Swedish noun and adjective morphology in a natural language interface to databases

Peter Cedermark
peced@stp.ling.uu.se

Master's thesis
Uppsala University
Department of linguistics
Language engineering programme

14th February 2003

Supervisors:
Anna Sågvall-Hein, Uppsala University
Gregor Jonsson, Dialogue Technologies AB

Abstract

This thesis describes a component for the analysis of Swedish noun and adjective inflection, taking noun and adjective tokens as input and mapping them to semantic representations. This component is part of the Swedish version of the Phoenix natural language interface to database systems, which maps questions in natural language to the corresponding SQL queries. The testing was made using sentences containing all the inflected forms of the nouns and adjectives covered by the rules. During testing, the semantic representations were examined and rewritten as needed. When the tests were concluded and the rules rewritten, all of the nouns and adjectives used in the test sentences produced all of the intended semantic representations, and no other representations were produced.

Keywords: Natural language interface, NLI, computational morphology, semantic representation, dialogue systems, Swedish grammar

Contents

Acknowledgements

First of all, I would like to thank Joakim Ingers, Gregor Jonsson, Christopher Dahrén, and Björn Alsén at Dialogue Technologies AB for making this thesis possible at all; my supervisor, professor Anna Sågvall-Hein for insightful comments, productive discussions and guidance; Karl Lundstedt for the syntactic rules and invaluable help with testing and debugging; Hubert Lehmann and Thomas Haag at linguatec Entwicklung & Services GmbH for professional help and answers; Kristina Nilsson for proof reading and LaTeX assistance. Also, a big thank you to the other students in the Skrubben society, as well as to my beloved Anna Wahlström, for all your support.

1 Introduction

A growing amount of today's information is stored in relational databases, where Structured Query Language, or SQL, is commonly used as the language for adding, changing and deleting entries, as well as for queries. Although it is very powerful, it is not easy to learn; therefore, it is common for a few SQL experts to either search on demand for information, or construct easy-to-learn graphical interfaces. It is difficult for inexperienced users to enter ad hoc queries and receive the information they are looking for.

A different approach to database manager (DBM) querying is to use a natural language interface, where the user phrases a query in his or her own (human) language, and the information is presented in a similar manner. This eliminates the need to learn how to use either SQL or the DBM software in order to find the requested information. Such an interface makes all the information in a relational SQL database available to inexperienced users as well as SQL experts, and therefore they can be used to answer queries which otherwise have to be answered by another human, who first rephrases the question in SQL and then presents the answer. When equipped with a speech-to-text module, a natural language interface, or NLI, opens up a new world of possibilities.

Using a natural language interface together with a speech-to-text module, a travel agency or a support centre can redirect some of the calls to an automatic answering service. A lot of time is spent waiting for an operator to be available, and even when the charge for the call is no different from a local call, an automatic service may decrease the need to wait for simple answers to simple questions, so the savings in time will also be substantial.

1.1 Purpose

The purpose of this thesis is to devise and implement a subset of the master grammar specification using the Phoenix formalism, which is a multilingual natural language query interface to relational databases. The language part in question concern morphological and lexical features for the analysis of Swedish nouns and adjectives. The implementation is focused on the analysis grammar only; implementing a suitable generation grammar is beyond the scope of this thesis.

1.2 Why use natural language interfaces?

Consider the following example by Lehmann (2001) on page 8, in the context of a software development application:

- Who coded a module?

This query may also be phrased differently, depending on the grammar. All of the examples below are paraphrases with the same intentional meaning as the question above, i.e., the person stating the query wants to find information about who codes modules:

- Find programmers that write modules.
- Show me who codes modules.
- List the people who code modules.
- Give me a list of the people who write a module.

The same query in SQL may, for example, correspond to the following:

```
SELECT DISTINCT x1.SERIAL,x1.EMPNAME,x3.MODULE
FROM Phoenix.SEMPLx1,Phoenix.SMODULES x3
WHERE x3.PROGRAMMER=x1.SERIAL
```

The differences between the SQL query and the queries stated in natural language are obvious. The natural language question is much more forgiving when it comes to syntax – the syntax can vary[1] while the intention remains unchanged – and it is also much easier to correct the input in case of misspellings and/or grammatical errors. Furthermore, the user does not have to know about the internals of the database in terms of table structure to find relevant information.

1.3 Phoenix

Phoenix is a multilingual NLI for querying relational databases[2] which utilize SQL. The idea behind it is that computer users[3] who wish to find obtain information from a database or a database coupled web site should not have to learn a complex query language to do so, but instead formulate the query in a way similar to how he or she would ask it when communicating with a human being.

Phoenix utilizes so called *conceptual schemas* for the representation of the lexicon. Lehmann (2001) defines a conceptual schema as follows: *A consistent collection of sentences expressing the necessary propositions that hold for a universe or discourse.* In this context, a conceptual schema contains information about classes of objects in the domain, **entities**, and sets of **relations** between them. Phoenix uses three kinds of schemas:

1. The **user schema** contains domain specific information. Such information is, for example, information about entities such as clients, bank accounts, money transfers and so on, if the application is used in a bank. Similarly,

[1] Provided that the grammar developer has included rules and mechanisms for it.
[2] For example IBM DB/2, Oracle, MySQL et cetera.
[3] Throughout this thesis, the term “user” will be used for the person stating the query.

in a software development application, the entities could be programmers, modules and version numbers for software.

2. The **base schema** contains information and entities which are not specific for any domain or application. These words are mainly from the closed categories,[4] but also some words in other categories, for example the verbs *ha* (have) and *vara* (be) and nouns such as the names of months, weekdays and fixed holidays. The base schema also contains a concept hierarchy, with classifications such as *person*, *quantity*, and *event*, together with descriptions of the relations which hold between them.

3. Finally, the **meta schema** contains the words and concepts necessary to be able to talk about the system itself. The meta schema thus allows the user to ask questions like *What words do you know?*

[4] Prepositions, conjunctions, subjunctions, numerals, ordinals, and pronouns.

2 The Phoenix formalism

2.1 How it all works

Here follows a condensed description of all the steps involved in the translation of a query in natural language to the corresponding SQL query. Figure 1 illustrates this process and shows the information used in each processing step.

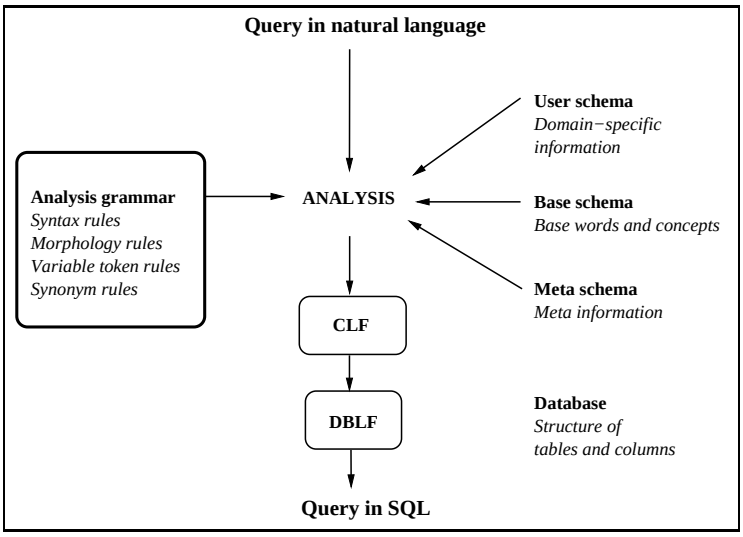

Figure 1: Overview of the modules involved in SQL generation.

1. The user states the query in natural language.
2. The query is analyzed lexically and syntactically, using the previously mentioned schemas and the rules in the grammar. The format of the output is in concept logical form, or CLF. This is an extended version of predicate logic, which is linked to the information in the conceptual schema and used as an intermediate representation of the query.
3. The CLF is translated to database logical form, or DBLF. This format contains references to the tables and the columns in the database. This step is automatic.
4. As the final step, the query is converted from DBLF to the corresponding SQL query. This step is also automatic.

The following subsections describe the Phoenix system and the XLDG formalism in more detail.

2.2 XLDG: a detailed look

XLDG is the grammar formalism used for the analysis grammar. The notation is similar to standard Prolog.

The grammar rules written in XLDG are not used directly as they are. They are first converted into a DCG, or Definite Clause Grammar, format. The DCG rules are then compiled with the schemas to produce the actual Prolog code. XLDG rules are considerably easier to read and construct than Prolog; it can be said that XLDG provides syntactic sugar for DCG.

XLDG grammars are basically made up of phrase structure rules, with checking and setting of features as in ordinary HPSG, or Head-driven Phrase Structure Grammar (Pollard and Sag 1994). XLDG is designed to handle one sentence or command at a time.

Each grammar rule has both syntactic and semantic parts. The syntactic parts are used for checking and unification, and the semantic parts are used for compositional construction of the conceptual logic form, or **CLF**. The CLF representation is achieved by compositional evaluation of the semantic tree. Because the module responsible for the conversion of the semantic tree to CLF is non-deterministic, a query may produce more than one CLF variant.

CLF is purely logical and is not associated in any way to the DBM used. It is totally language-independent – the corresponding CLF representation of a query does not depend on the corresponding SQL query, but only on the schemas and the grammar rules. The following is an example of a CLF representation of the question *Are the countries European?* (The numbers and colons in bold are added just for clarity, and are not parts of the CLF).

```
1: clf(9, 1, 1,
2: query(yesno(nodoch, noreverse),
3: exist(y39,
4: instance(country, y39) &
5: instance(european, y39)))).
```

One of the features of CLF is that it changes depending on how the question is posed. The above example is a question which can be answered with a simple “yes” or “no”. If, on the other hand, the question is “Show all European countries”, the corresponding CLF question looks like the following:

```
1: clf(9,1,1,
2: query(report,
3: set(y39,
4: instance(country, y39) &
5: instance(european, y39)))).
```

As seen in line 2 of the two CLF representations, the query type is changed from `yesno` to `report`.

The last step before the SQL is generated from the CLF is taken with the conversion to DBLF. This format contains crucial information about the internal structure of the database used (tables and columns), which provides a pseudo-logical expression, which is then automatically converted into the actual SQL query.

The XLDG formalism is made up of primitives, constructs, and rules, all of which are described in the following sections.

2.3 Primitives

The smallest recognizable units are characters, digits, punctuation marks, and symbols. These have to be defined beforehand, and may vary between languages. They are defined as Prolog clauses in the following form:

```
100.0:- primitive(name,'character_string').
```

It is possible to define multiple sets of primitives – which may overlap – and assign different names in the first argument. This enables the grammar to operate on sets of characters; instead of having to write separate rules where one character is the only difference, the grammar developer can define two sets of primitives. This is especially useful when constructing rules for expressions which have multiple writings; some users may be used to writing a dot as a decimal point, while other rather use a comma.

Here are a few examples of definition of sets of primitives, which give a hint of the overlapping possibilities.

```
primitive(alpha, 'abcdefghijklmnopqrstuvwxyzåäö').
primitive(digit, '0123456789').
primitive(alnum,
'abcdefghijklmnopqrstuvxyzåäö0123456789').
primitive(posdigit, '123456789').
```

There is one special primitive, which is used to denote the beginning and the end of the whole input string. The functor for this special primitive is `surrounding`.

2.4 Constructs

Constructs closely correspond to lexical categories (Lehmann 2001). Each string is assigned a construct from the grammar rules, and are subcategorized by features. The constructs are then operated on by grammar rules, which as input take constructs and as output give either one or more new constructs, or permute the input constructs. The naming of the construct is not coded into Phoenix, and can

therefore be named at the grammar developer's will, although it is better to use familiar names. A construct can be assigned to sentences, phrases, individual words, or even parts of words, which is useful in morphology rules when working with various affixes.

2.5 Rule types and annotation

All rules consist of an *output structure*, a *separator*, and a *structural description*. The separator is always two dashes, followed by a greater than sign (`-->`). The output structure is written on the following form:

```
construct name(feature,
                          ipe(word structure test),
                          fpe(semantic call)).
```

The `feature` argument is a comma-separated list of feature settings. The predicate with the functor `ipe` is mainly used in morphological rules, to determine whether a word can take a prefix, a suffix, or neither. Finally, the third argument, `fpe(semantic call)` is used to build the corresponding part of the semantic tree from which the CLF, DBLF and SQL are created. The FPE and IPE tests are described in further detail in sections 2.8.2 and 2.8.1, respectively.

There are four kinds of rules in XLDG that a grammar writer can use:

1. **Fixed token rules**. These rules recognize lexical items and assign a category, a set of features, the IPE, and the FPE. The output follows the structure outlined above.

2. **Synonym rules**. These rules are used to show that two strings are equivalent with regard to features, construct, IPE, and FPE. These are simply written as `'string' --> 'synonym'`.

3. **Variable token rules**. These rules are used to assign a category, a set of features, IPE, and FPE to a string of primitives. They are useful when analyzing strings following a certain pattern, for example date expressions. There are predicates for denoting the Kleene star (zero or more characters) and the plus sign (one or more characters), and there are also mechanisms for denoting context limits.

4. **Grammar rules**. Grammar rules are the most common type of rules. Constructs or combinations of constructs and strings are used as input, and one or more constructs are produced in the output. Today, the only limitation to the number of construct is that rules with only one input construct cannot have more than one output construct. This may be changed in the future.

2.6 Dictionary entries

An example of a dictionary entry is the following (the line breaks are added just for readability; the rule itself is usually written on a single line). The numbers to the left, in bold letters, are not part of the rule, but are added only for the purpose of the explanations in the following paragraphs.

```
1: b_dictionary(who,
2: np(np(99020,
3: feature(nom=1,dat=1,acc=1, ...),0),
4: [test(0,2)],
5: wque(str(''who'')))  =>
6: str(''who'')).
```

The information on line 1 states which schema the entry belongs to, in this case the base schema, and the internal name of the term. The internal name can be arbitrary chosen. The alternatives to `b_dictionary` are `m_dictionary` (the meta schema) and `u_dictionary` (the user schema).

Line 2 consists of grammar information and a rule number. The grammatical information says this word is a noun phrase, whose rule number is 99020. Line 3 is the list of features associated with the entry. Line 4 contains the IPE test (for more information on IPE tests, cf. section 2.8.1 on page 11).

Line 5 contains the semantic call for the rule, which is used to create the semantic tree representation, in this case an interrogative term, `wque`. Finally, the last line contains the surface form of the entry.

2.7 Features

2.7.1 Feature types

A feature in XLDG consists of a name and a value. There are three different kinds of features:

1. **Logical** features are those which have exactly two values, 0 and 1. These correspond to the Boolean values TRUE or FALSE. The feature and its value are written as `feature = value`, where `value` is either 0 or 1.
2. **Integer** features take positive integers as values. This feature type is useful for features which can have several values, for example different articles. One integer feature can thus be used in place of a combination of logical features, but at the expense of clarity; the grammar developer has to know what the different values mean. Integer features can be assigned descriptive names as to make it easier to read or edit them.

3. **Case features** are assigned values from a defined set of *case values*. These are somewhat similar to integer features, but instead of integer values, they have values in the form of strings. Case features are mainly used to denote various types of complementation which a verbal construct can take. Because verb phrases and verbal constructs are not a part of this thesis, case features will not be described here.

2.7.2 Declaring features

All features have to be declared, along with their respective type, before they can be used in the rules. Features are declared in the form of binary Prolog facts as in the following examples:

```
feature(nom, logical).
feature(lab, integer).
```

2.7.3 Setting features

A feature in a construct must have exactly one value. An explicit feature setting therefore always looks like `Feature = Value`.

When one or several constructs are transformed or combined, respectively, the feature set from one construct can be *inherited* by the new, resulting construct. One can say the features *percolate* to the new construct. This mechanism is very useful in morphology; for example, when a suffix is added to a noun, which already has a feature for gender, the gender should not change between different inflected forms, but instead percolate from one of the incoming constructs the resulting one.

Every construct in the body of a rule has a number; the first is denoted 1, the second 2, and so on. By putting a construct number as the first argument in the head of the rule, one can make sure that all features for it will percolate up into the new construct. Only one construct feature may percolate – the grammar developer has to decide which construct has the most reusable feature set.

Features can also be set or checked explicitly by simply writing the feature name, an equal sign (=) and its value in the new construct.

2.7.4 Checking features

One method for checking a feature's value is to look at its absolute value and state that it has to be equal to a certain value. When using integer features, which can take more than two values, it is useful to be able to tell whether a value is greater than, equal to, or smaller than a certain number. There are mechanisms for this,

which enables the grammar developer to define ranges of values. The mechanisms for this type of checking are listed in table 1.

Writing	**Meaning**
`gt(Feature,Value)`	The value for Feature is *greater* than Value
`lt(Feature,Value)`	The value for Feature is *less* than Value
`ge(Feature,Value)`	The value for Feature is *greater than or equal* to Value
`le(Feature,Value)`	The value for Feature is *less than or equal* to Value
`eq(Feature,Value)`	The value for Feature is *equal* to Value (equivalent with `Feature = Value`)
`ne(Feature,Value)`	The value for Feature is *not equal* to Value

Table 1: Feature checking predicates.

2.7.5 Using Boolean expressions

When checking features, the Boolean expressions AND and OR can be used, which are denoted as `&` and `!`, respectively. The usual priorities hold between them, and parentheses can be added to avoid potential ambiguities. The Boolean operator NOT does not have an operator of its own, but for integer and logical features, it is written as `ne(Feature, Value)`, cf. table 1.

2.7.6 Relative values

When combining two or more constructs, for example by forming a noun phrase with an article, two genitive nouns and one basic noun, it is useful to check that a certain construct has the same value, or range of values, for a feature as another construct. There are mechanisms for checking *relative* values in Phoenix as well. Consider the following example rule:

```
nomen(2, def=def(1), art=1,
ipe, fpe(adj(1,2))) -->
qu &
nomen(art=0).
```

This rule is for combining the first construct, a quantifier, with the second construct, a noun which is a proper noun phrase, to form a new noun phrase. Let us concentrate on what the bold letters mean.

The first argument, `2`, states that all the features in the second construct should percolate to the resulting construct. The next argument, `def=def(1)`, states that the feature `def` in the new construct will be set to the same value as in the first

construct. This way of setting features relatively can also be used to check features relatively:

```
nomen(2, ipe, fpe(adj(1,2))) -->
adject(pl=pl(2)) &
nomen.
```

The example above combines an adjective and a noun, which is a proper noun phrase, to form a new noun phrase. The feature `pl` in the noun – the second construct – is checked in the adjective, which is the first construct; the value for `pl` has to be the same in the adjective as in the noun for the rule to be applied.

2.8 IPE tests and FPE tests

These tests are used in morphological and semantic analysis, respectively. The IPE test is mainly concerned with word structure – affixes and starting and end points for the string – and the FPE test is used to build the semantic tree from which the CLF, DBLF and, ultimately, SQL is constructed. These tests are further discussed in the following sections.

2.8.1 IPE tests

The IPE test is the part of the head in the rule immediately following the set of features. The IPE test is in fact a Prolog list of the tests applied, so in dictionary entries – which have not undergone any tests – only a pair of square brackets are used inside the round parentheses ([]). All rules except grammar rules have an IPE statement in them.

The IPE test is also called a *word structure test*, and it is mainly used in morphological rules to determine whether a word can take an affix or not. It is written in the head of the rule, immediately following the set of features, as `ipe(ArgList)`.

There are a few different types of IPE test, depending on which checks the grammar developer wants a string to go through:

1. **The empty IPE.** This test should be applied for grammar rules and dictionary entries, where no IPE test is necessary. The empty IPE is simply written as `ipe`, without any parentheses, in XLDG files; in dictionary entries they are written as a pair of empty square brackets ([]).

2. **The null IPE.** This test makes the input string invisible to the parser and is written as `ipe(null)`. The most important example of rules with this IPE test is blanks between lexical items. They are not removed from the input strings, so the information about blanks is still available when recognizing affixes. Other examples where this test comes in handy is when dealing with

so-called polite words, i.e., words which do not contain any information relevant to the query, such as the English “please” and the corresponding *tack* or, in some contexts, *var god och*.... A string which has been assigned a construct with the null IPE does not become a part of the semantic tree representation of the query.

3. **The prefix test.** This test is written as `ipe(0,0)` in dictionary entries and fixed token rules. Strings which are recognized by a rule with this IPE test must not be preceded by any of the primitives, which should be the case for all prefixes. If prefixes are stacked, they usually follow a certain order, and by assigning the first prefix the prefix test, non-grammatical constructions can be avoided.

4. **The suffix test.** This test is written as `ipe(0,1)` in XLDG files and as `[test(0,1)]` in dictionary entries. This test makes sure that none of the declared primitives follow directly after the string input to the rule.

5. **The separate word test.** This test checks that the string input to the rule does not have a member of the primitive alphabet immediately preceding or succeeding it. The most common use of this test is in abbreviations, where a short abbreviation, for example “m” (for *metre*) should only be analyzed by a rule when it is written as a separate word. If this test is not present, any word beginning with “m” could potentially be parsed as an abbreviation of *metre*. Even if this interpretation would not get very far in a well-written grammar, they take up system resources because they are not thrown away immediately. This test is written as `[test(0,2)]` in dictionary entries and fixed token rules, and as `ipe(0,2)` in grammar rules.

These are the types of IPE tests which are used in grammar rules, dictionary entries, and fixed token rules. Other types of IPE tests are used in so called *variable token rules*, which are used to denote grammatic expressions following a certain format, for example notations for date and time.

2.8.2 FPE tests

The purpose of the FPE test – also referred to as the *semantic call* – is to provide the building blocks for the construction of semantic trees, whose construction is compositional. The semantic construction rules are independent of the language in which the questions are expressed. The available set of construction rules is fixed and cannot be extended by the grammar developer.

Every dictionary entry in the schema files have an FPE value, depending on which semantic analysis it should get if given as input in the query.

The functor for the FPE test is `fpe`, and it takes one argument. The argument taken depends on the semantics which are to be assigned to the resulting construct.

Consider the following – very general – context-free grammar rule by Lehmann (2001):

```
resultingConstruct(Features,
ipe(IpeSet), fpe(Fpe)) -->
construct1(FeatureSet1) &
construct2(FeatureSet2) &
...
constructn(FeatureSetn).
```

Inside the parenthesis in the `fpe` predicate, there is an argument which contains two things:

- The name of the semantic call and
- the constructs involved in it.

An example of how the semantic call looks is given below. A verbal construct, `vc`, is the result of combining a verbal construct and a noun phrase into a new verbal construct:

```
vc(1, cn = 1, ...ipe, fpe(nom(2,1))) -->
vc(...)
np(nom = 1).
```

`nom` is the name of the semantic construction rule which takes two arguments – in this case `vc` and `np` – and attaches the first as nominative of the second.

A special case is the empty FPE, which is written as `fpe(Construct)`. The variable `Construct` is substituted by the number of the construct involved. This semantic call is used when the resulting construct does not contribute to the semantic analysis, which is the case for morphological suffixes. This type of semantic call is written `fpe(1)` if the second construct is the suffix; the semantic value of the stem is passed on unchanged.

3 Swedish morphology: nouns and adjectives

This section discusses the approach used in the creation of morphology rules for Swedish nouns and adjectives. Adjectival expressions as a whole is covered, which also includes perfect participle and present participle forms of verbs, which we treat as adjectives in our grammar.

3.1 Reasons to use morphological rules

Granted, it is possible to have every word in every inflected form in the lexicon. This would lead to a mass of extra features, some of which would be difficult to understand and maintain; the lexicon also becomes very large, because nouns can potentially have up to eight different surface forms. Larger lexicons take longer time to search, and because a lot of information is stored in RAM in Phoenix, the impact on system resources would be significant.

By using morphological rules, the lexicon becomes smaller, which leads to better search performance, and it takes less time to add new entries because the grammar developer(s) in a best-case scenario only needs to add one entry with the proper features, instead of having to add up to eight entries. In my personal experience, the process of adding lexicon entries is prone to errors, which may become a serious problem when testing the grammar.

3.2 Overview of Swedish morphology

As for nouns and adjectives, the Swedish language relies quite heavily on morphological inflections. Swedish uses four natural genders: masculine, feminine, neuter, and reale. The grammar makes use of the grammatical genders neuter and uter, plus the natural genders masculine and feminine, so some agreement issues must be solved in order for the analysis grammar to work. The masculine and feminine genders only play a part in pronoun analysis, and because this thesis only concerns noun and adjective morphology, rules were written using neuter or uter gender only.

3.3 Noun morphology

Number: Swedish singular and plural forms may be discerned via suffixes or stem changes, and sometimes a combination of both. Some Swedish examples, where the whole stem is altered, are *man – män, gås – gäss*. When inflected, the suffix added often depends on how the word is spelled. There are also words whose singular and plural forms are identical, at least when in indefinite basic forms, for example *fönster, hus, djur*.

Swedish nouns are traditionally grouped into a set of *declinations* (Thorell 1977). Nouns which belong to one of these declinations follow a given inflection pattern when the number changes from singular to plural; the case remains basic, and the words remain in indefinite form. Table 2 shows the different declinations in Swedish, along with examples of words belonging to them. In a few cases, the stems between plural and singular forms differ, but the declinations only define the suffixes used.

Declination	**Suffix**	**Example**
first	-or	*flick/a-or, väsk/a-or*
second	-ar	*båt-ar, stol-ar, knut-ar*
third	-(e)r	*katt-er, parti-er*
fourth	-(e)n	*äpple-n, knyte-n*
fifth	-∅	*hus, mil*

Table 2: Swedish declinations with examples.

These declinations provide a good starting point for the noun morphology. This is not enough for the full noun morphology implementation; the reasons behind are discussed in section 4.3.1.

Species: In Swedish, changes in species are obtained via inflection. There is a large number of inflections for species, depending on the spelling of the word and also, in singular forms, partly on the gender; words with neuter gender have a tendency to have the suffix *-t* appended in singular basic definite form, and uter gender words tend to have the suffix *-n* appended. However, this is just a tendency which is not consistent, which makes the rules more complicated and does not allow easy coverage of all words.

Case: The only cases in Swedish are basic and genitive, and the general genitive suffix in Swedish is *-s*. As for consistency, there is yet a word which takes another suffix to indicate genitive case to be found; from this, the following conclusion was drawn: If a word takes a suffix in genitive case, the suffix will be -s.

However, there are quite a few words which do not take a suffix in genitive case. These are either words whose last character is *s*, or whose last phone produces the same sound; an example of the former is *hus*, and of the latter *lax*, whose singular indefinite forms are the same regardless of the case. Some writers' recommendations include the use of *-'s* as a genitive suffix in ambiguous cases, but in general, this is neither needed nor recommended.

3.4 Adjective morphology

Swedish adjective morphology is also fairly complex. The surface form depends on the gender and the number of the noun which it specifies, although not all adjectives

have different surface forms in all combinations of gender and number.

Number: The surface form for Swedish adjectives in positive forms is changed when the noun they specify change in number. There is a tendency for an a positive singular adjective in uter form to be concatenated with the suffix *-a*: *röd – röd-a, vit – vit-a*. The comparative forms is the same for singular and plural, regardless of number. The surface forms in attributive and predicative roles are the same in both singular and plural.

Gender: In Swedish, the neuter form of an adjective in singular positive form tends to get the suffix *-t* or *-tt*, for example *vit – vit-t, blå – blå-tt*, but this is just a tendency and not a general rule. In some cases, most notably adjectives ending with *-d* or *-n* in singular uter form, the last character is replaced with the suffix *-tt* as in *rö-d – rö-tt*. Other adjectives whose last character is *-t* in singular uter form are unchanged in neuter: *platt, stött, svart*. This phenomenon occurs not only in adjectives with *-t* as their last character, though; an example of such a word is (the colour) *orange*.

Species: The surface form of the majority of Swedish adjectives changes depending on whether the head is definite or indefinite. For uncountable nouns, the surface form is the same as with countable nouns in the singular form.

Species inflection is different depending on whether the adjective is used attributively or predicatively, and also depends on the number.

SINGULAR		
Role	**Species**	**Example**
Attributive	Indefinite	En **röd** leksak.
Attributive	Definite	Den **röda** leksaken.
Predicative	Indefinite	En leksak är **röd**.
Predicative	Definite	Leksaken är **röd**.
PLURAL		
Role	**Species**	**Example**
Attributive	Indefinite	Många **röda** leksaker.
Attributive	Definite	De här **röda** leksakerna.
Predicative	Indefinite	Många leksaker är **röda**.
Predicative	Definite	De här leksakerna är **röda**.

Table 3: Differences between attributive and predicative positive adjectives.

As shown in table 3, the surface form changes to the plural form when the adjective's role is attributive, and the species is definite.

3.4.1 Morphology in comparative and superlative adjectives

The adjective forms used are positive, comparative, and superlative. The previous paragraphs describe the inflection properties for the positive forms only, where there are agreement requirements in gender and species, as well as in number.

Role	Number	Example
Attributive	Singular	Den *rödaste* leksaken
Attributive	Plural	De *rödaste* leksakerna
Predicative	Singular	Leksaken är *rödast*.
Predicative	Plural	Leksakerna är *rödast*.

Table 4: Examples of constructions with the superlative adjective *röd*.

There are no agreement requirements in the comparative adjective form, and it is also the same in both attributive and predicative roles. Typically, there is only one surface form of any comparative adjective, which remains the same regardlesss of the context. Genitive forms of adjectives lead to ellipsis,[5] which is not handled by the Phoenix; therefore, genitive adjective words are not treated here nor implemented in rules.

The superlative adjective forms possess somewhat special agreement properties. There are two possible forms of each superlative adjective, and the second one is usually the same as the first with an extra *-a* or *-e* suffix. This last form is manifest only in attributive phrases. The examples in table 4 show that the adjective's role (attributive or predicative) in the phrase determines the surface form, and not the number properties of the head.

3.4.2 Periphrastic comparison of adjectives

An adjective does not have to be compared morphologically; they may also be compared periphrastically, using different forms of the adverb *mycket*; this method is also used for comparison of verb participles. An adjective which has a morphological comparison pattern can always be compared periphrastically, too, although this option is, in my personal experience, mostly used only for stilistic purposes.

When using periphrastic comparison, the adverb is used to reflect the compared form only. The adjective itself is inflected in number and gender only; neither species nor compared form affect its surface form. Table 5 shows examples of how periphrastic comparison is carried out in Swedish.

[5] For example: De *blindas* förening.

COMPARATIVE		
Number	**Gender**	**Example**
Singular	Uter	En **mer röd** leksak.
Singular	Neuter	Ett **mer rött** skynke.
Plural	Neuter or uter	Två **mer röda** leksaker.

SUPERLATIVE		
Number	**Gender**	**Example**
Singular	Uter	Leksaken är **mest röd**.
Singular	Neuter	Skynket är **mest rött**.
Plural	Neuter or uter	Två **mest röda** leksaker.

Table 5: Periphrastic comparison of Swedish adjectives.

3.5 Computational morphology in Phoenix and elsewhere

The earliest approaches to automatic recognition and generation of different morphological word forms were of the simple "cut-and-paste" type. This approach is very straightforward: by affixing strings at the end or the beginning of a string, the different word forms can be recognized or analyzed. For the English language, this method is often sufficient, since English morphology is rather limited. For other languages, such as Finnish and Hungarian with their rich morphology, this method is not very well suited, and for semitic languages exhibiting so-called "root and template" morphology, cutting and pasting becomes next to unusable, since there are few discernable suffixes and prefixes. Research for other methods was scarce until other languages than English were to be the subject of computational analysis and generation.

The two-level model by Koskenniemi (1983) uses a set of finite-state transducers. This model contains not only mechanisms for adding or deleting affixes, but also provides rewriting mechanisms, reflecting the relationship between orthography and phonology in the language. The rewrite rules can often be written as regular expressions, and are therefore well suited to be implemented as finite-state transducers, which can be made very space efficient – especially in morphologically rich languages – and the data structures and algorithms for working with finite-state transducers and automata are well known, fairly simple to implement, and fast.

The Phoenix system does not contain two-level or other sophisticated means for morphological analysis, nor mechanisms for rewriting sets of characters, and any implementations of morphology must therefore be made using cut-and-paste methods. However, it is possible to rewrite constructs, i.e., transform a construct containing a certain string to another construct, leaving the string unchanged, and may contain feature value changes. This mechanism makes it possible to discern between orthographically identical words and thus makes disambiguation possible.

4 Morphology rules for nouns in Phoenix

4.1 Background

Morphology rules are written in the same way as all other rules in Phoenix. The difference in suffix morphology is that a suffix is added directly after the word or stem, without blank separators. For this reason, the suffixes used have the suffix IPE test, and the roots have the prefix test. No other types of affixes have been used.

The first approach was to use a system of stems, where a suffix could change or add features to the word. Every word in the dictionary, except for the suffixes, would be a valid form of a word, i.e., no so-called technical stems as defined by Hellberg (1978). Words with irregular inflection patterns would be put into the dictionary in all their respective word forms, together with the appropriate features.

4.2 Planning

The plan was to create an assortment of nouns which could be used as prototype words for an inflection paradigm. Because there are two possibilities each in nouns for number, gender, and case, the total number of inflection possibilites is eight, at least for countable nouns. For uncountable nouns, the number of possibilites is reduced to four (excluding the plural forms).

As for the grammar, the plan was to create an integer feature and assign a number to each of the sets of inflection suffixes. Each dictionary entry should have the same feature with the same value as the set of inflection suffixes used. Properly written morphology rules would provide the means for adding the proper suffixes to the proper stems and change the features for the resulting string accordingly.

Using different morphology rules for different inflections was thought of. There is some redundancy in the suffixes, and almost all genitive inflections are formed by adding an *s* at the end of the form which has the same grammatical properties except for case. It was sawn as being easier to use the full set of inflection suffixes as a group.

4.3 Gathering prototype words

From here on, the term *prototype word* will be used to denote a word whose inflection pattern is unique compared to other prototype words, and which serves as a master example of a certain inflection pattern. Another – and more common – term is *pattern word*.

Lemma	Dicstem	Inflection	Key form	Freq
stol.nn	stol	.stol	stol, +en, +ar	6 726
film.nn	film	.film	film, +en, +er	4 931
seger.nn	seg	.seger	seg-er, -er+n, -r+ar	811

Table 6: Examples of inflection patterns.

A description and collection of Swedish nouns in Sågvall-Hein's (2000) article was used as the primary source of prototype words. This article contains inflection patterns for all the open word categories in tabular form, as the snippet in table 6 shows.

The first step was to gather all the noun prototypes and write them down in a separate file. A small Bash script was written to extract all the unique prototype nouns.

For each noun, all the inflected forms were written down and analyzed manually to determine whether their largest possible stem was a valid form of the word; these stems were saved in a new file. Words were then removed, if they met one of these criteria:

1. The stem is not a valid form of the lemma, or
2. the whole set of inflection suffixes is identical to a word already in the list.

After analysis of both of the databases, there was a list consisting of about 15 different prototype words, which was later extended to the present 20. The list may be appended to as needed, if new words are to be used which justify the addition of a new inflection paradigm.

4.3.1 Extending the list of prototypes

The goal was to use prototype words where the stem was a valid form of the word, thus avoiding non-words in the lexicon. This approach made aware of the fact that a lot of Swedish nouns use stems which are not valid word forms. Because the wish was that as many words, and corresponding inflections, as possible should be covered by morphology rules, thus avoiding a large lexicon, the original approach was changed to include prototype words with technical stems as well. Both of the corpora were reconsulted, and after analyzing them again for technical prototypes, there was another list consisting of 22 items. This list may also be appended to in the future.

4.3.2 List of prototype words

The result of the prototype word search was two lists, one for *real prototypes*, where the stem is a valid word form, and one for *technical prototypes* where the stem is not. These two sets will have to be treated differently. A real prototype can be used directly, and thus it has feature values for number, gender, and case. A technical prototype can not be used as is, and thus has no feature values except gender – all other features are contained in the suffixes. The lists of real prototypes and technical prototypes are shown in tables 7 and 8.

Number	Prototype	Number	Prototype
1	eko	11	rest
2	fax	12	ros
3	fru	13	rum
4	garage	14	sko
5	gryn	15	ton
6	hjärta	16	vis
7	huvud	17	mil
8	kam	18	leverans
9	ordförande	19	dag
10	papper	20	motor

Table 7: List of real prototypes for nouns.

Number	Prototype	Number	Prototype
1	aft-on	12	möb-el
2	ankar-e	13	nyck-el
3	bott-en	14	pojk-e
4	centr-um	15	seg-el
5	fib-er	16	seg-er
6	fing-er	17	som-mar
7	flick-a	18	vatt-en
8	förar-e	19	vim-mel
9	hum-mer	20	ög-a
10	jub-el	21	vittn-e
11	lag-er	22	läm-mel

Table 8: List of technical prototypes for nouns. The stem is separated from the suffix with a dash.

4.3.3 About the lists

The tables 7 and 8 show only the prototype words to save space. The full tables consist of the prototype word and all its inflections in different combinations of

number, case, and species, making a total of up to eight suffixes for each word. With each suffix follows a set of features, which are added to the inflected form together with the suffix.

The real prototypes (in table 7) do not always take a suffix when their features change, which constitute a problem when constructing the morphological rules. Another problem, although of less significance for the grammar development, is that the surface forms of suffixes with different features, in some cases, are identical.

The technical prototypes (in table 8) always take a suffix. The only inherent properties of the prototype stems and other nouns with the same inflection paradigm are, in these cases, gender and if the noun is an uncountable or a countable noun (in some cases they can be both).

4.4 Feature set for nouns

4.4.1 Features common to all nouns

All Swedish nouns have the following grammatical properties in all inflected and uninflected forms: species, number, gender, countability, and case. Features were created for all of these. These features are used in the noun stems as well as in the suffixes.

4.4.2 “Plural only” property

A small set of Swedish nouns have a property which states they can only be used in their plural forms, for example *kläder* (Eng. *clothes*). Because the aim was to cover as large a part as possible of Swedish noun morphology, The logical feature `allplural` was introduced to denote this property. This feature is used in noun stems only.

4.4.3 Avoiding infinite loops

As stated earlier, there are Swedish nouns where one or more of the inflected forms is the same as the lemma form in the lexicon. An example is the word `fax`, which does not take an inflection suffix in singular indefinite genitive nor plural indefinite basic form. The analyzer module in Phoenix matches as far as possible, and if the surface form has not changed after a morphological rule has been applied, this can lead to infinite loops. Therefore, a feature is needed whose value changes when a rule is applied, to make sure a morphological rule is applied only once. The result

was the logical feature `fle` (for in**fle**cted), whose value changes from 0 to 1 when a morphology rule is applied. This feature is also used in noun stems only.

Note: Looping can also be controlled by proper specification of allowable feature values. This method is chosen to enable underspecification of these feature values, which makes the rules less complex.

4.4.4 Morphosyntactical features

Different inflections of the same noun can – or must – have different sets of specifiers to form a grammatical noun phrase. A mechanism was needed to control rule reapplication; for example, a noun with a nominal genitive specifier in front of it should not take another such specifier again, as in **Pelles Kalles leksak* (Eng. "Pelle's Kalle's toy"), **en en leksak* (Eng. "*a a toy"), or **hans sina leksaker* (Eng. "*his their toys"). Another reason for introducing such a mechanism is to make sure the analysis remains correct when multiple specifiers are stacked in front of the head noun. The following integer features were created for this purpose:

- `speca`: A value of 1 means that the noun needs an **article or an indefinite pronoun** to form a grammatical noun phrase. The value 2 indicates that the noun is already specified by an article, indefinite pronoun, or measure expression.
- `specb`: A value of 1 means that the noun needs a **possessive pronoun** to form a grammatical noun phrase. The value 2 indicates that the noun is already specified by a possessive pronoun.
- `specc`: A value of 1 means that the noun needs a **genitive specifier** in the form of a pronoun or a genitive noun to form a grammatical noun phrase. The value 2 indicates that the noun is already specified by a genitive specifier.

All of the above features are used only in the noun stems. As in the case with the `fle` feature, this mechanisms enables underspecification of features. When referring to them as a group, the denotation `spec[a|b|c]` will be used.

4.4.5 Features pertaining to stemming

For the real prototypes in table 7 on page 21, an integer feature with the name `nrstem` – an abbreviation of **n**oun, **r**eal **stem** – was created. The suffixes were then added to the dictionary, together with those features which are to be added to the stem when adding the suffix. Each feature value corresponds to a certain set of inflection suffixes; those nouns which follow the same inflection pattern as one of the prototypes in table 7 are assigned a value which is the same as the prototype number. The same feature value is given to each of the inflection suffixes in the

same set, which is needed for the morphology rules to work correctly. The number of values an integer feature can have is only limited by the computer running the system, which makes the addition of new inflection paradigms rather easy. At present, there is a total of 21 values – including 0 – for the feature `nrstem`.

The corresponding feature for the technical prototypes in table 8 on page 21 is `ntstem`, which stands for **n**oun, **t**echnical **stem**. At present there are 23 values in total – including 0 – for this feature.

The full list of features used in the nouns and the suffixes are listed in table 9 together with brief explanations of which properties they denote.

Feature name	Type	Meaning
`def`	Logical	Species: definite
`indef`	Logical	Species: indefinite
`sg`	Logical	Number: singular
`pl`	Logical	Number: plural
`utr`	Logical	Gender: uter
`neu`	Logical	Gender: neuter
`count`	Logical	Count noun
`uncount`	Logical	Uncountable noun
`allplural`	Logical	Occurs as plural form only
`basic`	Logical	Case: basic
`gen`	Logical	Case: genitive
`nrstem`	Integer	"Real" stem
`ntstem`	Integer	Technical stem
`fle`	Logical	Morphology rule applied
`speca`	Integer	Needs (value 1) or has (value 2) an article or indefinite pronoun as specifier
`specb`	Integer	Needs (value 1) or has (value 2) a possessive pronoun as specifier
`specc`	Integer	Needs (value 1) or has (value 2) a genitive specifier

Table 9: List of features used in nouns and suffixes.

4.5 Assigning features to the suffixes

At this stage, when the lists of both real and technical prototypes were assembled, the time had come to list the suffixes, assign features to them, and add them as dictionary entries.

The built-in construct `finmot` (short for French *mot final*, meaning "last word") is used for suffix dictionary entries, and the IPE test used is (0,1). The internal names – the first argument in the `b_dictionary()` clause used for entries in the base

dictionary – given to the suffixes were chosen as to reflect their respective sets of features. A listing of names and feature sets is given in table 10 on page 25.

Internal name	Feature values
sgindefbas	`sg = 1, basic = 1, indef = 1`
sgdefbas	`sg = 1, basic = 1, def= 1`
sgindefgen	`sg = 1, gen = 1, indef = 1`
sgdefgen	`sg = 1, gen = 1, def = 1`
plindefbas	`pl = 1, basic = 1, indef = 1`
pldefbas	`pl = 1, basic = 1, def = 1`
plindefgen	`pl = 1, gen = 1, indef = 1`
pldefgen	`pl = 1, gen = 1, def = 1`

Table 10: Internal names and feature values for noun suffixes.

Not all forms of nouns with the same inflection paradigms as one of the real prototypes, i.e., noun stems with a value greater than 0 for the feature `nrstem`, take inflection suffixes. Therefore, only technical stems, i.e., stems with a value greater than 0 for `ntstem`, have dictionary entries for all eight of the suffixes. For example, no noun stems whose value of `nrstem` is greater than 0 have the suffix `sgindefbas`, since this is the lemma form of the word, which is used as the stem itself. Other gaps in the suffix set depend on the inflection paradigm.

4.6 Attaching stems to suffixes with rules

The XLDG code with all the rules is considered to be confidential information, which may not be disclosed in this thesis. As detailed an explanation as possible will be given to present a feel for how they work.

4.6.1 Real stems

Real noun stems are added to the dictionary as nouns, with the construct name `nomen`. The morphological rules which attach a suffix to the proper stem produce either a noun or a noun phrase construct, depending on whether the resulting form can be used as a grammatical noun phrase or not. In most cases, the features from the suffix are added to the resulting construct, and the value of `fle` is changed from 0 to 1. The following paragraphs discuss the feature checks, feature settings, and problems involved in the inflection of nouns with real stems.

Singular basic indefinite form: No suffix is attached to this form. The rule producing this form checks that the value of `nrstem` is greater than 0, and that the stem is in singular, indefinite, basic form. Also, the noun with the stem may not be a noun which is used only in plural, i.e., the feature `allplural` must have a

value of 0. Lastly, the value of `fle` must be 0, as to avoid multiple applications of morphology rules.

All the features from the dictionary entry are passed to the resulting construct. For countable nouns, the resulting construct is `nomen` (a noun). It needs a specifier to form a grammatical noun phrase, and therefore, the features `speca, specb,` and `specc` all have a value of 1. For uncountable nouns, the resulting construct should be `np` (a grammatical noun phrase), which has the same values for the `spec[a|b|c]` features. To indicate that the resulting construct has undergone a morphological transition, `fle` is set to 1. The FPE for the resulting construct is `npindef`, which translates to an indefinite noun phrase.

Singular basic definite form: All noun entries with an `nrstem` value greater than 0 take a suffix for both countable and uncountable nouns. Therefore, only one rule is needed for the production of this inflection form. This rule checks that the noun has the same features as the definite singular basic form.

The incoming constructs are `nomen` and `finmot`, which is the suffix. The rule checks that the features associated with the suffix are singular, basic, and definite, and that it has the same value for `nrstem` as the stem. In the resulting construct, the features `def, fle,` and `speca` are set to 1. The features from the stem percolate into the resulting construct. The FPE for the resulting construct is `npdef`, which translates do a definite noun phrase.

Singular genitive indefinite form: For this form, four rules were needed. To deal with nouns where this form is identical to the lemma or not, two sets of rules are needed where the input construct for the first case is `nomen`, and for the second case `nomen` and `finmot`. Since uncountable and countable nouns behave differently, there are two additional alternatives, yielding a total of four necessary rules. The output construct for countable nouns is `nomen`, and for uncountable nouns `np`. Both output constructs have `npindef` as the FPE.

Checks are performed on the `nrstem` value of the incoming `nomen` construct to make sure a suffix with the same value is added. For nouns without suffixes, the check is only performed on the value of the incoming `nomen` construct. The feature set percolates from the stem to the resulting `np` or `nomen` construct. Because there are two logical features for case, `basic` and `gen`, the value for `basic` is set to 0 in the output construct. In all cases, all of the `spec[a|b|c]` values are set to 1.

Singular genitive definite form: Because there is a suffix added at the end of the stem for all values of `nrstem` greater than 0, only one rule is needed. The constructs entering the rule are `nomen` and `finmot`, and the resulting construct is `np` with `npdef` as the FPE.

Plural basic indefinite form: Two rules are needed here: one for those nouns which take a suffix, and another for those nouns which do not. In the first case, there are two input constructs, `nomen` and `finmot`, and in the second case only

one input construct, which is `nomen`. The features from the `nomen` construct percolate to the output `np` construct, and in the cases where a suffix is added, the features from the suffix are set in the output construct as well. Because number is the only property which changes compared to the lemma form, the only suffix feature set explicitly is `pl`, which is set to 1, and the feature `sg` is at the same time set explicitly to 0. As earlier, the value of `nrstem` is checked in both cases to make sure only the proper stems and suffixes are combined. The FPE for the outgoing `np` construct is `npindef`.

Plural basic definite form: All noun stems with an `nrstem` value greater than 0 take a suffix in this form; therefore only one rule is needed. The input constructs are `nomen` and `finmot`, and the resulting construct an `np` with `npdef` as the FPE. The features from the first construct percolate to the result. Worth noting is that the only `spec[a|b|c]` feature set in the result is `speca` (which is set to 1), because a noun in this form can take only an article or an indefinite pronoun to form a grammatical noun phrase.

Plural genitive indefinite form: Some of the nouns with an `nrstem` value greater than 0 do not take a suffix compared to the lemma form, and therefore two rules are needed to handle this inflection form. The rules are very similar to the rules applying to plural basic indefinite form; the only differences are that the sets of `nrstem` values are different, and that different features are added to the resulting construct. Also, the values for the `spec[a|b|c]` features are all set to 1 in the resulting construct, which is a `np` construct with the FPE test `npindef`.

Plural genitive definite form: Because all nouns with an `nrstem` value higher than 0 have a suffix in this form, only one single rule is needed. The only difference between this rule and the rule for analyzing the stem and suffixes in the basic definite form is the set of features transferred from the suffix to the resulting construct; in all other aspects – except for the rule numbers, which have to be unique – the rules are identical.

4.6.2 Technical stems

The rules for the technical stems are considerably easier to implement, because every inflection form – including the lemma form – takes a suffix. The only features in the stems in the dictionary are gender, countability, and an `ntstem` value. If the noun belongs to the rather small group which are plural only, the logical feature `allplural` is set to a value of 1.

The names, features, and FPE tests in the resulting constructs for nouns where the value of `ntstem` is greater than 0 are the same as the corresponding rules applied to stems with `nrstem` values – the rules are used only for the analysis of different surface forms of words in the same category and the same grammatical properties.

Singular basic indefinite form: Two rules are needed here – one for the handling of countable nouns, and one for uncountable nouns. These rules are very similar – the differences lie in the name of the resulting construct (`nomen` for countable nouns and `np` for uncountable nouns). Also, the feature values set in the resulting construct are different, because the added suffixes have different properties.

Both of the rules have the same kinds of incoming constructs – `nomen` and `finmot`. The `spec[a|b|c]` values are all set to 1 in the resulting construct, which has `npindef` as the FPE test.

Singular basic definite form: This form is covered by a single rule. The constructs going into the rule are `nomen` and `finmot`, and the result is an `np` construct with the FPE test `npdef`. The value of the feature `speca` is set to 1 after the rule has been applied.

Singular genitive indefinite form: Again, two rules are needed for each of the groups uncountable and countable nouns. The difference between them is that the resulting construct is `nomen` in the rule which is applied on countable nouns and `np` in the one applied to uncountable nouns. The usual checks on stem features and `ntstem` values are performed, and the the values of the `spec[a|b|c]` features are set to 1 in the result. The FPE test is `npindef` for the resulting construct in both rules.

Singular, genitive, definite form: Only one rule is needed here, because this form is not affected by countability properties. This rule is very similar to the form with basic case, and the only difference is the set of features set in the resulting construct.

Plural basic forms (definite and indefinite): Only two rules, one for definite and one for indefinite forms, are needed here. The input constructs are in both cases a `nomen` and a `finmot` construct, which are checked for consistency in their respective `ntstem` values. The resulting construct is an `np`, with the FPE test `npindef` for the indefinite form and `npdef` for the definite form. In the basic indefinite form, the values for all of the `spec[a|b|c]` features are set to 1, while only `speca` is set to 1 in the basic definite form.

Plural genitive forms (definite and indefinite): As in the previous paragraph, one rule for definite form and one for indefinite form are needed. The differences between these and the corresponding rules for nouns with basic case are that the output in this case is a `nomen` construct, and that the genitive feature is carried into the result instead of the feature for basic case.

4.7 Nouns without morphology rules

There are a number of nouns whose inflection paradigms do not match any of the prototype words in tables 7 and 8 on page 21. The inflected forms of these words will have to be added by hand, together with all respective features.

Those paradigms were chosen which were deemed to be common enough to motivate an inflection paradigm and morphology rules. There is little meaning in creating new inflection paradigms if there are very few words in the language which follow them; the addition of new dictionary entries will take a very long time if the number of prototype words is large. The grammar developer has to go through all the inflection forms of the noun to be added and check the prototype lists to determine whether there is already an inflection paradigm for it.

The decision to include new paradigms is up to the grammar developer creating the morphology rules. If the dictionaries are created by other developers, who add words following a certain paradigm which do not have a prototype word, the person or persons developing the base grammar should be able to add paradigms on request.

5 Morphology rules for adjectives in Phoenix

5.1 Planning

The methodology used when formalizing the noun morphology seemed to be working, so it was reused when working with the adjective morphology.

5.2 Gathering prototype words

A description and collection of Swedish adjectives in Sågvall-Hein's (2000) article was used as a source for the gathering of prototype nouns. Example data in the two sources can be found in section 4.3 on page 19.

For the adjectives, the adjectives from the two databases were extracted or copied, and the list was pruned to contain only unique entries. The inflection paradigms for each of the adjectives were written down– which also indicated which adjectives could be used as technical and real prototypes, respectively – and removed those enties whose inflections patterns were either identical to another entry, or where the inflection paradigm was of such a type that it could not be used effectively as a prototype word in the morphology. By this, adjectives whose stem alters significantly when inflected or compared are referred to; for example, the comparison *dålig – sämre*, whose stem is fully altered and whose suffix morphology rules it out as a good candidate for a prototype stem.

5.3 List of prototype words

The result of the prototype word search was two lists, one with real prototypes and one with technical prototypes. These lists are represented in tables 11 and 12 on this page and page 31, respectively.

Number	Prototype
1	allmän
2	blek
3	ensam
4	fri
5	hög
6	lätt

Table 11: List of real adjective prototypes.

Number	Prototype
1	angeläg-en
2	gla-d
3	hår-d
4	när-a
5	tun-n
6	vack-er
7	äd-el

Table 12: List of technical adjective prototypes.

5.4 About the lists

The lists of adjective prototypes are considerably shorter than the corresponding list of noun prototypes. This is partly due the more regular inflection pattern for adjectives compared to nouns. The shorter lengths of the lists does not necessarily mean that the inflection paradigms cover a smaller subset of Swedish adjectives than the noun coverage of the inflection paradigms for the prototype nouns in the tables 7 and 8 As in the work done for prototype nouns, the prototype adjectives are words with inflection patterns which seem to be the most common in Swedish.

5.5 Feature set for adjectives

To correctly work with adjectives and adjective words, a number of features had to be used. A subset of the features in table 9 on page 24 were used, but because these were not enough to properly represent adjectives, new ones had to be created as well.

5.5.1 Features for comparison: Different comparison forms

For representing the comparison forms for adjectives, the following logical features were created:

- `pos:` A value of 1 means **positive** form.
- `grd:` A value of 1 means **compared** or graded form.
- `spl:` A value of 1 means **superlative** form.

5.5.2 Features for periphrastic and/or morphological comparison

Some Swedish adjectives do not exhibit morphological comparison forms. To denote those stems which can be compared periphrastically, morphologically, or both,

we introduced the integer feature `mcomp` (short for **m**orphological **comp**arison) in adjective stems. This feature can be set to the following values:

- `mcomp=0`: This means that the adjective cannot be compared morphologically.
- `mcomp=1`: This means that the adjective can only be compared morphologically. This value is rare, because virtually all adjectives can be compared periphrastically, although morphological comparison, if possible, is recommended.
- `mcomp=2`: This means that the adjective can be compared both morphologically and periphrastically.

5.5.3 Features for definiteness

For adjectives, as well as for nouns, there are two species cases: definite form and indefinite form. These have to be assigned features. The formalism allows us to reuse the corresponding features used in nouns:

- `def`: A value of 1 means **definite** form.
- `indef`: A value of 1 means **indefinite** form.

5.5.4 Features for number

As in the case for nouns, adjectives have two numbers, singular and plural. The logical features `sg` and `pl` can be reused for nouns in adjectives.

5.5.5 Special cases

A special case for Swedish is the definite form which is graphically similar to the plural form when used predicatively. This is called the *weak* form. Because this weak form is unchanged when the number of the noun changes, a special feature for this property was needed. The logical feature `weak` was therefore introduced.

A small group of Swedish adjectives can only be used predicatively. An example is the expression *värd* (Eng. “worth”), where the dots are replaced with the value stated. For these adjectives, constructions such as *leksaken är värd 10 kronor* (Eng. “The toy is worth 10 kronor”), is perfectly valid, but the attributive writing **en värd 10 kronor leksak* is not valid in Swedish.

5.5.6 Omitting genitive forms of adjectives

Adjectives are used in genitive form only in the case of ellipsis. The Phoenix analysis grammar is not equipped with a mechanism to handle ellipsis, and therefore, genitive forms of adjectives will not be used at all.

5.6 Assigning features to the suffixes

The assignment of features was done similar to the corresponding step in the work with noun morphology. This step was performed after the lists of both technical and real prototypes had been assembled.

As with nouns, the built-in construct **finmot** is used for suffix dictionary entries, and the IPE test is (0,1). The internal names in the clause `b_dictionary` used for dictionary entries were chosen as to reflect the set of features. The suffix names and their feature sets are listed in table 13.

Internal name	Feature values
adjsgposutr	`sg=1, def=1, indef=1, utr=1, pos=1`
adjsgposneu	`sg=1, def=1, indef=1, neu=1, pos=1`
adjplpos	`pl=1, def=1, indef=1, pos=1`
adjcmp	`def=1, indef=1, sg=1, pl=1, grd=1`
adjspl	`def=1, indef=1, sg=1, pl=1, spl=1`
adjspl	`spl=1, weak=1`
adjspl	`pos=1, weak=1`

Table 13: Internal names and feature values for adjective suffixes.

5.7 Attaching stems to suffixes with rules

The XLDG code with all the rules is considered to be confidential information, which may not be disclosed in this thesis. As detailed an explanation as possible will be given to present a feel for how they work.

5.7.1 Real stems

Real adjective stems are added to the dictionary as adjectives, with the construct name `adj`; the morphological rules which attach a suffix to the proper stem produce an adjective construct. In most cases, the features from the suffix are inherited by the resulting construct.

In the case for nouns, the introduction of the `fle` feature was necessary to avoid infinite looping (suffixation ad infinitum). For adjectives, this is not needed. A real

stem is written in the dictionary in its singular positive uter form, i.e., the features `sg`, `pos` and `utr` are all set to 1. When a plural, a comparative or superlative, and/or a neuter form is analyzed by the rules, one or more of these features are changed. Because they are checked for their correct values before suffixation takes place, there is no need for any extra features.

The following FPE tests are used:

- For all positive forms, the FPE test is `fpe(1)`.
- For the comparative form, the FPE test is `fpe(grad(1,str(''grd'')))` (1 for the first incoming construct; `grad` for graded; `str(''grd'')` denotes comparative form).
- Lastly, the FPE test for superlative forms is `fpe(grad(1,str(''spl'')))` (where `str(''spl'')` denotes superlative form).

For a morphological comparison to take place, the value for the feature `mcomp` may not be 0. This value is checked in the rules for comparative or superlative form with `gt(mcomp,0)`, meaning that the value for this feature has to be greater than 0.

Positive singular uter form: This is the form in which real stems are written in the lexicon, and so no rule is applied for this form to be analyzed since no suffix is added.

Positive singular neuter form: A suffix is attached to stems with `arstem` values greater than 0, except when the value of `arstem` is 6 (prototype word is *lätt*), and therefore one of two rules can be applied. The value of the feature `utr` is changed from 1 to zero, and the feature `neu` is changed from 0 to 1. All other features are inherited from the stem.

Positive plural form: For all adjective stems with an `arstem` value greater than 1, a suffix is added to form the resulting construct. The rule application changes the values `sg` and `pl` for the resulting construct to 0 and 1, respectively. All other features are inherited from the adjective stem.

Positive weak form: This form is used in singular attributive form and plural predicative form. Since the form remains unchanged when the number of the noun changes, no feature values for singular or plural has to be changed. The feature `weak` has a value of 1 in the suffix, and because this is the only feature change in the resulting construct, its features are inherited from the suffix.

Comparative form: The features changed with the application of this rule are `pos` and `grd`, whose values are set to 0 and 1, respectively, in the resulting construct. Also, the `pl = 1` feature and value are set, because this form is unchanged as the number of the noun is changed. All other features are inherited from the stem.

Superlative forms: There are two kinds of superlative forms, the weak form and

the non-weak form. The weak form is used in attributive adjectives, and the non-weak form is used predicatively. For all stems with `arstem` values greater than 0, a suffix is added, so two rules are needed to handle these forms.

In the case of the non-weak form, the values of the features `pos` and `spl` are changed to 0 and 1, respectively, in the resulting adjective construct. All other features are inherited from the adjective stem.

The same values are changed in the case of weak forms. The difference is that the suffix, which contains the value 1 for the feature `weak`, is being used for feature value inheritance.

5.7.2 Technical stems

As in the case of nouns, the rules concerning technical adjective stems are easier to implement because every inflection and lemma form take a suffix. The only feature in the dictionary entries for technical adjective stems is `atstem`, which has a value greater than 0. The features `def` and `indef` are checked in the rules; their values have to be 0 for a technical adjective stem suffix rule to be applied, which eliminates the need for a `fle` feature indicating previous rule applications and therefore prohibiting loops.

The names, features, and FPE tests in the resulting constructs for adjectives where the value of `atstem` is greater than 0 are the same as the corresponding rules applied to stems with `nrstem` values greater than 0; i.e., the FPE test for all positive forms is `fpe(1)`. The FPE test `fpe(grad(1,str(''grd'')))` is used for the comparative form. Finally, the FPE test for the superlative forms is `fpe(grad(1,str(''spl'')))`. The constructs going into the rules are `adj` and `finmot`, and the resulting construct is of the type `adj`.

Singular positive neuter form: Only one rule is needed here. The first construct – the adjective stem – has to have an `atstem` value greater than 1 and `def` and `indef` features set to 0. The features and values added to the result are `sg=1, neu=1, pos=1, def=1`, and `indef=1`.

Singular positive neuter form: Similar to the rule mentioned above. The only difference is that the feature value `utr=1` replaces `neu=1` in the list above.

Positive weak form: The suffix construct, `finmot`, contains the feature `weak` with a value of 1. This and other features are inherited from the suffix in the resulting construct.

Plural positive form: Also very similar to the singular positive forms; the feature value `pl=1` replaces `sg=1`, and none of the features `neu` or `utr` are set.

Comparative form: Similar to the rule above. The feature value `grd=1` replaces `pos=1`. Also, the value of the `mcomp` feature of the stem has to be greater than 0,

which is checked for with `gt(mcomp,0)`.

Superlative non-weak form: The only relevant feature present in the suffix is `spl`, having a value of 1. This and other features are inherited from the suffix by the resulting construct.

Superlative weak form: The suffix construct `finmot` contains the features `weak` and `spl`, both with a value of 1. These and other features are inherited from the suffix by the resulting construct.

5.8 Periphrastic inflection

In order to make the adjective inflection patterns complete, periphrastic inflection also had to be dealt with.

In the base schema (base dictionary), there exist a number of grade adverbials, for which the construct is `gradadv`. These can be use to grade adjectives. So far, the only grade adverbial in our base schema is the lemma *mycket* (Eng. "very" or "much", depending on the context), which has the feature value `much=1`. The positive, comparative, and superlative forms of this lemma are *mycket*, *mer*, and *mest*, respectively, and they are contained in the base dictionary with the feature value `much=1` and one of the feature values `pos=1`, `grd=1`, and `spl=1`, denoting positive, comparative, and superlative forms, respectively. The superlative form can be either weak or non-weak, and thus requires two rules.

The rules for periphrastic inflection requires two constructs. The first construct is a `gradadv` as mentioned above. The second construct is a positive form of an adjective with the construct name `adj`. The adjective is inflected to correspond to the noun with regards to number and gender, thus the only requirements on this construct is that it is in positive form (`pos=1`) and that it can be periphrastically compared (`mcomp=0` or `mcomp=2`). The weak superlative form also has to have the feature value `weak=1`.

In all cases, the features in the resulting `adj` construct are inherited from the second construct (the adjective).

Positive periphrastic form: The FPE test for this rule is `fpe(2)`. No other feature values except for the ones inherited from the second construct are set or changed.

Comparative and non-weak superlative periphrastic form: Here, the FPE test is `adadj(1,2)`, denoting an adadjective. Because the `pos=1` feature value is inherited – together with other features in the second construct – it has to be reset to 0. At the same time, the feature values `grd=1` and `spl=1`, respectively, are set in the comparative and non-weak superlative forms. The rule for the non-weak superlative periphrastic comparison requires that the feature `weak` has a value of 0 in the second construct.

Weak superlative periphrastic form: Everything is identical to the rule for the periphrastic non-weak comparison, except for that the feature `weak` is required to have a value of 1 in the second construct.

6 Testing

6.1 Testing preliminaries

The Phoenix system is designed to accept only whole sentences, i.e., constructs having the name `sent`. These constructs act like top-level rules. It is possible to test single words and phrases, but this means that only the log file can be used for evaluation.

The minimum requirements for a complete Swedish sentence – not only in the Phoenix formalism, but also in general – are one finite verb and a noun phrase, which can act as a subject. The main purpose of Phoenix is to analyse requests for information, either stated as a question or as a command. The decision was made to use simple command phrases consisting of the imperative verb *lista* (Eng. "list") and a noun phrase. The noun phrase may or may not contain the universal quantifier *alla* (Eng. "all"), an optional adjective, and a noun. For the testing to be comprehensive, a number of things are required:

1. A small number of grammar rules to analyze the above mentioned command sentence.
2. A set of nouns which cover all the inflection patterns mentioned in tables 7 and 8, in all forms. The set has to contain uncountable nouns and countable nouns, as well as nouns which can be used in plural form only.
3. A set of adjectives which cover all the inflection patterns mentioned in tables 11 and 12, in all forms except genitive, which are not used in the Phoenix formalism because of ellipsis issues. The set should contain adjectives which can be inflected periphrastically and/or morphologically.
4. A set of sentences containing the above.

To make certain that the grammar does not accept improper nouns, adjectives, and sentences, a set of improperly formed sentences are needed which the grammar should not be able to analyze.

6.2 Sentences for testing noun morphology

The purpose of the tests is only to test the morphological rules, not taking into account or requiring feasible meaning, and therefore the sentences may in some cases be nonsensical in nature.

For most of the inflection patterns, two dictionary entries were used. In some cases, only one token – the prototype – could be found, and in those cases only one entry was used. A total of **666 grammatically correct sentences** were used for testing noun morphology, and were distributed as follows:

- **272** sentences contained countable nouns whose inflection patterns corresponded to one of the real noun prototypes;
- **304** sentences contained countable nouns whose inflection patterns corresponded to one of the technical noun prototypes;
- **48** sentences contained uncountable nouns whose inflection patterns corresponded to one of the real noun prototypes; and
- **42** sentences contained uncountable nouns with inflection patterns corresponded to one of the technical noun prototypes.

Note: The number of sentences containing uncountable nouns is much smaller than the corresponding number of sentences containing countable nouns. There are several inflection patterns which differ only in plural (for example, the prototypes *eko* and *hjärta* in table 7 on page 21). Because of this, only one of these `nrstem` values needs to be used.

6.3 Sentences with countable nouns

The following forms and example sentences were used for the testing of noun inflection for countable nouns. In these examples, the noun is *eko* (Eng. “echo”), which is italicized here. The noun phrases, which are the focal points of the tests, are surrounded by square brackets.

1. **Singular nominative indefinite form**:
   Lista [ett *eko*] (list an echo).
2. **Singular nominative definite form**:
   Lista *ekot* (list the echo).
3. **Singular genitive indefinite form**:
   Lista [ett *ekos* egenskap] (list an echo’s property).
4. **Singular genitive definite form**:
   Lista [*ekots* eko] (list the echo’s property).
5. **Plural nominative indefinite form**:
   Lista [alla *ekon*] (list all echoes).
6. **Plural nominative definite form**:
   Lista [*ekona*] (list the echoes).
7. **Plural genitive indefinite form**:
   Lista [alla *ekons* ekon] (list all echoes’ property).
8. **Plural genitive definite form**:
   Lista [*ekonas* ekon] (list the echoes’ property).

6.3.1 Results for countable nouns

The rules used in the above sentence types were corrected until only the intended semantic readings were achieved. A number of ungrammatical sentences with either improper inflection, improper article, or a combination of both were also analysed, but none of them were ever analysed insofar as to receive a semantic interpretation.

Conclusion 1: *After necessary adjustments, all of the countable noun tokens with inflection patterns following a technical or real inflection prototype were recognized correctly in all inflection forms.*

6.4 Sentences with uncountable nouns

The following forms and example sentences were used for the testing of noun inflection for uncountable nouns. In these examples, the noun is *vatten* (Eng. "water"), which is italicized here. The noun phrases, which are the focal points here, are surrounded by square brackets.

1. **Nominative indefinite form**: Lista [*vatten*] (list water).
2. **Nominative definite form**: Lista [*vattnet*] (list the water).
3. **Genitive indefinite form**: Lista [*vattens* egenskap] (list water's property).
4. **Genitive definite form**: Lista [*vattnets* egenskap] (list the water's property).

6.4.1 Results for uncountable nouns

As with sentences containing countable nouns, the rules used in the above sentence types were corrected until only the intended semantic readings were achieved. A number of ungrammatical sentences with either improper inflection, improper article, or a combination of both were also analysed, but none of them were ever analysed insofar as to receive a semantic interpretation.

Conclusion 2: *After necessary adjustments, all of the uncountable noun tokens with inflection patterns following a technical or real inflection prototype were recognized correctly in all inflection forms.*

6.5 Sentences for testing adjective morphology

As with the noun tests, the purpose is only to test the morphological rules, not taking into account or requiring feasible meaning. For each of the inflection patterns,

one dictionary entry was used. A total of **338 grammatically correct sentences** were used for testing adjective morphology, and were distributed as follows:

- **156** sentences contained adjectives whose inflection patterns corresponded to one of the real adjective prototypes, where 78 of them contained countable nouns and the remaining 78 contained uncountable nouns, and
- **182** sentences contained adjectives whose inflection patterns corresponded to one of the technical adjective prototypes, where 91 of them contained countable nouns and the remaining 91 contained uncountable nouns.

The following types of sentences were used for testing adjective inflection in noun phrases with countable nouns. In these examples, the adjective is *allmän* (Eng. “abundant”, “common”, or “public”). The noun phrases containing the noun or name specified by the adjective are surrounded by square brackets.

1. **Positive uter singular form**:
   Lista [en *allmän* regel] (list an abundant rule).
2. **Positive neuter singular form**:
   Lista [ett *allmänt* system] (list an abundant system).
3. **Positive plural form**:
   Lista [alla *allmänna* regler] (list all abundant rules).
4. **Positive weak form**:
   Lista [den *allmänna* regeln] (list the abundant rule).
5. **Comparative form, morphological inflection**:
   Lista [en regel *allmännare* än Allemansrätten] (list a rule more common than the Right of common access[6]).
6. **Comparative form, periphrastic inflection, singular, uter**:
   Lista [en regel *mer allmän* än Allemansrätten] (list a rule more common than the Right of common access).
7. **Comparative form, periphrastic inflection, singular, neuter**:
   Lista [ett system *mer allmänt* än Allemansrätten] (list a system more common than the Right of common access).
8. **Comparative form, periphrastic inflection, plural**:
   Lista [alla regler *mer allmänna* än Allemansrätten] (list all rules more common than the Right of common access).
9. **Superlative form, morphological inflection**:
   Är [regeln] *allmännast*? (Is the rule the most common?)

[6] A Swedish law stipulating what is allowed and disallowed when hiking and travelling in nature habitats and areas.

10. **Superlative form, periphrastic inflection, singular, uter**:
    Är [Allemansrätten] *mest allmän*? (Is the Right of common access the most common?)
11. **Superlative form, periphrastic inflection, singular, neuter**:
    Är [systemet] *mest allmänt*? (Is the system the most common?)
12. **Superlative form, periphrastic inflection, plural**:
    Är [reglerna] *mest allmänna*? (Are the rules the most common?)
13. **Superlative weak form**: Lista [den *allmännaste* regeln] (list the most common rule).

Note: A sentence such as *Lista en allmännare regel än Allemansrätten* is grammatically correct, but the syntactic rules for handling this syntax was not yet developed at the time of testing; however, this does not affect the correctness of the morphology rules for adjectives.

6.5.1 Test results for adjectives

As in the case for nouns, the rules involved in the above sentence types were corrected until only the intended semantic readings were achieved. A number of ungrammatical sentences with either improper adjective inflection, improper comparison, or a combination of both were also analysed, but none of them were ever analysed insofar as to receive any semantic interpretation.

Conclusion 3: *After necessary adjustments, all of the adjective tokens with inflection patterns following a technical or real inflection prototype were recognized correctly in all combinations of inflection and comparison.*

7 Conclusions and discussion

7.1 About the test results

As shown in the preceding section, all of the intended semantic readings and no unintended readings were obtained. It is not possible to test *all* ungrammatical variations of the sentence, since Swedish just like any other natural language is infinite in nature. Phoenix does not contain a spell checker, and therefore, the sentence is analysed as typed in by the user. Parses will only be generated if the sentence can be parsed at all, and misspelled words which can not be analyzed and made to fit syntactically and morphologically will make the parser halt.

In the dictionary used for testing, there could be two instances of the same dictionary entry were present, only differing in their respective values for the features `count` and `uncount`, for example *vatten* (Eng. "water"). In the cases where the test sentence was ambiguous enough to allow different readings, both of these readings were presented. When removing one or the other, the corresponding semantic reading did not show. This proves that structurally identical words have to occur in a syntactically feasible environment in order to be analysed correctly, if at all.

7.1.1 About the method

Since the XLDG formalism does not contain more advanced, e.g. finite-state, methods for analysing morphology, I do not see a clear alternative method to treat morphological analysis than the cut-and-paste method described in this thesis. The method used is similar to the corresponding method used in the German and the English grammars. Both of these languages rely more on syntax than on morphology, and the reference material is therefore limited, but the approach is the same.

If the cut-and-paste method for morphological analysis is not applicable, or applicable only to a limited extent, the grammar writer(s) have to add each inflection form individually to the dictionary. As large a part of the language involved will be covered as when using finite-state methods at the cost of the longer development time needed by the lexicographer.

7.2 Real and technical inflection prototypes

7.2.1 Errors discovered during testing

After compiling the lists of prototypes, some errors were discovered. Most of them were due either to improperly written suffixation rules (mostly because of feature value omissions) or improper suffix strings in the dictionary. The following errors are worth special mentioning:

- The real inflection prototypes with `nrstem` values equal to 1 (eko) or 6 (hjärta) have the same inflection pattern, thus shortening this list by one. The choice was made to delete the prototype "hjärta", but either one can be deleted. This does not affect the analysis in the test sentences, nor do any rules have to be changed, as long as the rest of the list is unchanged with regards to `nrstem` values and corresponding inflection patterns.
- The technical inflection prototypes with `ntstem` values 10 (jubel) and 15 (segel) also have the same inflection pattern. This list can therefore be reduced by one entry. Because the inflection rules for technical noun stems exhibit a more consistent behaviour, the rules remain the same, and the `ntstem` values denoting a certain inflection pattern may be rearranged without changes to any other rules.
- The technical inflection prototype with the `ntstem` value of 21 (vittne) is not a technical stem. The stem is always *vittne*, and the inflection pattern corresponds to the pattern of the real stem prototype with the `nrstem` value of 1 (eko).

Because the list was only pruned with regards to – in this case – duplicate entries, the coverage of the noun morphology remains unchanged.

7.2.2 About the choice of inflection prototypes

When selecting the inflection prototypes for nouns and adjectives, no considerations were made as to the real-world frequency of their respective inflection patterns. Therefore, there may very well exist inflection prototypes which could be added to the grammar, possibly replacing present prototypes because of better coverage of Swedish morphology. A strong recommendation advice to grammar writers implementing morphology in new language is therefore to undertake corpus research and try to determine which stems are the most common, and also how common they are, thereby yielding a list of inflection prototypes with as good a coverage as possible.

7.2.3 Benefits of using technical stems

The use of technical stems as opposed to real stems makes the rules considerably easier. A suffix must be appended to a technical stem when creating a surface form, which is not always the case when working with real stems; thus, the rules have to be written so that a $\emptyset$-suffix cannot be added to a real stem, since this can potentially lead to infinite loops. The rule which handles these constructions simply transforms the construct by changing the `fle` feature value from 0 to 1, and in some cases changes the construct's name.

A lot of the rules involving real stems have two instances, one for analysing stems whose surface form does not change when the grammatical properties change, and on for analysing stems where the surface form changes. These rules take into consideration different values for `nrstem` and `arstem`. Appending the list of real prototypes with new ones means that the suffix affigation rules may also have to be changed, which is always prone to errors since there is a human involved. Renumbering the real prototypes leads to the same considerations. If the real prototypes are reordered, i.e., the `nrstem` and/or `arstem` values denote different sets of suffixes, most or all of the suffixation rules have to be changed or possibly rewritten altogether.

As for rules governing suffixation of technical stems, there is only one rule for each grammatical form because the surface form *must* be changed when grammatical properties are inherited; the only condition is that an `ntstem` or `atstem` value must be present, and its value must be equal to or greater than 1. This makes technical stems easier to handle for the developer.

The grammar developer should think twice when adding new inflection patterns. In Swedish, there are nouns which can have real stem inflection patterns, but are better off as technical stems. Example are the words *egendom* (“property”), *visdom* (“wisdom”), and *sjukdom* (“disease”), which all can be considered as real prototypes; however, all these words are compounds involving the word *-dom*, whose inflection pattern and gender is the same. Choosing a technical stem approach in these kinds of words can reduce the number of stem prototypes needed; one should not always choose the technical stem by simply stripping the last one or two characters in the singular indefinite nominative form of the word.

7.2.4 Benefits of using real stems

For lexicographers extending the dictionary to handle new words and new domains, the use of real and technical inflection prototypes can simplify the process. The lexicographer can quickly determine if a new entry has a technical or real stem, which limits the number of possibilities by about half.

7.2.5 Technical or real stems: Which should be chosen?

The answer to this question is “it depends”. From the developer’s point of view, technical stems are preferred, since the complexity and number of rules decrease considerably. From a lexicographer’s point of view, however, sifting through a large number of technical inflection prototypes when adding a word, looking for the right inflection paradigm, is more demanding than firstly slicing the number of possibilities by determining if a certain entry has a technical or real inflection prototype.

After all, the grammar and its rules should need to be written only a small number of times – ideally only once – but several dictionaries may be created when using Phoenix in new linguistic domains where there is a large number of new dictionary entries, and several lexicographers may be involved in this process. My recommendation for future developers when writing new grammars for new languages is therefore that *real and technical prototypes should be used in conjunction*, as long as the structure of the language motivates it. Of course, dictionary development tools which can automatically or semi-automatically determine the stem and stem type. An example of such a tool is the pattern finder module by Starbäck and Tiedemann (1997) in the ScanCheck language checker for the checking of controlled language. A more general, domain-independent version of this tool can render these recommendations obsolote, if it fully hides these decisions from the lexicographer. If such a tool is developed, it will be adapted to the approach chosen by the grammar developer.

7.3 Future development

As for the Swedish analysis grammar, the approach described here might be used for the analysis of verbs. Swedish verbs contain fairly regular inflection patterns, and I am confident that a system of technical and/or real verb stems with suffixes can be used for verbs as well. In other languages with a similar inflection system, the stem-and-affix approach should be feasible for word categories exhibiting some inflection.

A two-level – or other finite-state based – mechanism for analysing morphology would be very useful in this system. Such a module would give way for morphological analysis rules when working with languages which are inflected differently; my first thought goes to semitic languages, but it can be useful in other languages too. A more advanced analysis mechanism would, however, demand more knowledge from the grammar developer.

There exist two-level systems for morphology which in the hands of good programmers could be integrated in Phoenix. The conclusion whether the result is worth the time and effort required depends on the language or languages which are to be analyzed as well as on the use in the presently implemented languages. Also, the influence on the computing power needed to parse queries using a two-level system will and should be tested and considered. On top of this are also potential licensing issues. I think that the use of a finite-state based morphological analyzer would be a good idea in the development of a set of morphologically rich languages, since adding every surface form by hand is both time-consuming and error-prone.

References

Dahlbäck, N., Jönsson, A. and Ahrenberg, L. (1993). Wizard of Oz studies – why and how, Workshop on intelligent user interfaces, Orlando, FL. Available at *http://citeseer.nj.nec.com/45570.html*.

Hellberg, S. (1978). The morphology of present-day Swedish. Word-inflection, word-formation, basic dictionary.

Jönsson, A. (1990). Linköping dialogue corpus. Download available from *http://www.ida.liu.se/∼nlplab/dialogues/corpora.html*.

Koskenniemi, K. (1983). *Two-Level Morphology: a General Computational Model for Word-Form Recognition and Production*, PhD thesis, University of Helsinki.

Lehmann, H. (2001). *Phoenix Grammar Development Handbook*, Direct Dialogue AB.

Lundstedt, K. (2002). *A computational grammar for swedish noun phrases in a natural language interface to databases*, Master's thesis, Uppsala University.

Mattisson, B. (2000). *Automatisk detektering av partikelverb*, Master's thesis, Uppsala University.

Phoenix AB (2001). Master grammar specification for analysis grammars in Phoenix natural language processing, Phoenix confidential, Release 1.

Pollard, C. and Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar*, The University of Chicago Press, Chicago.

Reuter, M. (1996). *Översättning och språkriktighet*, Svensk Språktjänst AB.

Sågvall-Hein, A. (2000). Swedish Morphology in the UCP Framework, *Technical report*, Department of Linguistics, Uppsala University.

Starbäck, P. and Tiedemann, J. (1997). *Användarhandledning till ScanCheck*, Department of linguistics, Uppsala University.

Teleman, U., Hellberg, S. and Andersson, E. (1999). *Svenska Akademiens grammatik*, Norstedts Ordbok.

Thorell, O. (1977). *Svensk grammatik*, Scandinavian University Books. Second edition.

Zoeppritz, M. (1984). *Syntax for German in the User Specialty Languages System*, PhD thesis, University of Tübingen.