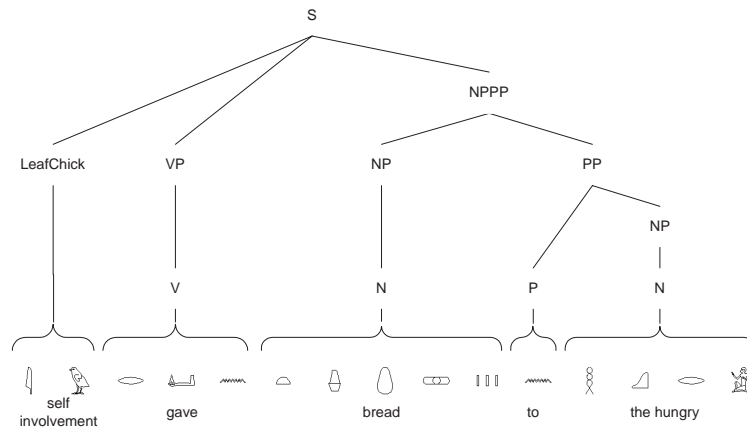




Figure 4.3: The different types of hieroglyphs

One could then think that the program will come up with two valid sentences in the case of both the  and the subject are present. That's partly true, since two grammar symbols `s` will be added in the constraint store. But the fact is that we consider that our program succeeds if it parses a text, rather than a single sentence. In our case, a text consists of at least one sentence, surrounded by the `begin` and `end` grammar symbols. These two symbols come from simplifications of the two tokens `begin_parse` and `end_parse` that have to be added by the user at the beginning and the end of the Prolog list of hieroglyphs he wants to be parsed. According to that, a text would be defined as follows :

```
begin, s(_), end <:> text.
```

From the two valid sentences we may have recognised, only one includes  and then makes this rule applicable.

In order to handle texts that consist of more than one sentence, we just have to define a new grammar symbol: `several_s`. It basically consists of two sentences:

```
s(_), s(_) ::> several_s.
```

and grows more and more by wrapping the next sentences one by one :

```
several_s, s(_) ::> several_s.
```

in order to finally fit between the **begin** and **end** grammar symbols :

```
begin, several_s, end <:> text.
```

Chapter 5

Experimental Results

5.1 Sentence Recognition

In our testing of the program, we were able to recognize all the sentences, upon which the program was designed. Additionally, we made it recognize a few homemade sentences, when testing conjunction of noun phrases and conjunction of sentences. In the example below, one can see the output of the program, when parsing the sentence *"I gave bread to the hungry"* where the subject is omitted. The rest of the output from the testing is attached in the appendix.


```

leafchick(1,3),
pp(5,10),
p(5,6),
pp(5,10),
pp(5,11),
pp(11,16),
pp(5,16),
pp(5,16),
text(0,17),
token(1,2,c20),
token(3,4,a38),
token(5,6,c15),
token(7,8,d25),
token(9,10,f9),
n(6,11,plur),
token(12,13,e35),
token(14,15,a38),
n(12,16,sing),
vp(3,5,sing,infinitive,2),
vp(3,6,sing,past,2),
np(6,10,sing,subj,subj_type),
np(6,10,sing,obj,ind),
np(6,11,plur,obj,dir),
np(6,11,plur,obj,ind),
np(12,16,sing,obj,ind),
p(5,6),
pp(5,11),
pp(5,11),
pp(5,11),
p(11,12),
pp(5,16),
pp(5,16),
nppp(6,16),
all(0,17),
token(2,3,b1),
token(4,5,a41),
token(6,7,d24),
token(8,9,b28),
token(10,11,d3),
token(11,12,c15),
token(13,14,c11),
token(15,16,a1),
pron(15,16,sing),
v(3,6,sing,past,2),
np(6,10,sing,_A,_B),
np(6,10,sing,obj,dir),
np(6,11,plur,_C,_D),
np(6,11,plur,obj,ind),
np(12,16,sing,_E,_F),
np(15,16,sing,_G,_H)

```

Figure 5.1 - Program output when parsing sentence:
(I) gave bread to the hungry.

As one can see, the program recognizes the complete sentence as `text(0,17)`.

Apart from recognizing sentences, the different layers of the program could be tested separately. That is not completely true, since there is only one way of running the parser, (`parse()`); but one could focus on one the rules fired from a single layer. The word production rules and morphological rules, however, are not very interesting to test, because there is not much which can go wrong. The word properties classification rules and syntax rules, however, are much more interesting.

5.2 Word Properties Classification Rules

Testing this layer by it self, would be a way to determine how well our rules of thumb perform. It could be done by testing an extensive amount of valid ancient Egyptian sentences, and carefully examining which word properties classification rules are fired. This might reveal if some of the rules are problematic in the sense that they might not be as wide-ranging as we thought.

5.3 Syntax Rules

It would be interesting to see if other sentences with same structure are recognized by the program. It would only be possible, of course, if entries of the new words were added to the lexicon. This might reveal if our syntax and sentence rules not are adequately general. Due to lack of time, we did not do any testing focussing on either of the two layers.

Chapter 6

Discussion

In this chapter we elaborate on the result we have reached and discuss the interesting aspects of the actual implementation as well as possible future extensions. This also implies evaluation and analysis of the most important characteristics, which are for the greatest part formulated through the conceptual requirements.

6.1 NPPP vs. NP(di)

Actually we came up with two versions of our program. The main differences between these two is the way the conjunction of noun phrases is implemented, and how indirect objects (and subsequently the combination of direct and indirect objects) are defined and managed.

As seen in the [ref : MODELLING CHAPTER], we used the `nppp` predicate so far. The NP(di) version takes its name from this `np(_, obj, di)` exotic entity it introduces. This idea of gathering direct and indirect objects into a single noun phrase came as a solution of the conjunction problem. The point was to make the recursive rule

```
np(_, Role, Type), np(_, Role, Type) ::> np(_,Role, Type).
```

applicable to as many types of noun phrase pairs as possible.

- Two subjects: *"My son (subj) and I (subj) gave bread to the hungry."*
- Two direct objects: *"I gave bread (dir obj) and wine (dir obj) to the hungry."*
- Two indirect objects: *"I gave bread to my wife (indir obj) and to my son (indir obj)."*

- Two pairs of direct and indirect objects:

”I gave bread to hungry and clothes to the naked .”
direct and indirect objects pair *direct and indirect objects pair*

The definition of an indirect object is slightly different in the NP(di) version. In fact, besides the noun, it also includes the preposition. The np(, obj, ind) actually is what linguists traditionally considers a prepositional phrase, represented by the pp grammar symbol in the NPPP version. Thanks to this tricky definition, the previous rule can be used to implement conjunction of indirect objects.

As for the construction of an np(, obj, di), it is stated by the following rule:

np(, obj, dir), np(, obj, ind) ::> np(, obj, di).

Like for the indirect object, the conjunction rule applies to a pair of direct and indirect objects.

The structure of a sentence including a conjunction of direct-indirect objects is therefore slightly modified:
 (see figure 6.1 below and figure 4.3 page 37).

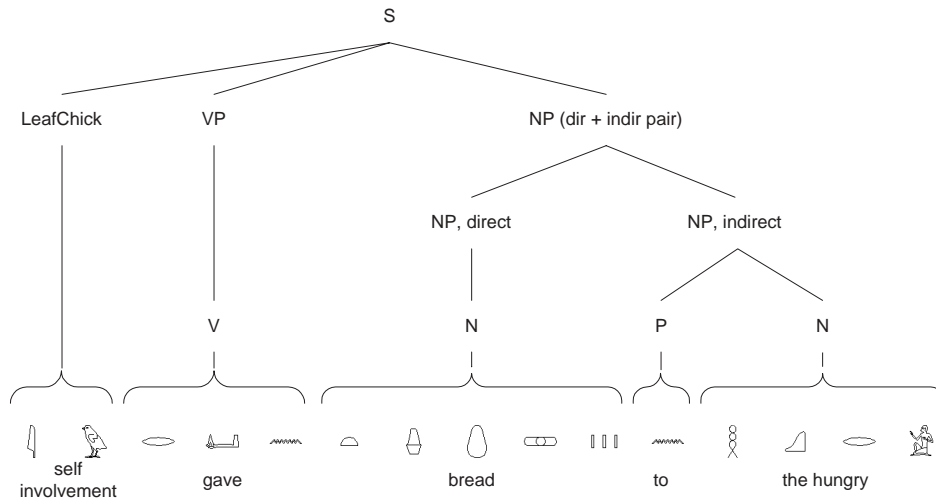


Figure 6.1: Syntax tree with a pair of direct and indirect objects.

6.1.1 Pros And Cons Of Each Version

Readability

The NP(di) version uses an alternate definition of the indirect object, which might be confusing (and maybe even repelling) at first sight for a linguist. Since we wanted to stick to the notation commonly used in the field of NLP - in order to avoid the confusion - the names of the grammar symbols as close to the *standard* as possible. From this point of view, the NPPP version is preferable, since it uses the notion of prepositional phrase, and the NPPP predicate itself intuitively could be interpreted as an NP and a PP.

Noise

As we tested the two versions, it turned out that the NPPP version produced much more noise than the NP(di) one. This may be due to the fact that the latter use an unique grammar symbol (with specific attributes) to describe any instance of a noun phrase, thus limiting the number of constraints in the store, and in the same way the number of possible wrong interpretations.

Which one to choose?

As explained above, the NP(di) version was written as an efficient solution to the problem of implementing conjunctions of noun phrases, regardless of how incorrect the grammar symbols would be, according to a linguist.

But as one of our most important conceptual requirements is the readability of our program, we found it quite obvious to make the NPPP version the *official* one, and just present the NP(di) version as a cunning way to deal with the conjunction problem.

And even if one tried to parse a very long text, the main requirement would not be likely to have the result of the parsing as quickly as possible, but to have the most readable one, implementing the notation he traditionally uses (in case of a linguist).

6.2 Conceptual Requirements Of The Program

6.2.1 Next Step Implementations

If our goal was to make a usable model, we would have needed much more time and much more knowledge of ancient Egyptian. In its present state it recognises a rather limited variety of sentences. Still a huge number of possible sentences are not covered (as ought to be obvious). Covering them

would probably be near to an infinite project (if at all possible, which is not very likely). But by a number of considerably small steps, we could make the model more realistic. As the most important and interesting, which would raise the number of possible input sentences, we have considered implementing:

- Adjectives.
- Other prepositions than 'to'.
- Kings names. (Are used a lot in the examples we have seen.)
- Gender of verbs and adjectives and their agreement.
- Gender of nouns. (Would be rather simple and is accounted for below)
- Verbs: strong, doubling, weak and extra weak. (Rather basic)
- Other inflections of the verb than infinitive and past. (Rather basic too)
- Adverbial phrases. (do.)
- Copula (equivalent to the verb 'is' in english) and auxiliary verbs (as e.g. 'be', 'do', 'have')(Self explanatory).
- Non-verbal sentences.¹

This list could of course be extended, but covering those parts of the ancient Egyptian grammar would make it nearer to being usable. Some (e.g. gender of nouns, verbs and adjectives) are surely easier to implement than others (e.g. non-verbal sentences).

6.2.2 Exceptions Handling

Like every grammar, ancient Egyptian grammar has rules and exceptions to these rules. One does not have to read much about ancient Egyptian to realise that ancient Egyptian grammar rules have a lot of exceptions, special cases and irregularities attached to them.

But how would our program and CHR_G perform if we imagine that exceptions to all the general grammar rules were to be implemented?

¹('Non-verbal sentences' have not been mentioned before but is an example of an 'exotic' feature of the ancient Egyptian language. We do not know the number of these in actual use, but in order to make our grammar more realistic, we would find it relevant. For a treatment of those, see [Gar27], p. 34.

Of course, with more rules, more constraints would be produced. But if it is possible to regard a considerable number of exceptions as special situations, it would in many cases be possible to use simplification or simplagation rules instead of the widespread propagation, and this would prevent the number of constraints from growing uncontrollably.

6.2.3 Readability

Our grammar is not very extensive and partially due to this, it is quite readable as is. Partially this is also due to the fact that it has been separated into the four layers, and that it is written with the CHR_G-notation.

Readability When Scaling Up (One Aspect Of Extendibility)

If our grammar was to be developed to an extent where it were nearer to being finished, it would be less readable for several reasons:

- It is rather possible that we would have to use more special features of CHR_G like the $\$$ -notation and 'dirty tricks' of various kinds as mentioned in 6.2.5 This would obviously require the user to have more knowledge about the notation and functionalities of CHR_G. But still, compared to using 'pure' Prolog-notation, we would, thanks to CHR_G, still have a layer that simplifies the expressions in a way that is far more readable.
- If the program was to be scaled up it would of course be less readable because of the length of the code itself, more rules gives more to read and to understand, naturally. But apart from this, and that the fact that we had to use special notation, as mentioned above, it would be less readable because we were to implement rules for all sorts of irregularities and exceptions, thereby introducing a lot of special rules for special cases. This would easily disturb the overview of the program, but this would not be possible to prevent in any grammar.

6.2.4 Generality of our Solutions

If we take one step back and look at the program and the ancient Egyptian grammar, one could think about how to reuse the solutions presented in the program to implement some of the large amount of remaining issues of the ancient Egyptian grammar. In what situations are the same type of solutions applicable?

One could e.g. use the solution dealing with the number agreement of the subject and the verb on other aspects of ancient Egyptian. The gender (masculine/feminine) agreement could be implemented the same way. The gender applies to nouns, verbs, genitive as well as adjectives ([CM98], pp.

40, 93). A rule implementing the noun and adjective gender agreement could look like this:

```
n(Gender), adj(Gender) ::> np(Gender).
```

The gender could then be introduced in the lexicon layer:

```
[a1] ::> n(masculine). % a man
[e59,b61,a38] ::> adj(masculine). % good, perfect
[e59,b61,a38, d24] ::> adj(feminine). % good, perfect
```

The example above could be implemented in a more intelligent way using an *adjective inflection* context sensitive rule

```
adj(masculine), adj_inflection(Gender) ::> adj(Gender).
adj(masculine) -\ [d24] ::> adj_inflection(feminine).
[e59,b61,a38] ::> adj(masculine). % good, perfect
```

Furthermore, the remaining unimplemented properties of the grammar symbols could be added; e.g. verbs have another property in addition to the number, tense and transitivity properties. It is a sub class property, which holds one of the following values: **extra_weak**, **weak**, **doubling** and **strong** ([CM98], p. 51). This could be added at the lexicon layer:

```
[a38, a41] ::> v(extra_weak) % to give
```

In the previous example the number, tense and transitivity attributes were omitted!

The class and the gender then dictates how the tense inflection is done, which would imply some changes and extensions to the inflection rules.

6.2.5 Noise Reduction

One way of reducing the noise, would be to incorporate sentence semantics on sentence level.

In some (lucky) cases there is an overlapping between the readability requirement and the reduction of noise. An example is the use of simplification or simpagation instead of propagation. With simplification and simpagation rules we are able to produce constraints that replaces exiting ones instead of just adding new ones to the pool. Hereby we can reduce the number of ambiguities and thereby reduce the noise in the output.

If we consider a simple rule like:

```
n(plu) -\ [c15] ::> p.
```


we immediately see a rule for determining a preposition. But what we also see by the propagation-arrow, that it possibly exists together with other interpretations of the C15-sign, being in the same context.

There are reasons (which are already explained) why we cannot use simplification for the determination of prepositions. But if contrary to this fact we imagine that it *was* however possible, we could instead write the rule as follows:

```
n(plu) -\ [c15] <:> p.
```

Then we could immediately see that we have a rule for determining a preposition - from this specific context - which stands alone. This could also be interpreted as very basic and general knowledge which does not have to be tested by other rules. The example rule is in some sense 'universal' since it can be read: "*in case we find a noun in plural followed by a [c15], we are always sure that the [c15] is a preposition*".

6.2.6 Scaling Up

Our program is a kind of prototype and is also an experiment to some degree. But the really interesting experiment, however, is yet to be done. What will happen when the program is scaled up, so the program has a complete lexicon and all the morphological rules? Will the program then produce extensive amount of noise and *wrong* interpretations? One can argue that if our *little* program creates that much noise (as mentioned earlier), the full-scale program will be useless. We do not think so! Even if it were the case that a lot of noise was created, it would be a very interesting experiment anyway! One of our motivations for creating the program in the first place, was to see if the program, based only on the distilled ancient Egyptian grammar knowledge, could find alternate meanings from already translated sentences and texts. This brings in the perspective that in dead languages like the ancient Egyptian, all known text was written a long time ago, which on the one hand means that much can have happened to the texts in the form of missing parts, scratches and the like. On the other hand it is quite special to dead languages that some words or sentences are so ambiguous according to modern theories that there is disagreement on what they actually mean. Here we could see an interesting use of NLP in Egyptology.

6.3 Conceptual Requirements Of The Formalism

6.3.1 Readability

Taking into account that we spent one month learning about ancient Egyptian grammar, one month learning CHRGE and writing the program, and then one month writing the report, we must say that the formalism it quite

easily learnt. To this might be added that none of us were very much acquainted to NLP from the beginning.

6.3.2 Limitations to CHR_G

There are some limitations to CHR_G, e.g. the way the `optional()` function is implemented make certain rules impossible. If one considers

```
a, optional(b), c ::> d
```

there is no problem since it is translated into the rules

```
a, c ::> d
a, b, c ::> d
```

which are mutually independent. However, if one then considers:

```
optional(a), b, c ::> d
```

there is a problem since it is translated into the rules

```
b, c ::> d
a, b, c ::> d
```

which means that both will apply if the latter applies. This could be avoided by implementing some sort of rule dependency like

```
if (a, b, c) then ...
else if (b, c) then ...
```

implicating that an instance of `b, c` only is checked if there is not an instance of `a, b, c`. Equally, the rule

```
a, b, optional(c) ::> d
```

is problematic.

Another limitation to the `optional()` function is that it cannot be used in the context sensitive rules. In most cases it is possible to express the rule another way, preserving the same meaning. This usually causes more rules and becomes for that reason less readable.

Chapter 7

Conclusion

We have made a program that recognises a very limited amount of sentences. But those sentences are formulated by abstract rules such that more entries could be added to the lexicon and the program would recognise many more sentences with not only the same basic structure as our example sentences, but also many sentences composed by the same constituents.

We have looked at some possible next-step-implementations, e.g. the easy implementation of gender and gender agreement - an example also showing that at least some of our methods are reusable in other contexts.

Those next-step-implementations could be done one step at a time, thereby allowing recognition of more and more possible input sentences.

We started out by wanting to know of some of the general problems connected to representing and handling natural language (grammar) in a computer, which is an activity belonging to the scientific field of NLP. Most generally we have experienced some of the many problems of ambiguity and especially we have seen how effective a tool context sensitive rules are when dealing with this matter. With those it is possible to narrow down very specific cases, thereby making the grammar grasp meanings very precisely.

As we see it we have in some sense done a 'case study' in linguistics, learning a grammar and trying to formalise it. But contrary to linguists, we did not have as our goal to find out anything interesting about the language modelled, but instead the focus have been on finding out something about the tool - of course when used with this specific language.

As an answer to the initial question of which are the 'key design decisions', we have come across - maybe not the key decisions of any NLP project, but at least the ones resulting from the focus we formulated early on in the process: An example is the two versions of our program (th NPPP and the NP(di)). Here we had a simple and efficient version, but wanted something more readable, i.e. more consistent with traditional linguistic analysis. Therefore we chose a less efficient and more noisy solution to the

problem of conjunctions.

We also made some decisions on how to structure our program: In order to confine to the conventions that we find in NLP, we made a distinction between the different layers of our program:

Syntax rules, Constituent rules, Morphological rules and Lexicon rules. According to this we have been careful where to place e.g. rules that handle inflections, since they are not necessarily obvious to place in one layer instead of another.

For the same reason we have chosen not to have other than word classification and production rules in the lexicon and we have chosen to add as little syntactic information about the words as possible.

Early in the process of modelling the grammar, we ran into several problems that prevented our grammar from being able to recognise even very simple sentences. This being due to the three main problems that we have described in detail throughout the report: the problem of conjunction, the problem of omitted subject and the problem of various uses of the same sign.

But despite of those problems we got a grammar running, that were using abstract sentence definitions, i.e. definitions of a structure and the elements in that structure, that were not specific to only one example sentence. This is in great part thanks to the context sensitivity rules that are central to CHR_G.

CHR_G are easily learnt, in the basic form, which makes it possible rather quickly to learn to write grammars that can be run through the Prolog prompt. This has been a great advantage to us, together with the robustness of CHR_G. We find CHR_G a good experimental tool, where it is also possible to get temporarily useful output, while developing the grammar, though it might not be actually running (i.e. parsing whole sentences).

As is probably the case with all formalisms, CHR_G has problems and limitations, some of which we looked at in the discussion chapter, e.g. the `textttoptional()` notation.

7.1 Perspective

As mentioned we spent a month on respectively learning some ancient Egyptian grammar, modelling it and writing the report. This way of scheduling, in addition to our experimental approach, has to our own opinion resulted in the report being thought and written maybe more from a linguist's perspective than from a computer scientist's. At least we have found ourselves spending more time on the purely linguistic aspects than on the more typically computational problems. But this does not mean that we only had the perspective of the linguist. Trying out a formalism to become familiar with it, through a 'case study', might be highly relevant to computer scientists.

”...Linguists don’t really design formalisms, or at least, they don’t seem to design very good ones. It is the business of computational linguists to do this, and this is what we are skilled at. But we’ve got to design formalisms that linguists will use, to make sure that they don’t come up and start using formalisms that are unimplementable. Then they’ll do all their work finding out all these wonderful facts about language, even writing them down, in a way that we can deal with. We want to come up with formalisms that we can get linguists to adopt, as a Trojan horse, to attract them to what we believe is the right way of doing things, to ways which will help us to do our work. It takes a fair amount of care and attention to design appealing formalisms...”

Ronald M. Kaplan, ‘*Three Seductions of Computational Psycholinguistics*’ (in Mary Dalrymple, Annie Zaenen, John Maxwell III, and Ronald M. “Formal issues in Lexical Functional Grammar”, University of Chicago Press 1994).

Chapter 8

Dictionary

Ambiguity [Wil00]

An ambiguity is a situation where more than one meaning is possible in a sentence. We consider three types of ambiguity:

(There can be situations where more than one of these is present)

Word Sense Ambiguity

A kind of ambiguity where what is in doubt is what sense of a word is intended. One classic example is in the sentence "John shot some bucks". Here there are (at least) two readings - one corresponding to interpreting "bucks" as meaning male deer, and "shot" meaning to kill, wound or damage with a projectile weapon (gun or arrow), and the other corresponding to interpreting "shot" as meaning "waste", and "bucks" as meaning dollars. Other readings (such as damaging some dollars) are possible but semantically implausible. Notice that all readings mentioned have the same syntactic structure, as in each case, "shot" is a verb and "bucks" is a noun.

Structural Ambiguity

A form of ambiguity in which what is in doubt is the syntactic structure of the sentence or fragment of language in question. An example of pure structural ambiguity is "old men and women" which is ambiguous in that it is not clear whether the adjective old applies to the women or just to the men. Frequently structural ambiguity occurs in conjunction with word-sense ambiguity, as in "the red eyes water" which could signify "the communist looks at water": (S (NP (ART the) (N red)) (VP (V eyes) (NP (N water)))) or alternatively "the reddened eyes drip tear fluid" (S (NP (ART the) (ADJ red) (N eyes)) (VP (V water))).

Referential Ambiguity

A type of ambiguity where what is uncertain is what is being referred to by a particular natural language expression. For example, in John hit Paul. He was angry with him. it is not entirely clear to whom the pronouns "he" and "him" refer - it could be that the second sentence explains the first, in which case the "he" is John, or it could be that the second sentence gives a consequence of the first, in which case the "he" is Paul.

Bottom-Up Parser[Wil00]

A parsing method that proceeds by assembling words into phrases, and phrases into higher level phrases, until a complete sentence has been found.

CHR[Fr98]

CHR are a high-level language to write constraint systems. CHR make it easy to define constraint reasoning: simplification and propagation as well as incremental solving (satisfaction) of constraints. Also, CHR can be used

- as general purpose concurrent constraint language with ask and tell.
- as fairly efficient production rule system.
- as special kind of theorem prover with constraints.
- as system combining forward and backward chaining.
- for bottom-up evaluation with integrity constraints.
- for top-down evaluation with tabulation.
- for combining deduction, abduction and constraints.
- as high-level language for manipulating attributed variables.
- for parsing with executable grammars.

Constraint[Dic02]

A Boolean relation, often an equality or inequality relation, between the values of one or more mathematical variables (often two). E.g. $x \neq 3$ is a constraint on x . Constraint satisfaction attempts to assign values to variables so that all constraints are true. (source: <http://www.dictionary.com>)

DCG

Definite Clause Grammars (DCGs) are convenient ways to represent grammatical relationships for various parsing applications. Typically it is used on either a character basis, tokenizing an input stream, or on a word basis, interpreting either formal or natural language input.

Morphology[Wil00]

The study of the analysis of words into morphemes, and conversely of the synthesis of words from morphemes.

NLP[Dic02]

Natural Language Processing (NLP):
Computer understanding, analysis, manipulation, and/or generation of natural language. This can refer to anything from fairly simple string-manipulation tasks like stemming, or building concordances of natural language texts, to higher-level AI-like tasks like processing user queries in natural language. (source: <http://www.dictionary.com>)

Punctuation Marks[Cry92]

A set of graphic sign used in written language to signal certain important grammatical and attitudinal contrasts. Its three main functions are: to separate units in a linear sequence (e.g. a space separates words, a period separates sentences); to indicate when one unit is included within another (e.g. a parenthesis, quotation marks, or a pair of commas); and to mark a specific grammatical or attitudinal function, such as a question (question-mark), exclamation (exclamation mark), or the notion of possession (the apostrophe).

Robustness[Wil00]

A parser or other NLP algorithm is robust if it can recover from or otherwise handle ill-formed or otherwise deviant natural language expressions. For example, we would not want a syntactic parser to give up as soon as it encounters a word that is not in its lexicon - preferably it should try to infer the lexical category and continue parsing (as humans do when they first encounter a new word).

Semantics[Wil00]

Semantics is the study of meaning (as opposed to form/syntax, for example) in language. Normally semantics is restricted to "meaning out of context" - that is, too meaning so far as it can be determined without taking context into account.

Syntax[Wil00]

Syntax means the rules of language that primarily concern the form of phrases and sentences, as distinct from the substructure of words (see morphology) or the meaning of phrases and sentences in or out of context.

Appendix A

Test sentences outputs

```
text(0,11),
token(1,2,c11),
token(3,4,e28),
token(5,6,c15),
pron(6,7,sing),
token(8,9,b3),
n(7,10,sing),
v(1,6,sing,past,1),
np(6,7,sing,A,B),
np(6,7,sing,obj,dir),
np(7,10,sing,obj,dir),
all(0,11),
token(2,3,a38),
token(4,5,f19),
token(6,7,a1),
token(7,8,c20),
token(9,10,a9),
vp(1,5,sing,infinitive,1),
vp(1,6,sing,past,1),
np(6,7,sing,subj,subj_type),
np(7,10,sing,_C,_D)
np(6,10,_E,obj,dir)
```

Program output when parsing sentence:

"I buried the old."

(Transitivity 1)

```

leafchick(1,3),
pp(5,10),
p(5,6),
pp(5,10),
pp(5,11),
pp(11,16),
pp(5,16),
pp(5,16),
p(22,23),
nppp(16,28),
text(0,29),
token(1,2,c20),
token(3,4,a38),
token(5,6,c15),
token(7,8,d25),
token(9,10,f9),
n(6,11,plur),
token(12,13,e35),
token(14,15,a38),
n(12,16,sing),
pron(15,16,sing),
token(17,18,a54),
token(19,20,b1),
token(21,22,d3),
token(22,23,c15),
token(24,25,b3),
token(26,27,f35),
n(23,28,sing),
vp(3,5,sing,infinitive,2),
vp(3,6,sing,past,2),
np(6,10,sing,subj,subj_type),
np(6,10,sing,obj,ind),
np(6,11,plur,obj,dir),
np(6,11,plur,obj,ind),
np(12,16,sing,obj,ind),
np(16,22,plur,_I,_J),
np(23,28,sing,_K,_L),
np(27,28,sing,_M,_N)
p(5,6),
pp(5,11),
pp(5,11),
pp(5,11),
p(11,12),
pp(5,16),
pp(5,16),
nppp(6,16),
pp(22,28),
nppp(6,28),
all(0,29),
token(2,3,b1),
token(4,5,a41),
token(6,7,d24),
token(8,9,b28),
token(10,11,d3),
token(11,12,c15),
token(13,14,c11),
token(15,16,a1),
s(1,16,past),
token(16,17,e35),
token(18,19,e28),
token(20,21,f35),
n(16,22,plur),
token(23,24,e35),
token(25,26,c21),
token(27,28,a1),
pron(27,28,sing),
v(3,6,sing,past,2),
np(6,10,sing,_A,_B),
np(6,10,sing,obj,dir),
np(6,11,plur,_C,_D),
np(6,11,plur,obj,ind),
np(12,16,sing,_E,_F),
np(15,16,sing,_G,_H),
np(16,22,plur,obj,dir),
np(23,28,sing,obj,ind),

```

*Program output when parsing sentence:
(I) gave bread to the hungry and clothes to the naked.
(Transitivity 2 + omitted subject + conjunction)*

```

leafchick(1,3),
pp(5,10),
p(5,6),
pp(5,10),
pp(5,11),
pp(11,16),
pp(5,16),
pp(5,16),
p(22,23),
nppp(16,28),
text(0,38),
token(1,2,c20),
token(3,4,a38),
token(5,6,c15),
token(7,8,d25),
token(9,10,f9),
n(6,11,plur),
token(12,13,e35),
token(14,15,a38),
n(12,16,sing),
pron(15,16,sing),
token(17,18,a54),
token(19,20,b1),
token(21,22,d3),
token(22,23,c15),
token(24,25,b3),
token(26,27,f35),
n(23,28,sing),
pron(27,28,sing),
token(29,30,a38),
token(31,32,f19),
token(33,34,a1),
token(34,35,c20),
token(36,37,a9),
s(28,37,past),
v(3,6,sing,past,2),
np(6,10,sing,_A,_B),
np(6,10,sing,obj,dir),
np(6,11,plur,_C,_D),
np(6,11,plur,obj,ind),
np(12,16,sing,_E,_F),
np(15,16,sing,_G,_H),
np(16,22,plur,obj,dir),
np(23,28,sing,obj,ind),
vp(28,32,sing,infinitive,1),
vp(28,33,sing,past,1),
np(33,34,sing,subj,subj_type),
np(34,37,sing,_Q,_R),
np(33,37,_S,obj,dir)
p(5,6),
pp(5,11),
pp(5,11),
pp(5,11),
p(11,12),
pp(5,16),
pp(5,16),
nppp(6,16),
pp(22,28),
nppp(6,28),
all(0,38),
token(2,3,b1),
token(4,5,a41),
token(6,7,d24),
token(8,9,b28),
token(10,11,d3),
token(11,12,c15),
token(13,14,c11),
token(15,16,a1),
s(1,16,past),
token(16,17,e35),
token(18,19,e28),
token(20,21,f35),
n(16,22,plur),
token(23,24,e35),
token(25,26,c21),
token(27,28,a1),
s(1,28,past),
token(28,29,c11),
token(30,31,e28),
token(32,33,c15),
pron(33,34,sing),
token(35,36,b3),
n(34,37,sing),
vp(3,5,sing,infinitive,2),
vp(3,6,sing,past,2),
np(6,10,sing,subj,subj_type),
np(6,10,sing,obj,ind),
np(6,11,plur,obj,dir),
np(6,11,plur,obj,ind),
np(6,11,plur,obj,ind),
np(12,16,sing,obj,ind),
np(16,22,plur,_I,_J),
np(23,28,sing,_K,_L),
np(27,28,sing,_M,_N),
v(28,33,sing,past,1),
np(33,34,sing,_O,_P),
np(33,34,sing,obj,dir),
np(34,37,sing,obj,dir)

```

Program output when parsing sentence:

*(I) gave bread to the hungry and clothes to the naked. I buried the old.
(Transitivity 2 + omitted subject + conjunction + two sentences)*

Bibliography

- [Bra90] Ivan Bratko. *Prolog, Programming for Artificial Intelligence*. Addison Wesley, 2nd edition, 1990.
- [Chr02a] Henning Christiansen. Chrgrammars <http://www.dat.ruc.dk/~henning/chrg/PapersOnCHRG/CHRG.tlp.pdf>, 2002.
- [Chr02b] Henning Christiansen. User's guide to chr grammars (chrgs): <http://www.dat.ruc.dk/~henning/chrg/CHRGusersGuide.html>, 2002.
- [CM98] Mark Collier and Bill Manley. *How to Read Egyptian Hieroglyphs*. British Museum Press, 4th edition, 1999 (first published 1998).
- [Cov94] Michael A. Covington. *Natural Language Processing for Prolog Programmes*. Prentice Hall, Inc. (Simon & Schuster Company), Englewood Cliffs, New Jersey, USA, 1 edition, 1994.
- [Cry92] David Crystal. *The Penguin Dictionary of Language*. Penguin, Suffolk, England, 2nd edition, 1999 (First published 1992).
- [Dic02] Dictionary.com. Dictionary.com: www.dictionay.com, 2002.
- [Fr98] Thom Frhwirth. Theory and practice of constraint handling rules: <http://www.pst.informatik.uni-muenchen.de/personen/fruehwir/jlp-chr1/>, 1998.
- [Gar27] Sir Alan Gardiner. *Egyptian Grammar*. University Press, Oxford; Griffith Institute, Ashmolean Museum, 3rd edition, 1982 (first published 1927).
- [GHvdP93] Nicolas Grimal, Jochen Hallof, and Dirk van der Plas. *Hieroglyphica*. Centre for Computer-aided Egyptological Research (CCER), 2nd edition, 2000 (first published 1993).

- [Hie02] Hieroglyphs.net. Egyptological computing:
<http://www.hieroglyphs.net/000501/html/000-048.html>,
January 2002.
- [Ned02] Mark-Jan Nederhof. Sign lists:
<http://www.dfki.de/~nederhof/AEL/signlists.html>, 2002.
- [Ros96] Serge Rosmorduc. *Analyse morpho-syntaxique de textes non
ponctues*. PhD thesis, Ecole Nationale Suprieure de Cachan,
December 1996.
- [Ros02] Serge Rosmorduc. Hiero \TeX :
<http://www.iut.univ-paris8.fr/~rosmord/archives/>, 2002.
- [Wil00] Bill Wilson. The natural language processing dictionary:
<http://www.cse.unsw.edu.au/~billw/nlpdict.html>, 2000.