

# Modelling the Morphology of the Quenya Noun using XFST

## Assignment for 'Finite State Techniques in Computational Morphology'

Martin Schneider, University of Osnabrück, email: maschnei@uos.de

1st December 2003

## 1 A short Introduction to Quenya

### 1.1 Origin

Quenya and Sindarin are the two main elvish languages used throughout the works of J.R.R Tolkien. Those languages are artificial but they were designed to be historic languages, reflecting the transformation of language through the usage over thousands of years.

Quenya was the language of the first elves and is somewhat comparable to latin, in that it was rarely spoken in the more recent times and is mainly used for lithurgical purposes. Sindarin developed out of Quenya and became also common among other species, after the elves taught it to them.

Although artificial, Quenya combines influences from several real world languages, including Greek, Latin and Finnish. It is supposed to be more pure and lyric than Sindarin due to its more complex inflectional grammar.

Its phonological appearance is characterized by many open vowels, and a very limited number of consonant clusters as opposed to the Klingon language which can be considered at the opposite end of the spectrum. Quenya is written either in tengwar - a calligraphic script or in cirth which is inspired by celtic runes.

Fortunately Tolkien provided detailed advice on how to read them aloud and how they should be transliterated to latin script. (Therefore any Quenyish dictionary currently available is confined to latin script.) For the purpose of modelling the morphology of Quenya we will also use the latin script representation.

### 1.2 Sources

Quenya is a dead language. When Tolkien died, the language of the elves died with him. He did not even release an official Quenya grammar. Although there are very few original Quenyish texts, Tolkien made sure that those he published conveyed every important aspect of the grammar, so that it can be extracted when supplied with the proper translation.

Our work sticks very closely with the grammar as described in [W1]. As additional sources we used the websites and books listed in the appendix.

### 1.3 Objective of this Work

It seems as if Tolkien was mainly interested in nouns, since the grammar for verbs and adjectives is less succinct, and is still subject to a lot of debate and speculation amongst Quenya linguists.

The morphology of the noun includes ten cases (nominative, accusative, genitive, possessive, dative, locative, ablative, allative, instrumental and respective) and four different numerals (singular, plural, dual and partitive plural). Thus it seemed to provide an appropriate amount of work for the final assignment.

## 2 The Grammar

### 2.1 Function

Of course the function and usage of the different numbers and cases in Quenya doesn't play any role for building or understanding the transducer. Therefore we only explain them very shortly:

Apart from singular and plural there is also the dual referring to 'two of a kind' and the partitive plural, probably referring to 'some of a kind'. The partitive plural is also referred to as collective plural by people who believe that it should rather mean 'all of a kind'.

Quenya provides the cases known from English and other indogermanic languages: nominative, accusative, dative and genitive. Nominative usually designates the subject of the sentence where as the accusative form is used for the object or target of an action. Dative refers to the indirect object or recipient of an action whereas genitive designates the origin of an object. But there are also respective, locative, allative, ablative and instrumental cases in Quenya which can only be translated to English using appropriate prepositions.

The respective is the so called 'mystery case'. Tolkien once wrote a letter giving sample declinations of Quenya, where declinations for this case were given without labeling it. This case might identify the object that something is referring to.

Locative is used to specify the place, and possessive to designate the owner of an object. Both would be subsumed under the genitive case in English. Then there is the ablative for designation of the origin (translated as *from*) and the allative for the destination (translated as *to*, *into* or *upon*). The instrumental case finally relates to the instrument or cause of an action, and may be translated using *by*.

### 2.2 Phonological Rules

Traditionally the phonological structure of a language (like which combinations of sounds or letters can appear) is not considered a grammatical feature of a language but rather a surface structure of the words themselves.

In my opinion phonological rules which constitute a kind of microgrammar are highly useful if they can be regarded as playing a role in composition of words or stems and their endings. Quenya is a prime candidate for a microgrammar, since it is highly phonologically restricted. In Quenya words the vowels or diphthongs (ai, au, oi, eu, iu) are interspersed with consonants or consonant clusters.

Valid consonant clusters are cc, ht, hty, lc, ld, ll, lm, lp, lq, lt, lv, lw, ly, mb, mm, mn, mp, my, nc, nd, ng, ngw, nn, nq, nt, nty, nw, ny, ps, pt, cw, rc, rd, rm, rn, rq, rr, rt, rty, rs, rw, ry, sc, sq, ss, st, sty, sw, ts, tt, tw, ty, ks.

Upon composition/declination we may also observe a contraction. This means that a vowel can be removed without producing a non-Quenyish word. The phonological rules are implemented via the Quenyish test described later on.

### 2.3 Lexicon

Only a small list of stems was entered into the lexicon. These words and their English translation is listed below:

**cirya** [k'irja] – ship

**lasse** [lasse] – leaf

**nat** [n'at] – thing

**elen** ['elen] – star

**alda** ['alda] – tree

**tië** [ti'e] – path

### 2.4 Combination Rules

When combining two Quenya words in general, or when merging a stem and its ending in our special case, two effects may be observed: first there might occur a mutation or deletion. This affects either the end of the first part or the beginning of the second part, or even both.

But there is also the opposite effect: To make sure that merging doesn't lead to non-Quenyish consonant clusters a vowel will be inserted inbetween stem and ending if necessary. (usually *e* for singular and *i* for plural cases)

Although the mutations may be considered part of the individual case inflection rules, we have generalized them into a single combination rule.

### 2.5 Rules for Stemforms

For singular and plural declination the base stemform is used as is. The stemform for partitive plural is obtained by adding the ending *-li* to the base stemform. (the vowel *e* may be inserted between stemform and ending if necessary.)

The dual stemform is created by cutting of the stem after the last occurrence of *d* or *t* and attaching a final *-u* to it. If there is neither *d* nor *t* in the stem then a final *-t* is simply attached. (For example *alda* = tree becomes *aldu*.)

This rule seems somehow weird because a very significant part of the stem might be cut off. For example *tië* (path) which is transliterated to *tije* would simply become *tu*, and is not recognizable anymore. Therefore we decided that the cutoff rule should not expand beyond certain markers such as the diresis delimiter ']''. As a result the dual stemform of *tije* becomes *tijet*. rather than *tu*.

## 2.6 Rules for Singular Cases

**nominative** uninflected form.

**accusative** if the stemform ends in a vowel, the respective vowel will be lengthened.

**genitive** an *-o* is attached. (if the stem has a final *a* then *o* will replace it : *alda* + *-o* = *ald-o*)

**possessive** either *-va* or *-wa* is attached to the accusative (softening occurs after a consonant)

**dative** *-n* **locative** *-sse* **ablative** *-llo* **allative** *-nna*

**instrumental** *-nen* **respective** *-s*

## 2.7 Rules for Plural Cases

**nominative** *-r* for stems ending in vowels (except for *e*). and *-i* for stems ending in consonants. A final *e* will be replaced by *i*. There is an exception for words ending in *-ië* like *tië*. Here the plural form is created as if *e* was a normal vowel (*tiër*). (Note that the diresis does not turn *e* into a different vowel in Quenya, it just indicates that *i* and *e* should be pronounced separtely.)

**accusative** *-í*. if the stem ends in *e* then *í* will replace the final *e*. if the stem ends in *a* this results in the diphthong *ai* rather than *aí*.

**genitive** *-on* is attached to the nominative form.

**possessive** *-iva* (a final *e* in the stem is devoured).

**locative** like singular + *-n*,

**allative** *sg* + *-r*

**ablative** *sg* + either *-n* or *-r*

**dative**, **instrumental** and **respective** like singular, but an *i* is inserted between stem and ending.

## 2.8 Rules for Partitive Plural Cases

All wordforms for partitive plural are obtained by applying plural inflection rules to the 'partitivized' stemform. (except that there seems to be no ablative ppl ending in *-r*).

**locative**, **ablative** and **allative** can also be built using the singular rules.

## 2.9 Rules for Dual Cases

**nominative**, **accusative** and **genitive** apply singular inflection rules to the 'dualized' stemform

**possessive** is analogous to singular form, but the rules are applied to the nominative rather than the accusative form.

**locative** *-tse* if the stem ends in *t*. Note that the *t* of the stem is now considered part of the ending: (*ciryat* + *-tse* = *ciryatse*). This assignment is somewhat arbitrary, but it makes sense, since all inflection information (including the dualization indicator *t*) should be considered part of the ending in an inflectional grammar. Alternatively we could split the wordforms according to a *stem – number – case* scheme with two separators (*ciryat-se*).

Unfortunately this scheme cannot be applied to the remaining cases, since they wrap the case postfix around the number indicator (we refer to this as circumpostfix in our program.)

**ablative** *-l-o* ( *ciryat* + *-l-o* = *ciryatlo*) around a final *t* or *-llo* after a final *u*. (Mind that *t* and *u* are the only possible 'stemform endings' for dualized stemforms).

**allative** *-n-a* ( *ciryat* + *-n-a* = *ciryanta* ). *-nna* after final *u*.

**instrumental** *-n-en* around *t* or *-nen* after *u*.

**dative** *-n-* before *t* (*ciryat* + *-n-* = *ciryant*), or *-en* after *u*  
**respective** *-es* or *-s* respectively.

## 3 The Program

### 3.1 Program Structure

To get acquainted with XFST and the process of defining and enhancing a grammar, we decided to incrementally implement the grammar in the order it is introduced in [W1]. The enumeration scheme of the program adheres to the order in which grammatical rules are introduced in this reference.

Some details of the morphology were not mentioned explicitly, so we had to derive them from the examples given in the appendix of [W1]. Those examples, were also used as test cases to find out whether the FSA did what it was meant to do.

### 3.2 Modelling Techniques

#### 3.2.1 Four Steps of Transformation

The transducer is supposed to take a wordstem followed by markers for word class, case and number, and spit out the appropriate wordform (or vice versa). For example *ciryā+Noun+Possessive+PPl* would return the possessive form of the partitive plural of *ciryā* which is *ciryā́íva*. (long or accentuated vowels like *í* are represented as double-vowel symbol such as *ii* in our XFST-program)

Usually the transformation would happen in three steps: First the program would take the base stem, then it would add the appropriate ending and finally it might apply some phonological rules. For example in German *Gang+Noun+Dative+Pl* would be reduced to the stem *Gang*, attaching *e* for genitive and finally mutating *a* to *ä*.

In Quenya it turns out, that the stem is modified depending on the number before the case ending will be applied. Thus we introduce a fourth step: before we perform the declination we create a lexicon of modified stemforms. According to this model the partitive plural can be obtained by almost exactly the same rules as the simple plural, but by applying the rules to the modified 'partitivised' stem form.

Since the declination rules are applicable to any kind of stemform, we made use of variable unification to let the automaton select the appropriate combinations of modified stemforms and declinations at runtime. This is quite clever

in terms of code reuse and maintainability, but it blows up the resulting automaton a lot. As you can see in the visualisation, every path through the FSA that contains nonunifiable variable markers is inaccessible, and might simply be removed.

#### 3.2.2 Three Levels of the Dictionary

The final dictionary is build up in three steps:

In the first step we defined general rules for every possible declination (like *NounAccSgLex*, *NounDatSgLex* ...). Those were then applied to all stemforms, including the modified ones, and finally collected into lexicons according to their numbers. (*NOUNSTEMSG* for singular, *NOUNSTEMPL* for plural etc.).

At this point variable unification is used to filter out invalid combinations like singular declination of a dual-wordstem. The combination rules that control what happens upon combination of wordstem and ending, can be regarded as being dependent on the numerus only. Therefore the corresponding rules are applied to each of the four lexicons individually.

Finally *NOUNLEX* is obtained by forming the union of those four lexicons, deleting all markers and removing any leftover ambiguity by applying the Quenyish test.

#### 3.2.3 The Quenyish Test

This test is a final bottleneck to make sure that only words are generated that have a Quenyish surface structure (alas words that sound genuinely Quenyish). This kind of final surface test is very useful: since it applies to general properties of the language, it allows for a reduction of special cases, that would otherwise be considered part of local grammatical rules.

Also it is often the case that grammatical rules are only loosely defined and leave room for ambiguity which can only be removed by the final surface structure test.

For example according to [W1] in Quenya a single *i* or *e* may be inserted between stem and ending 'if necessary'

This necessity is defined by the need to be valid Quenyish.

## 4 Testing

### 4.1 The test file

```
# =====
# CONTENT:      test file for Quenya
#
# DESCRIPTION:  This file contains test cases for some
#               nouns and their inclinations
#               They are given in the format
#               (stems + markers = wordforms)
#               See the appendix of [1]
#
# SOURCES:      [W1] http://www.uib.no/People/hnohf/
#               quenya.htm
#               [L1] Pesch Helmut W.
#               "Elbisch - Grammatik, Schrift und
#               Woerterbuch der Elben-Sprache
#               von J.R.R: Tolkien"
#               2003, Bastei Luebbe Taschenbuch
#
#               All declination examples except for ti|e
#               have been taken from [W1]'s Appendix
#               The declination for ti|e has been adapted
#               according to that of li|e as given in [L1]
#
# AUTHOR:       Martin Schneider
# DATE:         01-Okt-2003
#
# PURPOSE:      this file is used to test the
#               performance of quenya.fst
#
```

# cirya (ship)

```
cirya+Noun+Nominative+Sg = cirya
cirya+Noun+Accusative+Sg = ciryaa
cirya+Noun+Dative+Sg     = ciryan
cirya+Noun+Genitive+Sg   = ciryo
cirya+Noun+Possessive+Sg = ciryaava
cirya+Noun+Locative+Sg   = ciryasse
cirya+Noun+Allative+Sg   = ciryanna
cirya+Noun+Ablative+Sg   = ciryallo
cirya+Noun+Instrumental+Sg = ciryanen
cirya+Noun+Respective+Sg = ciryas

cirya+Noun+Nominative+Pl = ciryar
cirya+Noun+Accusative+Pl = ciryai
cirya+Noun+Dative+Pl     = ciryain
cirya+Noun+Genitive+Pl   = ciryaron
cirya+Noun+Possessive+Pl = ciryaiva
cirya+Noun+Locative+Pl   = ciryassen
cirya+Noun+Allative+Pl   = ciryannar
```

```
cirya+Noun+Ablative+Pl = ciryallon
cirya+Noun+Ablative+Pl = ciryallor
cirya+Noun+Instrumental+Pl = ciryainen
cirya+Noun+Respective+Pl = ciryais

cirya+Noun+Nominative+Ppl = ciryalii
cirya+Noun+Accusative+Ppl = ciryalii
cirya+Noun+Dative+Ppl     = ciryalin
cirya+Noun+Genitive+Ppl   = ciryalion
cirya+Noun+Possessive+Ppl = ciryaliiva
cirya+Noun+Locative+Ppl   = ciryalisse
cirya+Noun+Locative+Ppl   = ciryalissen
cirya+Noun+Allative+Ppl   = ciryalinna
cirya+Noun+Allative+Ppl   = ciryalinnar
cirya+Noun+Ablative+Ppl   = ciryalillo
cirya+Noun+Instrumental+Ppl = ciryaliinen
cirya+Noun+Respective+Ppl = ciryalis

cirya+Noun+Nominative+DI = ciryat
cirya+Noun+Accusative+DI = ciryat
cirya+Noun+Dative+DI     = ciryant
cirya+Noun+Genitive+DI   = ciryato
cirya+Noun+Possessive+DI = ciryatwa
cirya+Noun+Locative+DI   = ciryatse
cirya+Noun+Allative+DI   = ciryanta
cirya+Noun+Ablative+DI   = ciryalto
cirya+Noun+Instrumental+DI = ciryanten
cirya+Noun+Respective+DI = ciryates
```

# lasse (leaf)

```
lasse+Noun+Nominative+Sg = lasse
lasse+Noun+Accusative+Sg = lassee
lasse+Noun+Dative+Sg     = lassen
lasse+Noun+Genitive+Sg   = lasseo
lasse+Noun+Possessive+Sg = lasseeva
lasse+Noun+Locative+Sg   = lassesse
lasse+Noun+Allative+Sg   = lassenna
lasse+Noun+Ablative+Sg   = lassello
lasse+Noun+Instrumental+Sg = lassenen
lasse+Noun+Respective+Sg = lasses

lasse+Noun+Nominative+Pl = lassi
lasse+Noun+Accusative+Pl = lassii
lasse+Noun+Dative+Pl     = lassin
lasse+Noun+Genitive+Pl   = lassion
lasse+Noun+Possessive+Pl = lassiva
lasse+Noun+Locative+Pl   = lassessen
lasse+Noun+Allative+Pl   = lassennar
lasse+Noun+Ablative+Pl   = lassellon
lasse+Noun+Ablative+Pl   = lassellor
```

lasse+Noun+Instrumental+Pl = lassinen  
 lasse+Noun+Respective+Pl = lassis  
  
 lasse+Noun+Nominative+Ppl = lasselii  
 lasse+Noun+Accusative+Ppl = lasselii  
 lasse+Noun+Dative+Ppl = lasselin  
 lasse+Noun+Genitive+Ppl = lasselion  
 lasse+Noun+Possessive+Ppl = lasseliiva  
 lasse+Noun+Locative+Ppl = lasselisse  
 lasse+Noun+Locative+Ppl = lasselissen  
 lasse+Noun>Allative+Ppl = lasselinna  
 lasse+Noun>Allative+Ppl = lasselinnar  
 lasse+Noun+Ablative+Ppl = lasselillo  
 lasse+Noun+Ablative+Ppl = lassellion  
 lasse+Noun+Instrumental+Ppl = lasseliinen  
 lasse+Noun+Respective+Ppl = lasselis  
  
 lasse+Noun+Nominative+Dl = lasset  
 lasse+Noun+Accusative+Dl = lasset  
 lasse+Noun+Dative+Dl = lasset  
 lasse+Noun+Genitive+Dl = lasseto  
 lasse+Noun+Possessive+Dl = lassetwa  
 lasse+Noun+Locative+Dl = lassetse  
 lasse+Noun>Allative+Dl = lassetta  
 lasse+Noun+Ablative+Dl = lassetto  
 lasse+Noun+Instrumental+Dl = lassetten  
 lasse+Noun+Respective+Dl = lassetes

#### # nat (thing)

nat+Noun+Nominative+Sg = nat  
 nat+Noun+Accusative+Sg = nat  
 nat+Noun+Dative+Sg = naten  
 nat+Noun+Genitive+Sg = nato  
 nat+Noun+Possessive+Sg = natwa  
 nat+Noun+Locative+Sg = natesse  
 nat+Noun>Allative+Sg = natenna  
 nat+Noun+Ablative+Sg = natello  
 nat+Noun+Instrumental+Sg = natenen  
 nat+Noun+Respective+Sg = nates

nat+Noun+Nominative+Pl = nati  
 nat+Noun+Accusative+Pl = natii  
 nat+Noun+Dative+Pl = natin  
 nat+Noun+Genitive+Pl = nation  
 nat+Noun+Possessive+Pl = nativa  
 nat+Noun+Locative+Pl = natissen  
 nat+Noun>Allative+Pl = natinnar  
 nat+Noun+Ablative+Pl = natillon  
 nat+Noun+Ablative+Pl = natillor  
 nat+Noun+Instrumental+Pl = natinen  
 nat+Noun+Respective+Pl = natis

nat+Noun+Nominative+Ppl = natelii  
 nat+Noun+Accusative+Ppl = natelii  
 nat+Noun+Dative+Ppl = natelin  
 nat+Noun+Genitive+Ppl = natelion  
 nat+Noun+Possessive+Ppl = nateliiva  
 nat+Noun+Locative+Ppl = natelisse  
 nat+Noun+Locative+Ppl = natelissen  
 nat+Noun>Allative+Ppl = natelinna  
 nat+Noun>Allative+Ppl = natelinnar  
 nat+Noun+Ablative+Ppl = natelillo  
 nat+Noun+Ablative+Ppl = nateliillon  
 nat+Noun+Instrumental+Ppl = nateliinen  
 nat+Noun+Respective+Ppl = natelis

nat+Noun+Nominative+Dl = natu  
 nat+Noun+Accusative+Dl = natu  
 nat+Noun+Dative+Dl = naten  
 nat+Noun+Genitive+Dl = natuo  
 nat+Noun+Possessive+Dl = natuva  
 nat+Noun+Locative+Dl = natusse  
 nat+Noun>Allative+Dl = natunna  
 nat+Noun+Ablative+Dl = natullo  
 nat+Noun+Instrumental+Dl = natunen  
 nat+Noun+Respective+Dl = natus

#### # elen (star) example for dual

elen+Noun+Nominative+Dl = elenet  
 elen+Noun+Accusative+Dl = elenet  
 elen+Noun+Dative+Dl = elenent  
 elen+Noun+Genitive+Dl = elento  
 elen+Noun+Possessive+Dl = elenetwa  
 elen+Noun+Locative+Dl = elenetse  
 elen+Noun>Allative+Dl = elenenta  
 elen+Noun+Ablative+Dl = elenelto  
 elen+Noun+Instrumental+Dl = elenenten  
 elen+Noun+Respective+Dl = elentes

#### # alda (tree)

alda+Noun+Nominative+Dl = aldu  
 alda+Noun+Accusative+Dl = alduu  
 alda+Noun+Dative+Dl = alduen  
 alda+Noun+Genitive+Dl = alduo  
 alda+Noun+Possessive+Dl = alduva  
 alda+Noun+Locative+Dl = alduisse  
 alda+Noun>Allative+Dl = alduinna  
 alda+Noun+Ablative+Dl = alduillo  
 alda+Noun+Instrumental+Dl = alduunen  
 alda+Noun+Respective+Dl = alduis

# ti|e (path)

ti|e+Noun+Nominative+Sg = ti|e  
ti|e+Noun+Accusative+Sg = ti|ee  
ti|e+Noun+Dative+Sg = ti|en  
ti|e+Noun+Genitive+Sg = ti|eeo  
ti|e+Noun+Possessive+Sg = ti|eva  
ti|e+Noun+Locative+Sg = ti|esse  
ti|e+Noun+Allative+Sg = ti|enna  
ti|e+Noun+Ablative+Sg = ti|ello  
ti|e+Noun+Instrumental+Sg = ti|enen  
ti|e+Noun+Respective+Sg = ti|es

ti|e+Noun+Nominative+Pl = ti|er  
ti|e+Noun+Accusative+Pl = tii  
ti|e+Noun+Dative+Pl = tiin  
ti|e+Noun+Genitive+Pl = ti|eron  
ti|e+Noun+Possessive+Pl = tiiva  
ti|e+Noun+Locative+Pl = ti|essen  
ti|e+Noun+Allative+Pl = ti|ennar  
ti|e+Noun+Ablative+Pl = ti|ellon  
ti|e+Noun+Ablative+Pl = ti|ellor  
ti|e+Noun+Instrumental+Pl = tiinen  
ti|e+Noun+Respective+Pl = tiis

ti|e+Noun+Nominative+Ppl = ti|elii  
ti|e+Noun+Accusative+Ppl = ti|elii  
ti|e+Noun+Dative+Ppl = ti|elin  
ti|e+Noun+Genitive+Ppl = ti|elion  
ti|e+Noun+Possessive+Ppl = ti|eliiva  
ti|e+Noun+Locative+Ppl = ti|elisse  
ti|e+Noun+Locative+Ppl = ti|elissen  
ti|e+Noun+Allative+Ppl = ti|elinna  
ti|e+Noun+Allative+Ppl = ti|elinnar  
ti|e+Noun+Ablative+Ppl = ti|elillo  
ti|e+Noun+Ablative+Ppl = ti|elillon  
ti|e+Noun+Instrumental+Ppl = ti|eliinen  
ti|e+Noun+Respective+Ppl = ti|elis

ti|e+Noun+Nominative+Dl = ti|et  
ti|e+Noun+Accusative+Dl = ti|et  
ti|e+Noun+Dative+Dl = ti|ent  
ti|e+Noun+Genitive+Dl = ti|eto  
ti|e+Noun+Possessive+Dl = ti|etwa  
ti|e+Noun+Locative+Dl = ti|etse  
ti|e+Noun+Allative+Dl = ti|enta  
ti|e+Noun+Ablative+Dl = ti|elto  
ti|e+Noun+Instrumental+Dl = ti|enten  
ti|e+Noun+Respective+Dl = ti|est

## 4.2 Wordform Generation

Here is a sample run, showing how the test file can be used to check the performance of the automaton for wordform generation (downwards application).

The functions for parsing the test file are contained in *quenia.fst* (1.a), which can be found in section 4. In our case we simply create a new FSA that extracts the lefthand side of the equations, using *downtest* (1.b). Then we feed it with the testfile (1.c)

1.a → **xfst[0]: source quenia/quenia.fst**

Opening file quenia/quenia.fst...

(...)

Closing file quenia/quenia.fst...

1.b → **xfst[0]: downtest**

13.7 Kb. 354 states, 626 arcs, 690 paths.

(...)

35.3 Kb. 421 states, 2332 arcs, Circular.

1.c → **xfst[1]: down < quenia/test.txt**

Opening file quenia/test.txt...

(...)

# cirya (ship)

2.a → **cirya+Noun+Nominative+Sg** = cirya

2.b → **cirya**

cirya+Noun+Accusative+Sg = ciryaa

ciryaa

cirya+Noun+Dative+Sg = ciryan

ciryan

(...)

cirya+Noun+Genitive+Ppl = ciryalion

ciryalion

cirya+Noun+Possessive+Ppl = ciryaliiva

ciryaliiva

**cirya+Noun+Locative+Ppl** = ciryalisse

3.a → **ciryalisse**

3.b → **ciryalissen**

**cirya+Noun+Locative+Ppl** = ciryalissen

3.a → **ciryalisse**

3.b → **ciryalissen**

cirya+Noun+Allative+Ppl = ciryalinna

ciryalinna

ciryalinna

(...)

nat+Noun+Respective+Ppl = natelis

natelis

nat+Noun+Nominative+Dl = natu

natu

**nat+Noun+Accusative+Dl** = natu

4. → **natuu**

nat+Noun+Dative+Dl = naten

natun

(...)

```

# alda (tree)

alda+Noun+Nominative+Dl    = aldu
alda
alda+Noun+Accusative+Dl    = alduu
alduu
alda+Noun+Dative+Dl      = alduen
5. → aldun
alda+Noun+Genitive+Dl      = alduo
alduo
(...)
ti|e+Noun+Instrumental+Dl  = ti|enten
ti|enten
ti|e+Noun+Respective+Dl    = ti|est
ti|etes
Closing file quenya/test.txt...

```

Xfst prints input lines (2.a) and output lines (2.b) in alternation. If the output is equal to the righthand side of the input equation, the FSA has worked as expected. In some cases there may be multiple declinations for a word. For example the partitive plural of *ciryā* can be both *ciryālisse* and *ciryāliszen*. (3.a and b.)

As we can see in (4) the automaton considers *natuu* the correct accusative form rather than *natu*. It turned out that our automaton worked correctly according to the grammar as described in [W1] and [L1]. Evidently there was a typographical error in the appendix containing the sample declinations given in [W1].

The same is true for *alduen* (5), which isn't even a valid quenyish word since it contains the two vowels *u* and *e* in a row (double vowels are only allowed if they constitute a diphthong or if a diarhesis is used → see quenyish test).

### 4.3 Wordform Parsing

Here is another sample run, demonstrating the performance for wordform parsing (upwards application). This time the righthand sides of the equations are extracted.

```

1.a → xfst[0]: source quenya/quenya.fst
      Opening file quenya/quenya.fst...
      defined VAR: 364 bytes. 2 states, 3 arcs, 3 paths.
      (...)
      Closing file quenya/quenya.fst...
1.b → xfst[0]: uptest
      13.7 Kb. 354 states, 626 arcs, 690 paths.
      (...)
      16.8 Kb. 387 states, 837 arcs, Circular.
1.c → xfst[1]: down < quenya/test.txt
      Opening file quenya/test.txt...
      (...)

```

```

# ciryā (ship)

2.a → ciryā+Noun+Nominative+Sg    = ciryā
2.b → ciryā+Noun+Nominative+Sg
      ciryā+Noun+Dative+Sg         = ciryān
      ciryā+Noun+Dative+Sg         = ciryān
      (...)
      ciryā+Noun+Respective+Ppl    = ciryālis
      ciryā+Noun+Respective+Ppl    = ciryālis
      ciryā+Noun+Nominative+Dl     = ciryāt
3.a → ciryā+Noun+Accusative+Dl
3.a → ciryā+Noun+Nominative+Dl
      ciryā+Noun+Accusative+Dl     = ciryāt
      ciryā+Noun+Nominative+Dl     = ciryāt
3.a → ciryā+Noun+Accusative+Dl
3.b → ciryā+Noun+Nominative+Dl
      ciryā+Noun+Dative+Dl         = ciryānt
      ciryā+Noun+Dative+Dl         = ciryānt
      (...)
      nat+Noun+Nominative+Dl        = natu
4. → nat+Noun+Nominative+Dl
      nat+Noun+Accusative+Dl        = natu
4. → nat+Noun+Nominative+Dl
      nat+Noun+Dative+Dl            = naten
      nat+Noun+Dative+Sg            = naten
      (...)
# alda (tree)
alda+Noun+Nominative+Dl    = aldu
alda+Noun+Nominative+Dl    = aldu
alda+Noun+Accusative+Dl    = alduu
alda+Noun+Accusative+Dl    = alduu
alda+Noun+Dative+Dl        = alduen
5. →
alda+Noun+Genitive+Dl      = alduo
alda+Noun+Genitive+Dl      = alduo
(...)
ti|e+Noun+Instrumental+Dl  = ti|enten
ti|e+Noun+Instrumental+Dl  = ti|enten
ti|e+Noun+Respective+Dl    = ti|est
Closing file quenya/test.txt...

```

Again input and output lines are alternating (2.a and 2.b). There may be multiple interpretations of a single wordform. For example *ciryāt* could be either nominative or accusative form (3.a and b).

As we can see in (4) *natu* is only considered nominative by the automaton, rather than accusative. As explained above this is due to an error in the test data. Since *alduen* is not a valid wordform the automaton returns nothing. (5)

Evidently our quenyish test works just perfect with the given set of wordstems. It is not too strict, since it doesn't suppress any valid wordforms nor is it too relaxed allowing for invalid wordforms.

## 5 Visualisation

Unfortunately the XFST tool doesn't offer any possibility to visualize the resulting FSA graphically. More advanced tools which are freely available (like the ones offered by AT&T) provide this facility, using the graphviz package.

In order to visualize the XFST automata we exported them as prolog program (*quneyish.pl* and *nounlex.pl*). The next step was to write the program *xfst2dot.pl* in SWI-prolog, which would read such files into the prolog database and export the corresponding graph in a format that can then be rendered using AT&T's graph layout program dot.

Our program allows to assign different layouts to the nodes which correspond to the initial, intermediate and final states of the FSA. In the graphs of this section the start node is

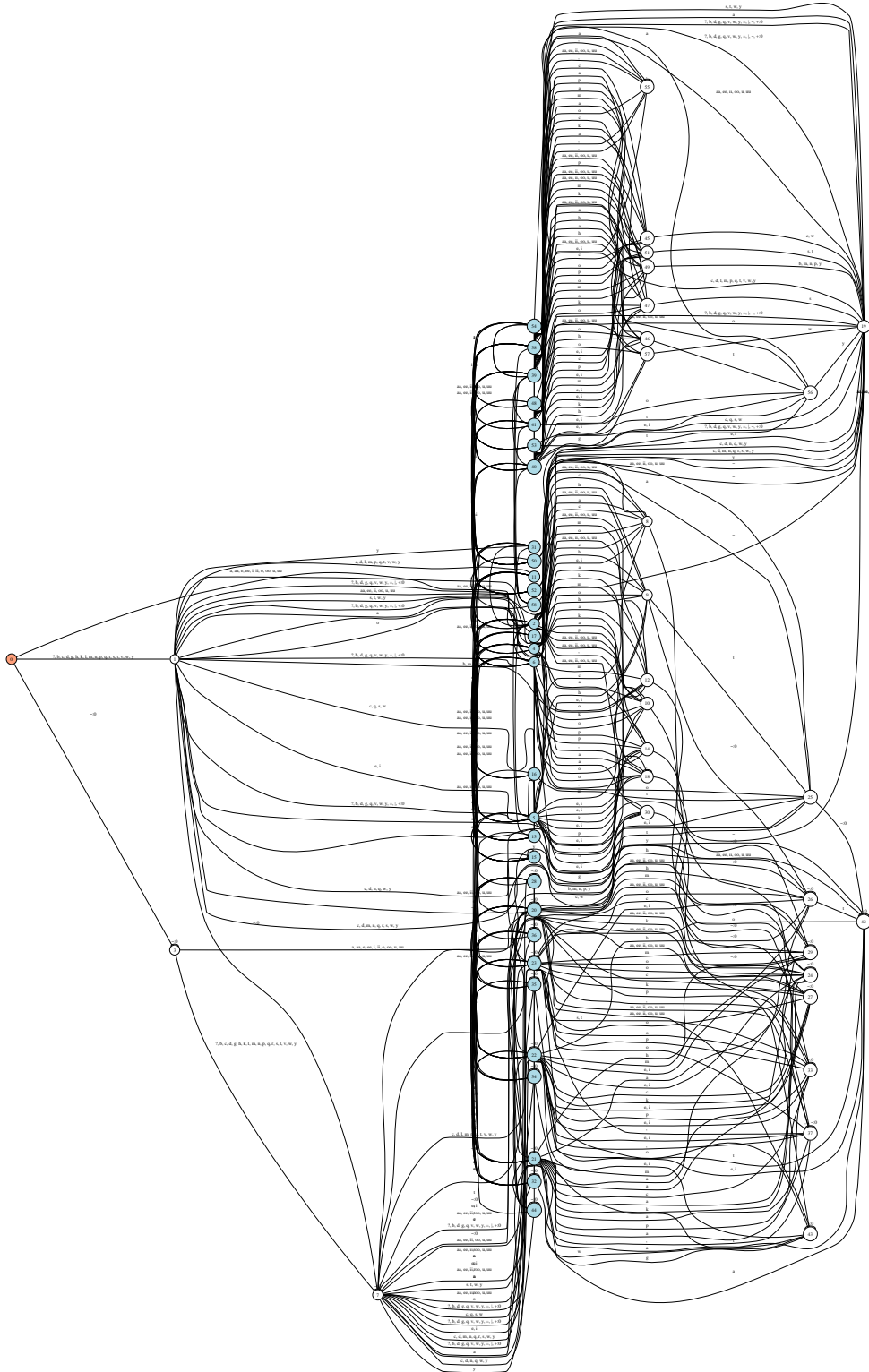
shown in salmon, whereas final nodes are all blue. We also provided flags to control whether initial and final states should be organized in different levels of hierarchy.

Since the resulting graphs were really huge and contained many parallel edges, we added a facility to our program, that would merge multiple edges into single ones (So that all transitions between any two states of the FSA would be assigned a single edge).

The documentation you are currently reading was typeset in TeX with the FST graphs included in postscript format. Thus you should be able to zoom into the pictures, to analyze the automata in more detail.

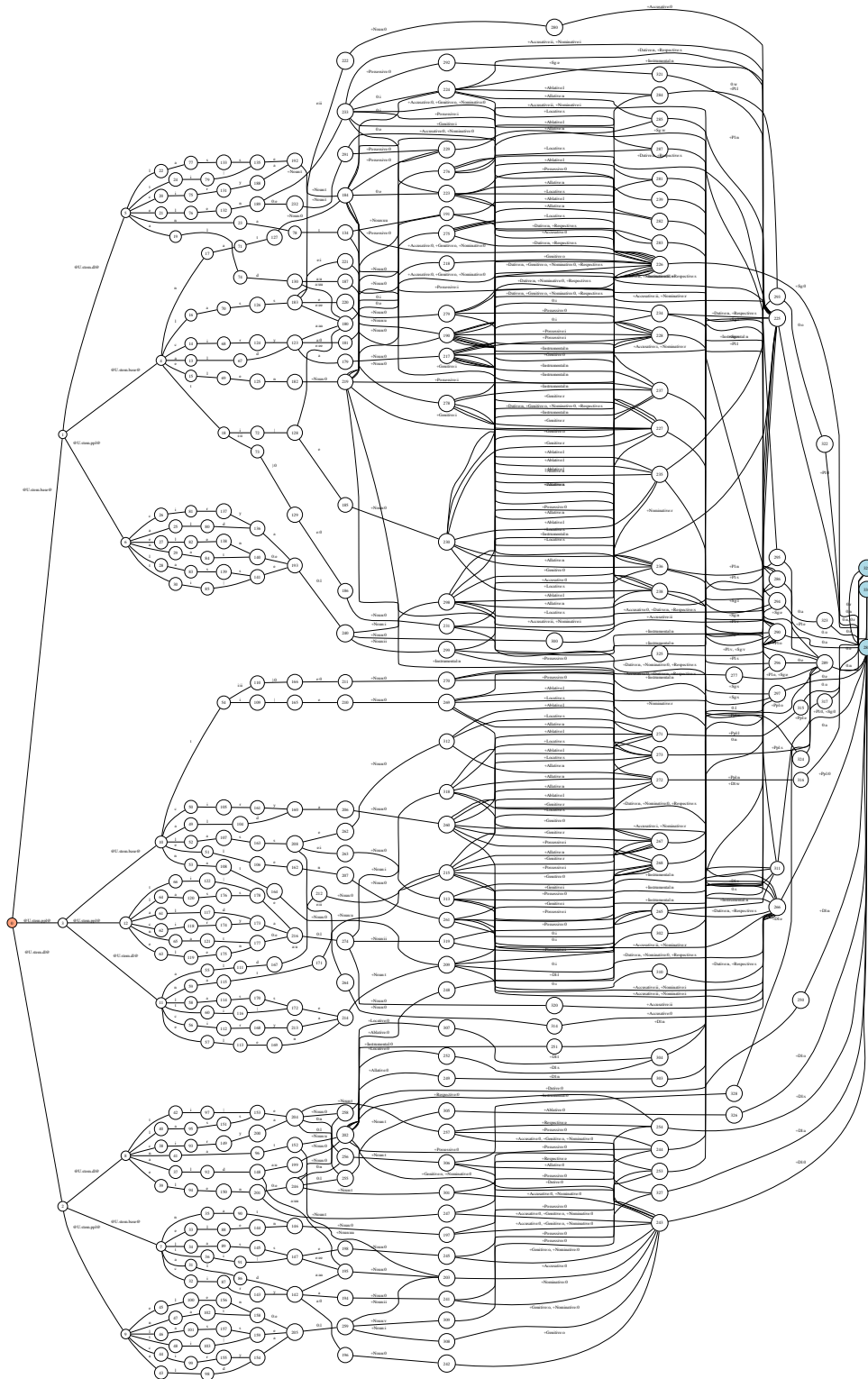
## 5.1 The Quenyish FSA

This Automaton can check if an arbitrary word could appear in Quenya language. It can also be used to generate words that have a Quenyish surface appearance.



## 5.2 The NounLex FSA

This transducer comprises the noun lexicon as defined in *quenya.fst*. It can be used to analyze or generate the corresponding wordforms



## 6 Programs

These are the two pieces of code we wrote for our assignment: the description of the finite state automaton in XFST syntax (*quenya.fst*), and a prolog program to convert the automaton into a format that can be visualized using the graphviz package (*xfst2dot.pl*).

### 6.1 quenya.fst

```
#=====#
#                               #
#       quenya.fst              #
#                               #
#=====#
#
# CONTENT:      xfst script for final
#               project
#               Quenya - the ancient
#               tongue
#               Morphology of the Noun
#               in Quenya
#
# DESCRIPTION:  Descended from
#               Primitive Elvish,
#               Quenya is the chief
#               of J.R.R. Tolkien's
#               Elvish languages,
#               used mainly in The
#               Lord of The Rings and
#               The Silmarillion.
#               The quenya noun has 10
#               cases and 4 numbers.
#
# SOURCES:      [1] http://www.uib.no/People/
#               hnohf/quenya.htm
#               [2] http://home.ix.netcom.com/
#               heensle/lang/elvish/elvish.html
#               [3] http://www.langmaker.com/
#
# AUTHOR:       Martin Schneider
# DATE:         01-Oct-2003
#
#=====#
# GENERAL REMARK
#=====#
# the order of the definitions adheres
# strictly to the order
# in which grammatical rules are
# introduced in [1].
```

```
# It might be more elegant to define
# all cases and numerals in
# a single effort, but I decided to
# increment the grammar
# in small steps.
```

```
#=====#
# (0) THE ALPHABET
#=====#
# quenya can be transcribed using the
# latin alphabet.
#
# variables
define VAR
    ["@U.stem.base@" |
     "@U.stem.ppl@" |
     "@U.stem.dl@"];
#
# delimiters to separate phonological
# units
define NEWSTOP
    [0:%+];
define STOP
    %+;
#
# delimiter to separate stem and ending
define BRK
    %?;
#
# diaeresis delimiter
#
# if two vowels occur in sequence (and
# they are not considered diphthongs)
# tolkien uses an umlaut to emphasize
# that they are spoken separately.
# we will use a DIA delimiter (% | )
# example: e | arendil or ti | e
#
# (this delimiter could be optionalized
# so that the quenyan test
# introduced considers gaps only
# if needed)
#
# define GAP
#     %-;
# define % |
```

```

# [GAP | 0];
# define DIA
# % | ;

define DIA
% | ;
define EQUAL
%=;

# valid Markers

define MARKER
[%+Noun | %+Nominative |
%+Accusative | %+Dative |
%+Genitive | %+Possessive |
%+Locative | %+Allative |
%+Ablative | %+Instrumental |
%+Respective | %+Sg | %+Pl |
%+DI | %+Ppl];

# delimiter to separate words

define SPACE
[" " | "\t"];
define DELIMITER
[STOP | BRK | DIA | SPACE |
EQUAL];

# VOWELS

define VOWEL
[a | e | i | o | u];

# long VOWELS are usually marked with
# an accent
# but we will represent them in the
# following way

define LONGVOWEL
[aa | ee | ii | oo | uu];

define VOWELISH
[VOWEL | LONGVOWEL];

# consonants

define CONSONANT
[VOWELISH | DELIMITER | VAR];

define ALPHA
[VOWELISH | CONSONANT];

#=====
# (1) ELEMENTARY PHONOLOGY
#=====

# common dipthongs in quenya

define DIPHTHONG
[{{ai} | {au} | {oi} | {eu}
| {iu}}];

# quenya is a very poetic language,
# made up mostly of VOWELS.
# inbetween VOWELS a limited number of
# consonant clusters may occur.
# the most common ones are given below

define CONSONANTCLUSTER
[{{cc} | {ht}(y) | {lc} | {ld}
| {ll} | {lm} | {lp} | {lq}
| {lt} | {lv} | {lw} | {ly}
| {mb} | {mm} | {mn} | {mp}
| {my} | {nc} | {nd} | {ng}(w)
| {nn} | {nq} | {nt}(y) | {nw}
| {ny} | {ps} | {pt} | {cw}
| {rc} | {rd} | {rm} | {rn}
| {rq} | {rr} | {rt}(y) | {rs}
| {rw} | {ry} | {sc} | {sq}
| {ss} | {st}(y) | {sw} | {ts}
| {tt} | {tw} | {ty} | {ks}}];

# now we can define an
# automaton that checks
# whether some word is 'quenyaish'.
# if some grammatical rule gives rise
# to ambiguity
# the quenyaish-check can be used to
# chose the correct form!

# delimiters are also considert
# consonantparts
# to allow for vowel-vowel combinations
# that only arise from compounds!
# for example the genitive form of
# lasse = lasse | o

define VPART
[VOWEL | LONGVOWEL |
DIPHTHONG];
define CPART
[CONSONANT | CONSONANTCLUSTER
| DELIMITER];

```

```
# each word must contain at least
# one vowel.
```

```
define QUENSINGLE
    NEWSTOP VOWELISH;
```

```
# We assume that consonant clusters can
# only appear in the middle of a word
# (so words begin either with
# a vowel/diphthong or a single
# consonant)
# All words either end with an open
# vowel or with a *smile*
# corresponding to one of the central
# consonants. (l,n,r,s,t)
# The sole known case of a consonant
# cluster in the end is {nt}
# the parts will be preceded by a
# NEWSTOP-separator
# Variables at the beginning of a word
# are simply ignored.
```

```
define CENTRALCONSONANT
    [l | n | r | s | t];
```

```
define QUENSTART
    NEWSTOP [VOWELISH | CONSONANT];
```

```
define QUENEND
    NEWSTOP [VPART |
    CENTRALCONSONANT | {nt}];
```

```
define QUENMIDDLE
    [ NEWSTOP [CPART | VPART] ]*;
```

```
define QUENSTRUCTURE
    [QUENSINGLE | [QUENSTART
    QUENMIDDLE QUENEND]];
```

```
# a quenyish word consists of
# an alternating sequence of VOWEL and
# consonant parts.
```

```
# the STOP separators are left as is
# while checking
# for alternation.
```

```
define ALTERNATE
    (STOP CPART) [STOP VPART STOP
    CPART]* (STOP VPART);
```

```
# sometimes we may want to split up a
# consonantcluster or join consonants
# to a cluster
```

```
# (stop-delimiters are inserted before
# each consonant / cluster)
```

```
define INSERTION
    STOP CONSONANT [NEWSTOP 0:VPART
    NEWSTOP CONSONANT ]*;
```

```
read regex 0:"^ [" 0:" {" STOP
    CONSONANTCLUSTER 0:"}" 0:{"
    .o. INSERTION} 0:"^ }";
    compile-replace lower;
```

```
define EXPANDCLUSTER;
    define CONTRACTCLUSTERS
    EXPANDCLUSTER.i;
```

```
# often shorter words (words with
# less alterations)
# are preferred over longer ones. (i.e
# natwa instead of natewa)
# this is achieved by contracting
# consonants which are
# interspersed with VOWELs into
# consonantclusters
# unfortunately we cannot use the
# definition of CONSONANTCLUSTER and
# CONTRACTCLUSTER
# since we want to transform the long
# version of a word into the short one.
# (In xfst we can achieve this only
# via the replacement operator)
# the definition below is very explicit
# but I didn't find a more elegant way
# to do it using xfst)
```

```
define CSHORTEN
```

```
    c VOWEL c -> {cc}    || VPART _ VPART .o.
    h VOWEL t -> {ht}    || VPART _ VPART .o.
    h VOWEL t VPART y -> {hty}
                                || VPART _ VPART .o.
    l VOWEL c -> {lc}    || VPART _ VPART .o.
    l VOWEL d -> {ld}    || VPART _ VPART .o.
    l VOWEL l -> {ll}    || VPART _ VPART .o.
    l VOWEL m -> {lm}    || VPART _ VPART .o.
    l VOWEL p -> {lp}    || VPART _ VPART .o.
    l VOWEL q -> {lq}    || VPART _ VPART .o.
    l VOWEL t -> {lt}    || VPART _ VPART .o.
    l VOWEL v -> {lv}    || VPART _ VPART .o.
    l VOWEL w -> {lw}    || VPART _ VPART .o.
    l VOWEL y -> {ly}    || VPART _ VPART .o.
    m VOWEL b -> {mb}    || VPART _ VPART .o.
    m VOWEL m -> {mm}    || VPART _ VPART .o.
    m VOWEL n -> {mn}    || VPART _ VPART .o.
    m VOWEL p -> {mp}    || VPART _ VPART .o.
```

```

m VOWEL y -> {my} || VPART _ VPART .o.
n VOWEL c -> {nc} || VPART _ VPART .o.
n VOWEL d -> {nd} || VPART _ VPART .o.
n VOWEL g -> {ng} || VPART _ VPART .o.
n VOWEL g VPART w -> {ngw}
                                || VPART _ VPART .o.
n VOWEL n -> {nn} || VPART _ VPART .o.
n VOWEL q -> {nq} || VPART _ VPART .o.
n VOWEL t -> {nt} || VPART _ VPART .o.
n VOWEL t VPART y -> {nty}
                                || VPART _ VPART .o.
n VOWEL w -> {nw} || VPART _ VPART .o.
n VOWEL y -> {ny} || VPART _ VPART .o.
p VOWEL s -> {ps} || VPART _ VPART .o.
p VOWEL t -> {pt} || VPART _ VPART .o.
c VOWEL w -> {cw} || VPART _ VPART .o.
r VOWEL c -> {rc} || VPART _ VPART .o.
r VOWEL d -> {rd} || VPART _ VPART .o.
r VOWEL m -> {rm} || VPART _ VPART .o.
r VOWEL n -> {rn} || VPART _ VPART .o.
r VOWEL q -> {rq} || VPART _ VPART .o.
r VOWEL r -> {rr} || VPART _ VPART .o.
r VOWEL t -> {rt} || VPART _ VPART .o.
r VOWEL t VPART y -> {rvy}
                                || VPART _ VPART .o.
r VOWEL s -> {rs} || VPART _ VPART .o.
r VOWEL w -> {rw} || VPART _ VPART .o.
r VOWEL y -> {ry} || VPART _ VPART .o.
s VOWEL c -> {sc} || VPART _ VPART .o.
s VOWEL q -> {sq} || VPART _ VPART .o.
s VOWEL s -> {ss} || VPART _ VPART .o.
s VOWEL t -> {st} || VPART _ VPART .o.
s VOWEL t VPART y -> {sty}
                                || VPART _ VPART .o.
s VOWELw ->{sw} || VPART _ VPART .o.
t VOWEL s -> {ts} || VPART _ VPART .o.
t VOWEL t -> {tt} || VPART _ VPART .o.
t VOWEL w -> {tw} || VPART _ VPART .o.
t VOWEL y -> {ty} || VPART _ VPART .o.
k VOWEL s -> {ks} || VPART _ VPART;
;

# functions to remove delimiters

define CLEANSTOPS
  STOP -> [..];
define CLEANBRKS
  BRK -> [..];

# remove all delimiters except for DIA
# (indicating diaeresis)

define CLEANUP
  [DELIMITER - DIA] -> [..];

```

```

define CLEANVAR
  VAR -> [..];

# combining all criteria ...
# QUENYISHCOMPOSED allows for compounds
# like lasse | o
# PUREQUENYISH is more strict (it does
# not allow for vowelclusters)
# SHORTQUENYISH prefers shorter words
# with consonant clusters

define QUENYISHCOMPOSED
  QUENSTRUCTURE .o. ALTERNATE
  .o. CLEANSTOPS ;
define QUENYISHPURE
  [CLEANBRKS
  .o. QUENYISHCOMPOSED];
define QUENYISH
  [QUENYISHPURE |
  QUENYISHCOMPOSED];

define SHORTQUENYISH
  QUENYISH .o. CSHORTEN ;

# quenyish word optionally preceeded
# by two variables
define VARQUENYISH
  [(VAR) (VAR) SHORTQUENYISH];

# Q: why did we need the STOP separator
# at all?

# A: nonquenyia words like 'ht' might
# be regarded as a cluster '+ht'
# or as single consonants in a row
# '+h+t'
# When interpreted as a single cluster
# alternate is satisfied,
# When interpreted as two
# single consonants it satisfies
# quenstructure.
# Thus 'ht' would be regared as valid
# quenyia, which it is not!
# Instead of inserting separators we
# could also have transliterated
# the consonantclusters into single
# tokens and back.

#=====
# (2) THE NOUN
#=====

```

```

# these are the nouns given in the
# appendix of [1]
# cirya (ship)
# lassë (leaf)
# nat (thing)
# ti"e (path)
# elen (star) |
# alda (tree)

# nouns used in example declinations
# from [1]
define NOUNSTEM
    [{cirya} | {lasse} | {nat} |
     {ti|e} | {elen} | {alda}];

# (2.1) COMBINATION
# combination is about what happens
# when the ending is combined
# with the stem.
# in some of the cases the ending is
# simply attached to the stem
# but both vowel modification and vowel
# insertion may occur.

# (2.1.1) VOWEL MODIFICATION

# when the ending is attached to the
# stem two vowels may collide
# usually one vowel will disappear or
# be modified

# the rules below were originally
# associated with individual cases
# but I decided to generalize them.
# as long as no counterexample shows
# up they may be applied to the set of
# all wordforms collectively

# The BRKs are kept, since they may be
# needed later on.

define CONTRACTABLE
    [[? ?] - [i DIA]];

define VMODIFY
    # e gets devoured before i/ii
    e BRK i -> i BRK .o.

    # merging i and ii
    e BRK ii -> ii BRK .o.
    i -> 0 || _ BRK ii .o.

```

```

# aii becomes ai
ii -> i || a BRK _ .o.

# ao becomes o
a -> 0 || _ BRK o .o.

# plural -i instead of -er
# (exception: ti | eř)
e BRK r -> i BRK ||
CONTRACTABLE _ .#. .o.

# plural -i instead of -ir
i BRK r -> i BRK || _ .#. .o.

# merge double i to ii
i BRK i -> ii BRK || _ {nen}
.#. , _ {va} .#. .o.

# merge double i to i
i BRK i -> i BRK .o.

# e gets devoured after u
e -> 0 || u BRK _ .o.

# merge i's if there is a
# leftover diaeresis
i DIA ii -> ii .o.
i DIA i -> ii .o.

# BRKS that have moved to the
# end of the word may be
# deleted;
BRK -> 0 || _ .#.

;

# (2.1.2) VOWEL INSERTION

# if case endings are added to a
# noun ending in a consonant an 'e'
# is often inserted.

define INSERTSGVOWEL
    [..] -> [e | 0] || CONSONANT
    BRK _ CONSONANT ;

# for plural cases an 'i' may be
# inserted
define INSERTPLVOWEL
    [..] -> [i | 0] || CONSONANT
    BRK _ CONSONANT ;

```

## # (2.2) STEMFORMS

# different grammatical rules can  
# give rise to new stemforms which are  
# introduced below  
# currently there are 3 classes of  
# stemforms:

# BASENOUNSTEM: a stem in base form,  
# can be used to build singular and  
# plural wordforms  
# PPLNOUNSTEM: stem is in partitive  
# plural base form. only plural can  
# be formed  
# DUALNOUNSTEM stem is in dual base  
# form. only dual can be formed

# we will use a variable so that  
# successive pluralization depends on  
# the type of the stem

### # (2.2.1) BASE NOUNSTEM

```
define BASENOUNSTEM
    "@U.stem.base@" NOUNSTEM;
```

### # (2.2.2) PARTITIVE PLURAL NOUNSTEM

# the partitive plural is obtained by  
# adding -li (after vowels)  
# or -eli (after consonants) to the stem  
# before the plural ending is formed.

```
define PPLNOUNSTEM
    "@U.stem.ppl@" [[[NOUNSTEM
    0:BRK 0:{li}] .o. VMODIFY
    .o. INSERTSGVOWEL
    .o. VARQUENYISH .o. CLEANUP]];
```

### # (2.2.3) DUAL NOUNSTEM

# the dual nounstem is derived from the  
# base nounstem in the following way:  
# if the last occurring consonant is d  
# or t the rest is cut off  
# and the ending '-u' is directly  
# attached  
# (i.e. tree = alda -> aldū)  
# otherwise a '-t' is appended.

# using the definition below the dual  
# of ti | e would be ti | eñ instead

# of just tū  
# that's because the cutoff doesn't  
# go beyond delimiters, which seems  
# reasonable.  
# if delimiters are to be ignored one  
# would use the alternative:

```
# ignore delimiters :
# define CUTSTEM
#     (DELIMITER | VPART) *
#     -> 0 || [t | d] _ .# . ;
# define NOMFORM
#     u => [t | d | DELIMITER] BRK _
#     .# . , _ ? .o.
#     t =>
#     [ALPHA-t-d] BRK
#     _ .# . , _ ?;
```

# stop at delimiters

```
define CUTSTEM
    (VPART) * -> 0 || [t | d] _
    .# . ;
define NOMFORM
    u => [t | d] BRK _ .# . , _
    ? .o.
    t => [ALPHA-t-d] BRK _ .#
    , _ ?;
```

```
# define DUALNOUNSTEM
    "@U.stem.dl@" [[NOUNSTEM
    .o. CUTSTEM] 0:BRK 0:[t | u]]
    .o. NOMFORM .o. VARQUENYISH
    .o. CLEANUP;
```

```
define DUALNOUNSTEM
    "@U.stem.dl@" [[NOUNSTEM
    .o. CUTSTEM] 0:BRK 0:[t | u]]
    .o. NOMFORM .o. INSERTSGVOWEL
    .o. VARQUENYISH .o. CLEANUP;
```

### # (2.2.4) ALL NOUNSTEMS

```
define ALLNOUNSTEM
    [BASENOUNSTEM | PPLNOUNSTEM |
    DUALNOUNSTEM];
```

## # (2.3) CASES

# the quyenya noun is inflected for nine  
# or ten cases

### # (2.3.1) NOMINATIVE SINGULAR

```

# basic uninflected form of the noun

define NounNomSg
    %+Noun %+Nominative %+Sg .x. 0;
define NounNomSgLex
    ALLNOUNSTEM NounNomSg;

# (2.3.2) ACCUSATIVE SINGULAR
# quenya as spoken in valinor had
# an accusative that was formed by
# lengthening the final VOWEL
# (nouns ending in a consonant
# presumably had no distinct
# accusative)

define NounAccSg
    %+Noun %+Accusative %+Sg
    .x. 0 ;
define LENGTHENLASTVOWEL
    a -> aa, e -> ee, i -> ii, o ->
    oo, u -> uu || _ .#.;
define NounAccSgLex
    [ALLNOUNSTEM NounAccSg]
    .o. LENGTHENLASTVOWEL;

# (2.3.3) GENITIVE SINGULAR
# the genitive has the ending -o
# if the stem ending is a consonant
# then the -o is attached
# if the stem ending is -a it gets
# displaced by -o but other vowels are
# not displaced
# this can also be regarded as
# attaching an o in any case
# and contracting the ending -ao to -a
# (see VSHORTEN)

define NounGenSg
    %+Noun %+Genitive %+Sg .x. BRK
    o ;
define NounGenSgLex
    [ALLNOUNSTEM NounGenSg] ;

# (2.3.4) POSSESSIVE SINGULAR
# also called 'associative' or
# 'adjectival' case.
# it has the ending -va.
# in case of consonants softening
# occurs: -wa
# ATTENTION: this ending must be
# attached to the accusative form(!!!)

# modify the accusative lexicon to
# match the +Possesive tag

read regex NounAccSgLex;
substitute symbol %+Possesive for
%+Accusative;
define ACCUFORM;

# we don't need to care about
# combination rules since
# the accusative form does'nt have
# an ending

define NounPosSgEnding
    0:[BRK [v | w] a ];
define NounPosSgSelect
    w => CONSONANT BRK _ a .#. .o.
    v => VOWELISH BRK _ a .#. ;

# applying the softening rule
define NounPosSgLex
    [ACCUFORM NounPosSgEnding]
    .o. NounPosSgSelect;

# (2.3.5) DATIVE SINGULAR
# the dative has the ending -n
# this ending generally translates as
# the preposition 'for' or 'to'

define NounDatSg
    %+Noun %+Dative %+Sg .x. BRK
    n ;
define NounDatSgLex
    ALLNOUNSTEM NounDatSg;

# (2.3.6) LOCATIVE SINGULAR
# the locative has the ending -sse
# it carries the meaning 'on' or 'in'

define NounLocSg
    %+Noun %+Locative %+Sg .x. BRK
    {sse} ;
define NounLocSgLex
    ALLNOUNSTEM NounLocSg;

# (2.3.7) ABLATIVE SINGULAR
# the ablative has the ending -llo
# it carries the meaning 'from' or
# 'out of'

define NounAbISg
    %+Noun %+Ablative %+Sg .x. BRK
    {llo} ;

```

```

define NounAblSgLex
    ALLNOUNSTEM NounAblSg;

# (2.3.8) ALLATIVE SINGULAR
# the allative has the ending -nna
# meaning 'to' 'into' or 'upon'

define NounAllSg
    %+Noun %+Allative %+Sg .x. BRK
    {nna} ;
define NounAllSgLex
    ALLNOUNSTEM NounAllSg;

# (2.3.9) INSTRUMENTAL SINGULAR
# the instrumental case has the
# ending -nen
# and marks the instrument with which
# something is done
# or simply the reason why something
# happens.

define NounInsSg
    %+Noun %+Instrumental %+Sg
    .x. BRK {nen} ;
define NounInsSgLex
    ALLNOUNSTEM NounInsSg;

# (2.3.10) RESPECTIVE SINGULAR
# also known as the mystery case

define NounResSg
    %+Noun %+Respective %+Sg
    .x. BRK s ;
define NounResSgLex
    ALLNOUNSTEM NounResSg;

# (2.3.11) COLLECTING ALL SINGULAR
# CASES

# the lexicon for singular wordforms is
# now restricted to wordforms derived
# from BASESTEMS
# but the individual lexical categories
# were not(!)
# this is because we want to reuse the
# definitions of the categories for
# the plural form.

define NOUNSTEMSG
    ["@U.stem.base@" [
    NounNomSgLex |
    NounAccSgLex |
    NounGenSgLex |

NounPosSgLex |
NounDatSgLex |
NounLocSgLex |
NounAblSgLex |
NounAllSgLex |
NounInsSgLex |
NounResSgLex ]];

## (2.4) THE QUENYA NUMBERS

# the numbers are singular, plural,
# partitive plural, and dual

# (2.4.1) THE PLURAL

# (2.4.1.1) NOMINATIVE PLURAL
# if the stem ends in a vowel (except
# e) the ending is '-r' is attached
# if the stem ends in a consonant the
# ending '-i' is attached
# when the stem ends in -e this is
# transformed into an -i (see VSHORTEN)

define NounNomPl
    [%+Noun %+Nominative %+Pl]
    .x. [BRK [i | r]];

# general case first
define PLURALENDING
    r => VOWELISH BRK _ .# , _ ?
    .o. i => CONSONANT BRK _ .# ,
    _ ?;

# special case: contract -er to -i
# a final -e in the stem will thus
# be replaced by -i
# but if the stem ends in -i | e (as
# in ti | e = path)
# the contraction is not applied
# (see VSHORTEN)

define NounNomPlLex
    [ALLNOUNSTEM NounNomPl]
    .o. PLURALENDING;

# (2.4.1.2) ACCUSATIVE PLURAL
# accusative plural ends in -ii
# when the stem ends in -e this is
# transformed to -ii rather than
# attaching -ii (i.e. lassii)

```

```

# aii is contracted to the more common
# diphthong ai (i.e. ciryai)
# (see VSHORTEN)

define NounAccPl
    %+Noun %+Accusative %+Pl
    .x. BRK ii;
define NounAccPlLex
    [ALLNOUNSTEM NounAccPl];

# (2.4.1.3) GENITIVE PLURAL
# -on is added to the nominative plural

# ATTENTION: this ending must be
# attached to the nominative form(!!!)

# modify the nominative lexicon to
# match the +Genitive tag
read regex NounNomPlLex;
substitute symbol %+Genitive for
%+Nominative;
define NOMIFORM;

# apply combination rules
define NounGenPl
    NOMIFORM .o. VMODIFY
    .o. INSERTPLVOWEL;

# attaching the ending to the
# accusative form
# depending on what you consider the
# stem there are two variants
# (i.e. ciryaron = ciryařon or
# ciryarõn)

# variant A: insert a new BRK
define NounGenPlLexA
    [NounGenPl .o. CLEANUP]
    0:[BRK {on}];

# variant B: keep the old BRK
define NounGenPlLexB
    NounGenPl 0:{on};

define NounGenPlLex
    [NounGenPlLexA |
    NounGenPlLexB];

# (2.4.1.4) POSSESSIVE PLURAL
# has the ending -iva
# when the stem ends in -e the e will
# be devoured
# (see VSHORTEN)

define NounPosPl
    %+Noun %+Possessive %+Pl
    .x. BRK {iva};
define NounPosPlLex
    [ALLNOUNSTEM NounPosPl];

# (2.4.1.5) LOCATIVE PLURAL
# like locative for singular
# but -n is added

read regex NounLocSgLex;
substitute symbol %+Pl for %+Sg;
define SGFORM;
    define NounLocPlLex
        SGFORM 0:n;

# (2.4.1.6) ABLATIVE PLURAL
# like ablative for singular
# but -n or -r is added

read regex NounAblSgLex;
substitute symbol %+Pl for %+Sg;
define SGFORM;
    define NounAblPlLex1
        SGFORM 0:n;
define NounAblPlLex2
    SGFORM 0:r;

# (2.4.1.7) ALLATIVE PLURAL
# like allative for singular
# but -r is added

read regex NounAllSgLex;
substitute symbol %+Pl for %+Sg;
define SGFORM;
    define NounAllPlLex
        SGFORM 0:r;

# (2.4.1.8) DATIVE PLURAL
# like dative for singular
# but an -i- is inserted between stem
# and ending
# if the stem ends in -e this is
# devoured

define INSERTI
    [..] -> i || BRK _;

read regex NounDatSgLex;
substitute symbol %+Pl for %+Sg;
define SGFORM;

```

```

define NounDatPILex
    SGFORM .o. INSERTI;

# (2.4.1.9) INSTRUMENTAL PLURAL
# like instrumental for singular
# but an -i- is inserted between stem
# and ending

read regex NounInsSgLex;
substitute symbol %+Pl for %+Sg;
define SGFORM;

define NounInsPILex
    SGFORM .o. INSERTI;

# (2.4.1.10) RESPECTIVE PLURAL
# like respective for singular
# but an -i- is inserted between stem
# and ending

read regex NounResSgLex;
substitute symbol %+Pl for %+Sg;
define SGFORM;

define NounResPILex
    SGFORM .o. INSERTI;

# (2.4.1.11) COLLECTING ALL PLURAL
# CASES

define NOUNSTEMPL
    ["@U.stem.base@" [
        NounNomPILex |
        NounAccPILex |
        NounGenPILex |
        NounPosPILex |
        NounLocPILex |
        NounAblPILex1 |
        NounAblPILex2 |
        NounAllPILex |
        NounDatPILex |
        NounInsPILex |
        NounResPILex ]];

# (2.4.2) THE PARTITIVE PLURAL

# the semantic function of the Ppl. is
# not fully understood,
# but it seems to denote "some out of
# a group".
# see stemforms!!

# derived from Plural definition

# but refering only to Ppl stemforms
# instead of base stemforms

# according to the examples in [1]
# locative ablative and allative may
# also be derived from the singular
# form
# evidently the alternative form for
# the ablative (NounAblPILex2) is not
# used in Ppl

read regex ["@U.stem.ppl@" [
    NounNomPILex |
    NounAccPILex |
    NounGenPILex |
    NounPosPILex |
    NounLocPILex |
    NounAblPILex1 |
    NounAllPILex |
    NounDatPILex |
    NounInsPILex |
    NounResPILex |

    NounLocSgLex |
    NounAblSgLex |
    NounAllSgLex ]
];

substitute symbol %+Ppl for %+Pl;
substitute symbol %+Ppl for %+Sg;
define NOUNSTEMPPL;

# (2.4.3) THE DUAL
# used to refer to a pair of objects
# (hands trousers etc.)
# this form is too weird to be derived
# from any of the other forms
# therfor we define it
    from scratch.

# some of the cases are formed using a
# postfix that somehow interweaves into
# the end of the stem. (thats why we
# call it circumpostfix)

define makecircumpostfix
    define CIRCUMPOSTFIX
        [.. -> BRK LEFT || _ FIX
        .#. .o. [.. -> RIGHT || FIX
        _ .#.;

# (2.4.3.1) - (2.4.3.3) NOMINATIVE,

```

```

# ACCUSATIVE, GENITIVE
# these three wordforms are exactly
# analogous to the singular form
# but based on the dual stem.
# All other endings are similar but
# eith a 't' inserted somewhere in
# the ending

# (2.4.3.4) POSSESSIVE
# analogous to singular but based on
# nominative rather than the accusative
# form

define NounPosDILex
  [ALLNOUNSTEM [%+Noun
    %+Possessive %+DI]:0
  NounPosSgEnding]
  .o.NounPosSgSelect;

# (2.4.3.5) LOCATIVE
# *-tse is postfixed replacing the
# final -t of the stem
# or if the stem ends in '-u' the
# '-sse' is added

define POSTFIX
  [..] -> BRK {sse} || u _ .#;
.o. t -> BRK {tse} || _ .#;

define NounLocDILex
  [DUALNOUNSTEM [%+Noun
    %+Locative %+DI]:0]
  .o. POSTFIX;

# (2.4.3.6) DATIVE
# n-* is prefixed before the final t of
# the stem
# or if the stem ends in '-u' then
# 'en is added

define FIX t;
define LEFT n;
define RIGHT 0;
makecircumpostfix;

define POSTFIX
  [..] -> BRK {en} || u _ .#;

define NounDatDILex
  [DUALNOUNSTEM [%+Noun %+Dative
    %+DI]:0] .o. CIRCUMPOSTFIX

.o. POSTFIX;

# (2.4.3.7) ABLATIVE
# the ending -l*-o is circumfixed
# around the final t of the stem
# or if the stem ends in '-u' then -llo
# is added

define LEFT l;
define RIGHT o;
makecircumpostfix;

define POSTFIX
  [..] -> BRK {llo} || u _ .#;

define NounAbIDILex
  [DUALNOUNSTEM [%+Noun
    %+Ablative %+DI]:0]
  .o. CIRCUMPOSTFIX .o. POSTFIX;

# (2.4.3.8) ALLATIVE
# the ending -n*-a is circumfixed
# around the final t of the stem
# or if the stem ends in '-u' then -nna
# is added

# general template for a postcircumfix
# ending

define LEFT n;
define RIGHT a;
makecircumpostfix;
define POSTFIX
  [..] -> BRK {nna} || u _ .#;

define NounAlIDILex
  [DUALNOUNSTEM [%+Noun
    %+Allative %+DI]:0]
  .o. CIRCUMPOSTFIX .o. POSTFIX;

# (2.4.3.9) INSTRUMENTAL
# the ending -n*-en is circumfixed
# around the final t of the stem
# or if the stem ends in '-u' then -llo
# is added

define LEFT n;
define RIGHT {en};
makecircumpostfix;

define POSTFIX

```

```

[.] -> BRK {nen} || u _ .#.;

define NounInsDILex
  [DUALNOUNSTEM [%+Noun
  %+Instrumental %+DI]:0]
  .o. CIRCUMPOSTFIX .o. POSTFIX;

# (2.4.3.10) RESPECTIVE
# -es is added
# if the stem ends in u -ues will be
# transformed to -us
# (see VMODIFY)

define CONTRACTE
  e -> 0 || BRK u _ .#.;
define NounResDILex
  [DUALNOUNSTEM [%+Noun
  %+Respective %+DI];[ BRK
  {es}]];

# (2.4.3.11) COLLECTING ALL DUAL CASES

read regex ["@U.stem.dl@" [
  NounNomSgLex |
  NounAccSgLex |
  NounGenSgLex |

  NounPosDILex |
  NounDatDILex |
  NounLocDILex |
  NounAblDILex |
  NounAllDILex |
  NounInsDILex |
  NounResDILex ]];

substitute symbol %+DI for %+Sg;
define NOUNSTEMDL;

# (2.5) THE NOUN LEXICON

define NOUNLEXSG
  NOUNSTEMSG .o. VMODIFY
  .o. INSERTSGVOWEL
  .o. VARQUENYISH .o. CLEANUP;
define NOUNLEXPL
  NOUNSTEMPL .o. VMODIFY
  .o. INSERTPLVOWEL
  .o. VARQUENYISH .o. CLEANUP;
define NOUNLEXPPL
  NOUNSTEMPPL .o. VMODIFY
  .o. VARQUENYISH .o. CLEANUP;
define NOUNLEXDL

```

```

NOUNSTEMDL .o. VMODIFY
.o. VARQUENYISH .o. CLEANUP;

# define NOUNLEXSG
  NOUNSTEMSG .o. VMODIFY
  .o. INSERTSGVOWEL .o. CLEANUP;
# define NOUNLEXPL
  NOUNSTEMPL .o. VMODIFY
  .o. INSERTPLVOWEL .o. CLEANUP;
# define NOUNLEXPPL
  NOUNSTEMPPL .o. VMODIFY
  .o. CLEANUP;
# define NOUNLEXDL
  NOUNSTEMDL .o. VMODIFY
  .o. CLEANUP;

define NOUNLEX
  [NOUNLEXSG | NOUNLEXPL |
  NOUNLEXPPL | NOUNLEXDL];

# (2.6) TESTING

# (2.6.1) TEST FILES

# test.txt contains sample tagged
# stemforms and the corresponding
# wordforms

# (2.6.2) FILTERS
# first we have to filter out the
# comments
# (lines starting with a hash sign)
# the define definition
# below takes the current net on
# the stack
# and precedes it with a filter for
# removing the comments

define uncomment
  define TMP;
  read regex [? - %#] ?* .o. TMP;
  undefine TMP;

# now we want to read just the lefthand
# or the righthand side of an equation

# any number of spaces before or
# after the term will be matched /
# or not matched
# so that it is up to the following

```

```

# step to decide whether
# the spaces should belong to the term
# or not

# possible space

define PSPACE
    (SPACE);
define EQUALS
    PSPACE %= PSPACE;
define TERMCHAR
    [ALPHA | MARKER | DIA];

define TERM
    TERMCHAR*;

define lhs
    define TMP;
    read regex [PSPACE]:0 TERM
    [EQUALS TERM PSPACE]:0 .o. TMP
    ; undefine TMP;
    define rhs
    define TMP;
    read regex [PSPACE TERM
    EQUALS]:0 TERM [PSPACE]:0
    .o. TMP; undefine TMP;

# if variables are to be used we need
# to account for them explicitly!
# make sure that space for two
# variables is created (!)

define VARTERM
    VAR VAR TERM;

define vlhs
    define TMP;
    read regex [PSPACE]:0 VARTERM
    [EQUALS TERM PSPACE]:0 .o. TMP
    ; undefine TMP;
    define vrhs
    define TMP;
    read regex [PSPACE TERM
    EQUALS]:0 VARTERM [PSPACE]:0
    .o. TMP; undefine TMP;

# (2.6.3) EXAMPLES

# for testing simply call the aliases
# defined below
# then type something like : down <
# test.txt

# (2.6.3.1) EXAMPLE 1 (quentest) check
# if all wordforms are really quenyish

alias quentest
    read regex QUENYISH;
    uncomment; # remove comments
    rhs; # parse right side only
END;

# (2.6.3.2) EXAMPLE 2 (downtest)
# creating wordforms

alias downtest
    read regex NOUNLEX;
    uncomment; # remove comments
    vlhs; # parse left side only
END;

# (2.6.3.2) EXAMPLE 3 (uptest)
# parsing wordforms

define uptest
    read regex NOUNLEX;
    # upwards application!
    invert net;
    uncomment; # remove comments
    vrhs; # parse right side only
END;

# 3. MISCELLANEOUS

# (3.1) USEFUL DEFINITIONS
# some define definitions
# to simplify interaction
# with xfst

define push
    read regex;

# (3.1) LOADING THE LEXICON
# put the lexicon on top of the stack
# when this file is parsed

# push NOUNLEX;

```

## 6.2 xfst2dot.pl

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   xfst2dot.pl
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% (c) by Martin Schneider.

% this program is written in swi-prolog.
% it converts finite state automata
% created by xfst from prolog format
% to graphviz .dot format.
% use AT&T's graph layout program 'dot'
% to print the automaton in postscript
% or any other format of your choice.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   MAIN PROGRAM
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% standard settings, overridden by flags
% passed as argument

:- dynamic flag/1.

flag(compact:on).
flag(ranking:off).

xfst2dot(Filename1, Filename2):-
    xfst2dot(Filename1, Filename2,
    []).

xfst2dot(Filename1, Filename2, Flags):-
    xfst2dot(Filename1, Filename2,
    node, link, Flags).

xfst2dot(Filename1, Filename2, NodeID, LinkID, Flags):-
    forany(member(F, Flags),
    setflag(F)),
    readXfst(Filename1, NodeID,
    LinkID),
    (flag(compact:on) *->
    compactEdges(LinkID); true),
    printGraph(Filename2, NodeID,
    LinkID).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   READ XFST
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% we can read single, as well as multiple
% files (each file is assumed to contain
% a single automaton.)
% each automaton will be assigned a
% separate level of the output graph
% the initial and final nodes will be aligned
% using graphviz's ranking mechanism

readXfst(Filenames, NodeID, LinkID):-
    retractGraph(NodeID, LinkID),
    readXfst(Filenames, NodeID,
    LinkID, 1).

% multiple files
readXfst([Filename], NodeID, LinkID, Level):-
    readXfst(Filename, NodeID,
    LinkID, Level).

readXfst([Filename|MoreFiles], NodeID, LinkID, Level):-
    readXfst(Filename, NodeID,
    LinkID, Level),
    NewLevel is Level + 1,
    readXfst(MoreFiles, NodeID,
    LinkID, NewLevel).

% single file
readXfst(Filename, NodeID, LinkID, Level):-
    retractXfst,
    consult(Filename),
    createNodes(NodeID, Level),
    createLinks(LinkID, Level).

% composing unique nodenames for each level.
nodeName(Number, Level, NodeName):-
    dotnodename(Name),
    concat_atom([Name, '_',
    Number, '_', Level], NodeName).

% start nodes are assigned rank 1, final nodes 2
% , intermediate nodes will not be ranked (0)
createNodes(NodeID, Level):-
    findall(Node, final(_Net,
    Node), FinalNodes),
    last(FinalNodes, LastNode),
    numlist(1, LastNode, AllNodes),
    subtract(AllNodes, FinalNodes,
    IntermediateNodes),
    startnodelayout(StartLayout),

```

```

internodelayout(InterLayout),
finalnodelayout(FinalLayout),
createNode(NodeID, 0, Level,
1, StartLayout),
forall(member(Number,
IntermediateNodes),
createNode(NodeID, Number,
Level, 0, InterLayout)),
forall(member(Number,
FinalNodes), createNode(NodeID,
Number, Level, 2,
FinalLayout)).

createNode(NodeID, Number, Level, Rank):-
createNode(NodeID, Number,
Level, Rank,").

createNode(NodeID, Number, Level, Rank, Layout):-
nodeName(Number, Level,
NodeName),
NewNode =.. [NodeID, NodeName,
Rank, Number],
assert(NewNode),
assert(layout(NewNode,
Layout)).

% convert fsa-output to printable form
output_to_atom(A1:A2, B):-
string_to_atom(A1, B1),
string_to_atom(A2, B2),
concat_atom([B1, B2],':', B).

output_to_atom(A, B):-
string_to_atom(A, B).

createLinks(LinkID, Level):-
forall(arc(_Net,
Number1, Number2, Label),
(output_to_atom(Label,
NewLabel), createLink(LinkID,
Number1, Number2, NewLabel,
Level))).

createLink(LinkID, Number1, Number2,
Label, Level):-
createLink(LinkID, Number1,
Number2, Label, Level,").

createLink(LinkID, Number1, Number2,
Label, Level, Layout):-
nodeName(Number1, Level,
NodeName1),
nodeName(Number2, Level,
NodeName2),
NewLink =.. [LinkID, NodeName1,
NodeName2, Label],
assert(NewLink),
assert(layout(NewLink,
Layout)).

retractGraph(NodeID, LinkID) :-
UnlabeledNode =.. [NodeID,
_, _],
LabeledNode =.. [NodeID, _,
_, _],
UnlabeledLink =.. [LinkID,
_, _],
LabeledLink =.. [LinkID, _,
_, _],
retractall(UnlabeledNode),
retractall(LabeledNode),
retractall(UnlabeledLink),
retractall(LabeledLink),
retractall(layout(UnlabeledNode,
_)),
retractall(layout(LabeledNode,
_)),
retractall(layout(UnlabeledLink,
_)),
retractall(layout(LabeledLink,
_)).

retractXfst :-
retractall(network(_)),
retractall(arc(_, _, _, _)),
retractall(final(_, _)).

%% %% %% %% %% %% %% %% %% %% %% %% %% %% %%
%
% PRINT GRAPH
%
%% %% %% %% %% %% %% %% %% %% %% %% %% %% %%

printGraph(Filename, NodeID, LinkID):-
tell(Filename),
printHeader,
printNodes(NodeID, LinkID),
printFooter,
told.

% dot format
dotformat(ul, "\ t %w -> %w [ %w ] ;
\n").
dotformat(ll, "\ t %w -> %w [ label =
\"%w\", %w ] ; \n").
dotformat(un, "\ t node [ %w ] ;

```

```

        %w; \n").
dotformat(ln, "\t node [ label = \"%w \",
        %w ]; %w; \n").

doheader(['/* .dot file for graphviz
generated by xfst2dot */',
'digraph nn {',
'\t rankdir = LR;',
'\t orientation =
portrait;',
'\t ratio = fill;',
'\t size = "7, 11";',
'\t edge [ ];',
'\t node [ color = black,
shape = circle, style =
filled ];']).

dotfooter(['}']).

rankheader(['\t { rank = same}').
rankfooter(['}\n']).

newline('\n').
separator('; ').
dotnodename('node').
nolabel("").
multilabelseparator(', ').

startnodelayout('fillcolor = lightsalmon').
internodelayout('fillcolor = white').
finalnodelayout('fillcolor = lightblue').

% printing functions

printHeader:-
    doheader(Header),
    newline(Cr),
    printList2(Header, Cr).
:printFooter:-
    dotfooter(Header),
    newline(Cr),
    printList2(Header, Cr).

printNodes(NodeID, LinkID):-
    UnlabeledNode =.. [NodeID,
A1, A2],
    LabeledNode =.. [NodeID, B1,
B2, B3],
    UnlabeledLink =.. [LinkID,
C1, C2],
    LabeledLink =.. [LinkID, D1,
D2, D3],

%collect data
findany([A0, A1,
A2], (UnlabeledNode,
layout(UnlabeledNode, A0)),
UNodes),
findany([B3, B0,
B1], (LabeledNode,
layout(LabeledNode, B0)),
LNodes),
findany([C1, C2,
C0], (UnlabeledLink,
layout(UnlabeledLink, C0)),
ULinks),
findany([D1, D2, D3,
D0], (LabeledLink,
layout(LabeledLink, D0)),
LLinks),

%print node lines
printDotLine(un, UNodes),
printDotLine(ln, LNodes),

%print rank list
findany([B2, B1], LabeledNode,
LNodes2),
findany([A2, A1],
UnlabeledNode, UNodes2),
append(UNodes2, LNodes2,
Nodes),
(flag(ranking:on) *->
printRankList(Nodes); true),
%print link lines
printDotLine(ul, ULinks),
printDotLine(ll, LLinks).

printRankList([]).
printRankList(Nodes):-
    sort(Nodes, Ranksorted),
    last(Ranksorted, [Rank|_Rest]),
    printRankList(Nodes, Rank).

printRankList(_Nodes, 0).

printRankList(Nodes, Rank):-
    findall(Name, member([Rank,
Name], Nodes), ThisRank),
    rankheader([RankHeader]),
    rankfooter(RankFooter),
    append([RankHeader|ThisRank],
RankFooter, List),
    separator(Sep),
    printList(List, Sep),
    findall([LoRank, Name],

```

```

(member([LoRank, Name], Nodes),
LoRank<Rank), RestRank),
printRankList(RestRank).

% formatted output
printDotLine(_, []).
printDotLine(Format, [First|Rest]):-
    dotformat(Format, Fstring),
    writef(Fstring, First),
    printDotLine(Format, Rest).

% separator between every element
printList(List, Sep):-
    printList(List, Sep,").

% separator after every element
printList2(List, Sep):-
    printList(List, Sep, Sep).

printList([], _Sep, _Final).

printList([Last], _Sep, Final):-
    writef(Last),
    writef(Final).

printList([First|Rest], Sep, Final):-
    writef(First),
    writef(Sep),
    printList(Rest, Sep, Final).

%%%%%%%%%%%%%%
%
% GRAPH MANIPULATION
%
%%%%%%%%%%%%%%

% turn all links into labeled links
labelLinks(LinkID):-
    nolabel(Nolabel),
    UnlabeledLink =.. [LinkID,
Source, Target],
    LabeledLink =.. [LinkID,
Source, Target, Nolabel],
    findany(LabeledLink,
UnlabeledLink, LLlist),
    retractall(UnlabeledLink),
    assertall(LLlist).

% merge all edges that have source and
% destination in common
% the layout information will be taken
% from the first edge encountered
% layout info for the old edges will

```

```

% not be removed

compactEdges(LinkID):-
    labelLinks(LinkID),
    LabeledLink =.. [LinkID,
Source, Target, _Label],
    findany(LabeledLink, Linklist),
    retractall(LabeledLink),
    partition((Source, Target),
LabeledLink, Linklist,
Sublists),
    forany(member([First|Sub],
Sublists),
(
mergeEdges([First|Sub],
Merged),
assert(Merged),
layout(First,
FirstLayout),
reassert(layout(Merged,
FirstLayout))
)
).

% merge all edges in the list
% (does not modify the database)
mergeEdges([First|Rest], Merged):-
    First =.. [LinkID, Source,
Target, _],
    Merged =.. [LinkID, Source,
Target, Label],
    findall(Label, (member(Link,
[First|Rest]), Link =..[LinkID,
Source, Target, Label],
islabel(Label)), LabelList),
    multilabelseparator(Separator),
    concat_atom(LabelList,
Separator, Label).

% check if some term is a label
islabel(Label):-
    nolabel(Nolabel),
    \+ Label = Nolabel.

%%%%%%%%%%%%%%
%
% SUBROUTINES
%
%%%%%%%%%%%%%%

% working with flags

flags :- listing(flag).

```



```

layout(slink(_, _),").
layout(slink(_, _),").

snode(a, 1).
snode(d, 2).
snode(b, 2,'foo').
snode(c, 3,'bar').

slink(a, b).
slink(a, c).
slink(a, d).
slink(d, c).
slink(b, c,'foobar').

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   SAMPLE XFST AUTOMATON
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% sample transducer

:- dynamic network/1, arc/4, final/2.

% name of the network
network(net_sample).

% arcs/links

arc(net_sample, 0, 1, "x":"0").
arc(net_sample, 1, 2, "f":"d").
arc(net_sample, 1, 3, "e":"a").
arc(net_sample, 2, 4, "s":"o").
arc(net_sample, 3, 5, "r":"t").
arc(net_sample, 4, 7, "t").
rc(net_sample, 5, 6, "o":"&").
arc(net_sample, 6, 7, "x":"t").

% final node
final(net_sample, 7).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   TEST FUNCTIONS
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% print sample graph
dotTest:- printGraph('dotTest.dot', snode, slink).

% print sample xfst automaton
xfstTest :-
    readXfst('xfst2dot', xnode,
            xlink),
    printGraph('xfstTest.dot',
            xnode, xlink).

```

## 7 Sources

### 7.1 Literature

- [L1] Pesch Helmut W., "Elbisch - Grammatik, Schrift und Wörterbuch der Elben-Sprache von J.R.R: Tolkien", 2003, Bastei Lübbe Taschenbuch
- [L2] Krege Wolfgang, "Elbisches Wörterbuch Nach J.R.R. Tolkien", 2003, Klett-Cota

### 7.2 Websites

- [W1] <http://www.uib.no/People/hnohf/kenya.htm>
- [W2] <http://home.ix.netcom.com/heensle/lang/elvish/elvish.html>
- [W3] <http://www.langmaker.com/>