Gefördert vom
bmb+f
Bundesministerium für
Bildung, Wissenschaft,
Forschung und Technologie

Verbmobil
Verbundvorhaben

X-Vario - Using the XFST Calculus for the Generation of Speech Variants

Ana Paula Quirino Simões

Universität Bielefeld

Report 235
August 1999

August 1999

Ana Paula Quirino Simões

Universität Bielefeld (UBI)
Fakultät für Linguistik und Literaturwissenschaft
Universitätsstr. 25
Postfach 10 01 31
33501 Bielefeld

Tel.: (0521) 106 - 3510
Fax: (0521) 106 - 6008
e-mail: `aquirino@Spectrum.Uni-Bielefeld.DE`

Gehört zum Antragsabschnitt: 1.3.1 Aufbau und Optimierung des Aussprachelexikons
6.4.1 Kommunikation, WWW-Dokumentation, Sprecherdatenbasis

Die vorliegende Arbeit wurde im Rahmen des Verbundvorhabens Verbmobil vom Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (BMBF) unter dem Förderkennzeichen 01 IV 101 B 2 gefördert. Die Verantwortung für den Inhalt dieser Arbeit liegt bei den Autoren.

ISSN 0949–6084

Contents

1 Introduction

One problem of automatic speech recognition is the high degree of phonetic variability in continuous speech. Words are assumed to have a so-called *canonical pronunciation*, which is the one listed in pronouncing dictionaries. This form usually corresponds to clear, isolated speech and does not take in account phenomena like assimilation and coarticulation. However, the canonical form is rarely realised in continuous speech. The mapping of the speech signal onto a sequence of lexical representations can get very complicated without additional information about possible variants of the canonical form.

There are basically two different approaches to solve this problem, which are referred to as *statistical* versus *rule-based* approaches. Statistical-based systems aim to map the speech signal onto a set of lexical representations using pattern classification techniques and stochastics. The most widespread technique in this paradigm is Hidden Markov Modelling. The rule-based approach aims at performing this mapping by making explicit the linguistic knowledge involved in human speech recognition.

The mapping of phonetic realisation onto the level of lexical representation (here assumed to be the canonical representation) does not occur at random, but is controlled by highly regular phonetic knowledge, which can efficienlty be described in terms of phonetic rules. These rules can be made explicit and be applied to generate all possible variants of canonical forms to expand the lexicon, which should result in a better recognition accuracy for both statistical and rule-based systems.

The rules presented in the present paper are taken from Katrin Kirchhoff's *Two-level Modelling of Speech Variant Rules* ([Kir95]), where she presents them in form of a declarative knowledge base, concentrating on phonotypical variation, i.e. those sound changes which affect the phonemic identity of sounds and therefore can be captured by segmental labelling of speech files. In the present paper, a specific implementation of this knowledge base will be discussed, as well as its application to the *Bielefeld Lexical Database* ([LGES98]), i.e. the generation of a set of speech variants from the existing lexicon. The present implementation was done with the *XFST*-system for reasons of efficiency and declarativeness, as will be discussed below. Some of the rules were slightly altered for specific purposes, and some of them were not implemented.

Kirchhoff states her rules for German in a declarative way, i.e. the form in which phonetic knowledge is represented is independent of the way in which this knowledge is used. Her rules are modelled using a two-level approach. In the present project, the rules are implemented by means of an FST-calculus, which will be described step by step in the sections below.

After some introductory words about the *XFST*-system and the reasons why it was employed for this implementation, the internal architecture of the *x-vario* system will be presented. The exact definition of the implemented automata follows then, as well as the definition of each particular rule of the system. Finally, some statistical figures are analysed and some concluding remarks follow.

2 The FST-calculus

Speech variant rules are always optional rules. Context-free rules may be applied whenever the specified segment is present in the input string. Context-sensitive rules may be applied when additionally a matching context is found. Therefore, multiple variant forms can be derived from a single canonical representation; possibly multiple identical output forms are generated by different rules, which should be avoided in order not to inflate the lexicon unnecessarily. Some of the rules can be applied sequentially, some of them *must* be applied sequentially.

The issues were treated in the following ways in the present implementation:

- In order to make sure that all possible variants are generated, all rules were implemented and compiled into one sole rule-automaton, which encodes all of the rules, so that all possible variants always will be generated. Through the single rule-automaton, every single rule is applied to every single input string. Whenever it is applicable, an output string will be generated.
- In order to guarantee that the rules apply only in a given context, this context had to be defined in the regular expressions for the single rules. In some cases, this resulted in very large transducers, as each possible context had to be stated separately. Such rules could not be implemented using the operation *one-plus* (cf. section 3), because with this operation, the *XFST*-system ignores the context defined for the application of the rule. The part where the substitution of the phonemes itself is described was defined as the mapping between an upper and a lower language realised by one specific transducer. Context-free rules were implemented *one-plus*, and can therefore potentially generate multiple output strings.
- In order to avoid multiple identical output, the output strings will be sorted using an awk-script which merges equal lines into a single one.
- Rules which have to be applied sequentially are combined into one complex transducer consisting of a cascade of simple transducers. This complex transducer was then treated as a single rule when the large rule transducer was defined. This way, the order in which the rules in question apply is fixed.

Johnson [Joh72] noted that phonological rules are equivalent in power to *finite state transducers* (FST) under the condition that they are barred from applying to their own output. FSTs are extensions of *finite state automata* (FSAs). Conceptually, one FSA consists of a read head and a tape. Every time one symbol can be read and recognized, the automaton advances to the next state and tries to read the next symbol on the tape according a certain transition function. If all symbols can be read this way and the automaton ends up in a final state, the input string on the tape is said to be accepted. In the case of FSTs, it is assumed that there is a second tape - the output tape - where the automaton can write after reading each symbol. This way, an FST can serve as a translator, defining a mapping between possible input and output strings.

Formally, a (two-tape) FST A is defined as a sextuple $A = < \phi, \Sigma_1, \Sigma_2, \delta, \lambda, S >$, where [1]

ϕ is the finite set of states,
Σ_1 is the finite input alphabet ($\Sigma_1 \cap \phi = \emptyset$),
Σ_2 is the finite output alphabet ($\Sigma_2 \cap \phi = \emptyset$),
δ is the transition function $\delta : \phi \times \Sigma_1 \rightarrow \phi$,
λ is the output function $\lambda : \phi \times \Sigma_1 \rightarrow \Sigma_2$, and
$S \epsilon \phi$ is the initial state.

The definition above describes a *deterministic* FST, i.e. both the transition and the output funtion produce singleton sets. All transducers used in the present implementation are deterministic.

[1] this is the definition of a *Mealy Machine* [HU93], one kind of FST whose output is associated with its transitions rather than with a current state (notice that the output function is defined as :$\lambda : \phi \times \Sigma_1 \rightarrow \Sigma_2$)

3 Why XFST?

XFST (Xerox Finite State Tool) is a general-purpose Unix application for calculating with finite-state networks, based on long-term Xerox research initiated by Ronald M. Kaplan and Martin Kay at the Xerox Palo Alto Research Center in the early 1980s. XFST can read finite-state networks from binary or Prolog files or compile them from regular expressions and text files very efficiently. The networks can be simple networks or finite-state transducers. They can be combined by means of a variety of operations, such as union and composition. The resulting networks can be saved into a binary, a Prolog or a text file. The user may apply a network to strings to determine whether the string is accepted by the network or to transform it to another string if the network is a transducer. XFST provides many ways to get information about a network and to inspect and modify its structure.

The manual for *XFST* can be found at:
`http://www.xrce.xerox.com/research/mltt/fsSoft/docs/fst-97/xfst97.html`

Here are the textitXFST operations used for the implementation of the *x-vario* system. The explanations of the operations below were taken from the *XFST* manual on the WWW site mentioned above. Some information concerning their use in the *x-vario* system is added.

apply down < *source-file*	executes the composition of the input string with the upper tape of the top network on the stack and extracts the result from the lower tape. In the implementation of *x-vario*, *apply down* was used to apply the defined transducers to the entries of the *Bielefeld Lexical Database*.
compose net	replaces the stack with the composition of all networks currently on the stack. If the stack consists of two networks, defined as A and B, with A on top of the stack, the result is equivalent to compiling the regular expression [A .o. B] (for the syntax of regular expressions see [Kar97]). Rules which have to be applied sequentially were combined with *compose net* in the present implementation.
define *symbol regular-expression*	binds the given symbol to a network compiled from the regular expression. Every subsequent occurrence of the symbol in a regular expression is interpreted as the name of the corresponding language or relation. The phoneme classes (presented below) were defined as regular languages using this command.
invert net	exchanges the two sides of the top network on the stack. If the network is defined as A, the result is equivalent to compiling the regular expression [A.i]. (for the syntax of regular expressions see [Kar97]). This operation was used to generate the second file of the Phoneme-to-Features-Converter.

load defined *source-file*

restores the definitions from a binary source file. The networks are assigned to the chosen names directly, without pushing them onto the stack. The file must be in binary format, as produced by the 'save defined' command. The command *load defined* was used to work with the defined classes of phonemes presented below.

load stack < *source-file*

reads the networks in the given file, and pushes them onto the the stack. The file must be in binary format, as produced by the 'save stack' command.

one-plus net

concatenates the top network any number of times with itself. If the network is defined as A, the result is equivalent to compiling the regular expression [A+]. This operation was used in the implementation of some of the rules, as described below.

print net

prints a text representation of the top network on the stack (by default) or the given definition. The output is directed to a file by adding a greater-than symbol, >, and a name of a target file to the end of the command.

print size

prints the size of the top network on the stack (by default) or the given definition.

print words

displays the paths in the top network or in the network assigned to the defined symbol. If a number is given, only that many paths are printed, otherwise every word is printed but not looping on any path more than once. The output can be directed to a file by adding a greater-than symbol, >, and a name of a target file to the end of the command.

read prolog < *source-file*

reads the network(s) contained in the file and pushes them onto the stack. The file must be in the format produced by the 'write prolog' command. This command was used to implement the Phoneme-to-Features-Converter.

read regex *regular expression*

creates a network from a regular expression and pushes it onto the stack. The input expression must be terminated by a semicolon. This command was used to implement the individual rules.

union net	replaces the stack with the union of all networks currently on the stack. This operation was used to assemble individual rules in one single transducer.
write prolog > *target-file*	writes the network as Prolog clauses.

4 The architecture of the *x-vario* system

The *x-vario* system is composed of four single finite state transducers. Each of the transducers consists of the composition of three single transducers. The first one is the so-called *converter*, which converts the phonetic transcription of the entries into a string of phonemic feature bundles, which constitute the input for the *rules automaton*. The rules are applied to this string of phonemic feature bundles linewise, and alter this set where required. The output of this process is a certain number of strings of phonemic features bundles, each one of which represents a speech variant of the input string. Finally, the speech variants are transduced into their corresponding phonetic transcription by the *inverter*. The final output of *x-vario* is generated by all four transducers. Using Unix utilities like , the output of the different transducers is combined in one single text file. The code of the *x-vario* script and of the awk-scripts can be found in the appendix.

There are four different large rule transducers, which were compiled using different rule combinations. The motivation for the building of four different rule transducers was not a linguistic one. Actually, the initial intention was to build one single rule transducer containing the combination of all rules applicable. But because of the big size of the binary files, the rules were distributed over four different transducers. Another reason for this distribution was the size of the rule files themselves.

Each one of the rule transducers was first composed by *union building* of some rules. The resulting transducer was then *composed* with the converter and the inverter, resulting in the final version of each rule transducer.

Here some data concerning each of the rule transducers:

4.1 testregel01to11.fst

File length: 12200411 bytes

```
xfst[0]: load rule01.fst
xfst[1]: load rule02.fst
xfst[2]: load rule03.fst
xfst[3]: load 1+rule04.fst
xfst[4]: load rule05.fst
xfst[5]: union
1229 states, 52115 arcs, Circular.
xfst[1]: save regel01to05.fst
```

```
xfst[0]: load 1+rule10.fst
xfst[1]: load 1+rule09.fst
xfst[2]: load regel08.fst
xfst[3]: load rule07.fst
xfst[4]: load rule06.fst
xfst[5]: union
105 states, 5107 arcs, Circular.
xfst[1]: save regel06to10.fst
```

```
xfst[1]: load regel01to05.fst
xfst[1]: load union06to11.fst
xfst[2]: union
58410 states, 4689875 arcs, Circular.
xfst[1]: save regel01to11.fst
```

```
xfst[1]: load regel01to11.fst
xfst[1]: load ../Converter/invert.fst
xfst[2]: turn
xfst[2]: load ../Converter/convert.fst
xfst[3]: compose
31275 states, 3217725 arcs, Circular.
xfst[1]: save testregel01to11.fst
```

4.2 testregel12to20.fst

File length: 7452 bytes

```
xfst[0]: load rule12.fst
xfst[1]: load rule13.fst
xfst[2]: load rule14.fst
xfst[3]: load rule16.fst
xfst[4]: load rule17.fst
xfst[5]: load regel18.fst
xfst[6]: load rule19.fst
xfst[7]: load rule20.fst
xfst[8]: union
34 states, 1294 arcs, Circular.
xfst[1]: save regel12to20.fst
```

```
xfst[0]: load regel12to20.fst
xfst[1]: load ../Converter/invert.fst
```

```
xfst[2]: turn
xfst[2]: load ../Converter/convert.fst
xfst[3]: compose
34 states, 1478 arcs, Circular.
xfst[1]: save testregel12to20.fst
```

4.3 testregel21to23+27+29to30.fst

File length: 1496640 bytes

```
xfst[0]: load rule21.fst
xfst[1]: load rule22.fst
xfst[2]: load rule23.fst
xfst[3]: load regel27.fst
xfst[4]: load rule29.fst
xfst[5]: load rule30.fst
xfst[6]: union
6007 states, 357794 arcs, Circular.
xfst[1]: save testregel21to23+27+29to30.fst
```

```
xfst[0]: load testregel21to23+27+29to30.fst
xfst[1]: load ../Converter/invert.fst
xfst[2]: turn
xfst[2]: load ../Converter/convert.fst
xfst[3]: compose
6008 states, 416973 arcs, Circular.
xfst[1]: save testregel21to23+27+29to30.fst
```

4.4 testregel31+33to34+36+24.fst

File length: 11991564 bytes

```
xfst[0]: load rule31.fst
xfst[1]: load rule33.fst
xfst[2]: load rule34.fst
xfst[3]: load rule36.fst
xfst[4]: load rule24.fst
xfst[5]: union
98257 states, 4919245 arcs, Circular.
xfst[1]: save regel31+33to34+36+24.fst
```

```
xfst[1]: load regel31+33to34+36+24.fst
xfst[1]: load ../Converter/inv4.fst
xfst[2]: turn
xfst[2]: load ../Converter/conv4.fst
xfst[3]: compose
45080 states, 2709456 arcs, Circular.
xfst[1]: save testregel31+33to34+36+24.fst
```

Only rules applicable at the word level were implemented. The rules applicable at word boundaries were not considered for this implementation.

5 The Phoneme-to-Features-Converter

The Phoneme-to-Features-Converter basically consists of two finite state transducers: *convert.fst* and *invert.fst*. The first one of the transducers transduces a string of phonemes into the corresponding string of phonetic features. The second one is its inversion, which was automatically generated, and it transduces a string of phonetic features back into a string of phonemes. The transducing information is based on the SAMPA conventions for German. It was implemented using the Prolog interface of the *XFST* system. The Prolog code for *convert.pl* can be found in the appendix.

6 The rules

The rules listed below were taken from Kirchhoff's "Two-level Modelling of Speech Variant Rules" ([Kir95]). The numbering of the rules was maintained in order to keep the reference to the definitions of the rules simple. In the cited report we find detailed information about the individual rules as well as some words about the motivation for them. Any change in the present implementaion of the rules will be explicitly stated. The rules were implemented using the XFST-command *read regex*. This way, one regular expression had to be defined for each individual rule and subrule. The regular expressions defined are explicitly specified below.

Following the explanation of the implementation of each rule, information about its size (as a network) and storage is given. This information can be obtained using the XFSTcommands *print size* and *print file-length*.

The rules can be found in the directory: XFST/SOFTWARE/Rules/ in the Bielefeld VM archive.

6.1 Naming Conventions for the Rules

Each rule file name is given the extension **.fst** including the subrules. The numbering of the rules was directly taken over from Kirchhoff's report ([Kir95]), their complete naming convention follows this schema:

ruleX.fst

where X stands for the rule number in the cited report.

The subrules were named after the following schema:

rX_YPhoneme(s).fst

where X stands for the rule number in the cited report, Y is an internal counter for the subrules of an individual rule, reflecting the order in which the subrules are to be composed. Phoneme(s) stands for the phoneme class(es) affected, or, if two phonemes are given, for the substition of the first one by the second one. For example, `r35_3gk.fst` stands for the third subrule of the rule `rule35.fst`, where 'g' is substituted by 'k'. The context specified in the rules is not reflected in the file names.

Names like:

regelX.fst

stand for rules which were delibaretely implemented in a way different from Kirchhoff's formulation (for reasons stemming from the processing of the *Bielefeld Lexical Database*). The majority of the rules were implemented exactly as stated in the reported cite above.

6.2 Definition of the phoneme classes

For the implementation of the rules, phoneme classes had to be defined. Their definition here has no immediate linguistic motivation, it was driven by the definition of the rules themselves. Only classes that were needed in at least one rule were defined.

The classes were defined as sets of phonemes (i.e. a formal language), using the command *define*. The information which phonemes are contained in each class can be obtained using the command *print words*, and is explicitly stated in the following.

The defined classes can be found in the directory: XFST/SOFTWARE/Rules/Classes/.

Here is the implementation of the individual classes, in alphabetical sequence:

- **ASchwa**

  2 states, 1 arc, 1 path.
  File length: 227 bytes

  Phonemes contained in this class:

  `(+voice,vowel,[low,central])`

- **Apical**

2 states, 6 arcs, 6 paths.
File length: 439 bytes

Phonemes contained in this class:

```
(+voice,+cont,apical)
(+voice,nasal,apical)
(-voice,affr,apical)
(-voice,+cont,apical)
(+voice,-cont,apical)
(-voice,-cont,apical)
```

- **Back**

  2 states, 4 arcs, 4 paths.
  File length: 339 bytes

  Phonemes contained in this class:

```
(-voice,-cont,velar)
(+voice,-cont,velar)
(-voice,+cont,velar)
(+voice,+cont,uvular)
```

- **BackVowel**

  2 states, 8 arcs, 8 paths.
  File length: 783 bytes

  Phonemes contained in this class:

```
(+voice,vowel,[high,back,+round])
(+voice,vowel,[long,high,back,+round])
(+voice,vowel,[high,back,central,+round])
(+voice,vowel,[low,back,-round])
(+voice,vowel,[long,low,back,-round])
(+voice,vowel,[mid,back,central,+round])
(+voice,vowel,[mid,back,+round])
(+voice,vowel,[long,mid,back,+round])
```

- **Central**

  2 states, 7 arcs, 7 paths.
  File length: 719 bytes

  Phonemes contained in this class:

```
(+voice,vowel,[high,front,central,-round])
(+voice,vowel,[high,front,central,+round])
(+voice,vowel,[high,back,central,+round])
(+voice,vowel,[low,central])
(+voice,vowel,[mid,central])
(+voice,vowel,[mid,back,central,+round])
(+voice,vowel,[mid,front,central,+round])
```

- **Consonant**

  2 states, 1 arc, 1 path.
  File length: 1911 bytes

  The class **Consonant** was defined with the following definition:

  ```
  xfst[0]: define Consonant [ \Vowel ];
  ```

  where **Vowel** comprises all the vowels defined in the class named this way *and* the words of the class **PrimStress**. The words stipulated by this class cannot be listed, as this class is implemented as the negation of a given set (**Vowel**).

- **ContMinus**

  2 states, 6 arcs, 6 paths.
  File length: 443 bytes

  Phonemes contained in this class:

  ```
  (-voice,-cont,labial)
  (-voice,-cont,apical)
  (-voice,-cont,velar)
  (+voice,-cont,velar)
  (+voice,-cont,apical)
  (+voice,-cont,labial)
  ```

- **ContPlus**

  2 states, 11 arcs, 11 paths.
  File length: 679 bytes

  Phonemes contained in this class:

  ```
  (-voice,+cont,labial)
  (+voice,+cont,labial)
  (-voice,+cont,apical)
  (+voice,+cont,apical)
  (+voice,+cont,palatal)
  (-voice,+cont,palatal)
  (+voice,+cont,palato)
  ```

```
(-voice,+cont,palato)
(+voice,+cont,uvular)
(-voice,+cont,glottal)
(-voice,+cont,velar)
```

- **Diphthong**

  2 states, 24 arcs, 24 paths.
  File length: 6072 bytes

  This class is actually implemented through 9 different nets. They can thus be accessed like one single class.

  Phonemes contained in this class:

```
(+voice,vowel,[long,low,back,-round])
(+voice,vowel,[mid,central])
(+voice,vowel,[low,central])
(+voice,vowel,[high,back,central,+round])
(+voice,vowel,[long,mid,back,+round])
(+voice,vowel,[mid,back,+round])
(+voice,vowel,[mid,back,central,+round])
(+voice,vowel,[low,back,-round])
(+voice,vowel,[low,front,-round])
(+voice,vowel,[long,low,front,-round])
(+voice,vowel,[mid,front,central,+round])
(+voice,vowel,[long,high,back,+round])
(+voice,vowel,[high,back,+round])
(+voice,vowel,[high,front,central,+round])
(+voice,vowel,[long,mid,front,-round])
(+voice,vowel,[mid,front,-round])
(+voice,vowel,[long,mid,front,+round])
(+voice,vowel,[mid,front,+round])
(+voice,vowel,[high,front,+round])
(+voice,vowel,[long,high,front,+round])
(+voice,vowel,[high,front,central,-round])
(+voice,vowel,[high,front,-round])
(+voice,vowel,[long,high,front,-round])
```

- **FrontPlus**

  2 states, 2 arcs, 2 paths.
  File length: 267 bytes

  Phonemes contained in this class:

```
(-voice,+cont,palatal)
(+voice,+cont,palatal)
```

- **FrontVowel**

  2 states, 13 arcs, 13 paths.
  File length: 1193 bytes

  Phonemes contained in this class:

  ```
  (+voice,vowel,[long,high,front,-round])
  (+voice,vowel,[high,front,-round])
  (+voice,vowel,[high,front,central,-round])
  (+voice,vowel,[long,mid,front,-round])
  (+voice,vowel,[mid,front,-round])
  (+voice,vowel,[mid,front,+round])
  (+voice,vowel,[long,mid,front,+round])
  (+voice,vowel,[high,front,central,+round])
  (+voice,vowel,[high,front,+round])
  (+voice,vowel,[long,high,front,+round])
  (+voice,vowel,[mid,front,central,+round])
  (+voice,vowel,[long,low,front,-round])
  (+voice,vowel,[low,front,-round])
  ```

- **Labial**

  2 states, 6 arcs, 6 paths.
  File length: 439 bytes

  Phonemes contained in this class:

  ```
  (+voice,+cont,labial)
  (+voice,nasal,labial)
  (-voice,affr,labial)
  (-voice,+cont,labial)
  (+voice,-cont,labial)
  (-voice,-cont,labial)
  ```

- **Lateral**

  2 states, 1 arc, 1 path.
  File length: 217 bytes

  Phonemes contained in this class:

  ```
  (+voice,lateral,apical)
  ```

- **LongVowel**

  2 states, 8 arcs, 8 paths.
  File length: 811 bytes

  Phonemes contained in this class:

```
(+voice,vowel,[long,high,front,-round])
(+voice,vowel,[long,mid,front,-round])
(+voice,vowel,[long,high,front,+round])
(+voice,vowel,[long,mid,front,+round])
(+voice,vowel,[long,low,back,-round])
(+voice,vowel,[long,mid,back,+round])
(+voice,vowel,[long,high,back,+round])
(+voice,vowel,[long,low,front,-round])
```

- **LowMinus**

  2 states, 10 arcs, 10 paths.
  File length: 953 bytes

  Phonemes contained in this class:

```
(+voice,vowel,[high,front,central,+round])
(+voice,vowel,[long,mid,front,-round])
(+voice,vowel,[mid,front,-round])
(+voice,vowel,[long,mid,front,+round])
(+voice,vowel,[mid,front,+round])
(+voice,vowel,[high,front,+round])
(+voice,vowel,[long,high,front,+round])
(+voice,vowel,[high,front,central,-round])
(+voice,vowel,[high,front,-round])
(+voice,vowel,[long,high,front,-round])
```

- **MinusContPlusApi**

  2 states, 2 arcs, 2 paths.
  File length: 277 bytes

  Phonemes contained in this class:

```
(-voice,-cont,apical)
(+voice,-cont,apical)
```

- **MinusContPlusLab**

  2 states, 2 arcs, 2 paths.
  File length: 277 bytes

  Phonemes contained in this class:

```
(-voice,-cont,labial)
(+voice,-cont,labial)
```

- **MinusContPlusVel**

  2 states, 2 arcs, 2 paths.
  File length: 273 bytes

  Phonemes contained in this class:

  ```
  (-voice,-cont,velar)
  (+voice,-cont,velar)
  ```

- **MinusVoi**

  2 states, 13 arcs, 13 paths.
  File length: 763 bytes

  Phonemes contained in this class:

  ```
  (-voice,-cont,labial)
  (-voice,-cont,apical)
  (-voice,-cont,velar)
  (-voice,-cont,glottal)
  (-voice,+cont,labial)
  (-voice,+cont,glottal)
  (-voice,+cont,velar)
  (-voice,+cont,palatal)
  (-voice,+cont,palato)
  (-voice,+cont,apical)
  (-voice,affr,palato)
  (-voice,affr,apical)
  (-voice,affr,labial)
  ```

- **MinusVoiPlusCont**

  2 states, 6 arcs, 6 paths.
  File length: 463 bytes

  Phonemes contained in this class:

  ```
  (-voice,+cont,labial)
  (-voice,+cont,apical)
  (-voice,+cont,palato)
  (-voice,+cont,palatal)
  (-voice,+cont,velar)
  (-voice,+cont,glottal)
  ```

- **Nasal**

  2 states, 3 arcs, 3 paths.
  File length: 299 bytes

Phonemes contained in this class:

```
(+voice,nasal,labial)
(+voice,nasal,apical)
(+voice,nasal,velar)
```

- **Onset**

  2 states, 5 arcs, 5 paths.
  File length: 231 bytes

  This class comprises the phonemes of the class **StemInitial**, the '.' for the syllable boundary marking, and the ' ', which is used for marking canonical variants that already exist in the *Bielefeld Lexical Database*, so that these can be distinguished from the ones produced by the *x-vario* system.

  Phonemes contained in this class:

```
#
+
.#.
.
```

- **PlusLowPlusFront**

  2 states, 3 arcs, 3 paths.
  File length: 421 bytes

  Phonemes contained in this class:

```
(+voice,vowel,[low,front,-round])
(+voice,vowel,[long,low,front,-round])
(+voice,vowel,[mid,front,central,+round])
```

- **PlusVoi**

  2 states, 35 arcs, 35 paths.
  File length: 2494 bytes

  Phonemes contained in this class:

```
(+voice,lateral,apical)
(+voice,nasal,apical)
(+voice,nasal,labial)
(+voice,nasal,velar)
(+voice,vowel,[high,back,central,+round])
(+voice,vowel,[high,back,+round])
(+voice,vowel,[high,front,central,+round])
```

```
(+voice,vowel,[high,front,central,-round])
(+voice,vowel,[high,front,+round])
(+voice,vowel,[high,front,-round])
(+voice,vowel,[long,high,back,+round])
(+voice,vowel,[long,high,front,+round])
(+voice,vowel,[long,high,front,-round])
(+voice,vowel,[long,low,back,-round])
(+voice,vowel,[long,low,front,-round])
(+voice,vowel,[long,mid,back,+round])
(+voice,vowel,[long,mid,front,+round])
(+voice,vowel,[long,mid,front,-round])
(+voice,vowel,[low,back,-round])
(+voice,vowel,[low,central])
(+voice,vowel,[low,front,-round])
(+voice,vowel,[mid,back,central,+round])
(+voice,vowel,[mid,back,+round])
(+voice,vowel,[mid,central])
(+voice,vowel,[mid,front,central,+round])
(+voice,vowel,[mid,front,+round])
(+voice,vowel,[mid,front,-round])
(+voice,+cont,apical)
(+voice,+cont,labial)
(+voice,+cont,palatal)
(+voice,+cont,palato)
(+voice,+cont,uvular)
(+voice,-cont,apical)
(+voice,-cont,labial)
(+voice,-cont,velar)
```

- **PlusVoiMinusCont**

  2 states, 3 arcs, 3 paths.
  File length: 321 bytes

  Phonemes contained in this class:

  ```
  (+voice,-cont,labial)
  (+voice,-cont,apical)
  (+voice,-cont,velar)
  ```

- **PrimStress**

  2 states, 1 arc, 1 path.
  File length: 181 bytes

  Phonemes contained in this class:

  ```
  '
  ```

- **Schwa**

  2 states, 1 arc, 1 path.
  File length: 225 bytes

  Phonemes contained in this class:

  `(+voice,vowel,[mid,central])`

- **SecStress**

  2 states, 1 arc, 1 path. File length: 179 bytes

  Phonemes contained in this class:

  `,,`

- **StemBoundary**

  2 states, 4 arcs, Circular.
  File length: 192 bytes

  Phonemes contained in this class:

  ```
  #
  +
  ```

- **StemInitial**

  2 states, 3 arcs, 3 paths.
  File length: 229 bytes

  Phonemes contained in this class:

  ```
  #
  +
  .#.
  ```

- **Stress**

  2 states, 2 arcs, 2 paths.
  File length: 179 bytes

  Phonemes contained in this class:

  ```
  ,
  ,,
  ```

- **Velar**

  2 states, 5 arcs, 5 paths.
  File length: 385 bytes

  Phonemes contained in this class:

  ```
  (-voice,-cont,velar)
  (+voice,-cont,velar)
  (-voice,+cont,velar)
  (+voice,+cont,uvular)
  (+voice,nasal,velar)
  ```

- **Vowel**

  2 states, 23 arcs, 23 paths.
  File length: 1924 bytes

  Phonemes contained in this class:

  ```
  (+voice,vowel,[long,high,front,-round])
  (+voice,vowel,[high,front,-round])
  (+voice,vowel,[high,front,central,-round])
  (+voice,vowel,[long,high,front,+round])
  (+voice,vowel,[high,front,+round])
  (+voice,vowel,[mid,front,+round])
  (+voice,vowel,[long,mid,front,+round])
  (+voice,vowel,[mid,front,-round])
  (+voice,vowel,[long,mid,front,-round])
  (+voice,vowel,[high,front,central,+round])
  (+voice,vowel,[high,back,+round])
  (+voice,vowel,[long,high,back,+round])
  (+voice,vowel,[mid,front,central,+round])
  (+voice,vowel,[long,low,front,-round])
  (+voice,vowel,[low,front,-round])
  (+voice,vowel,[low,back,-round])
  (+voice,vowel,[mid,back,central,+round])
  (+voice,vowel,[mid,back,+round])
  (+voice,vowel,[long,mid,back,+round])
  (+voice,vowel,[high,back,central,+round])
  (+voice,vowel,[low,central])
  (+voice,vowel,[mid,central])
  (+voice,vowel,[long,low,back,-round])
  ```

6.3 The Implementation of the Rules

The implementation of the individual rules follows.

- **rule01.fst** *Stem-initial devoicing*

  2 states, 10 arcs, Circular.
  File length: 278 bytes

  This rule was implemented as stated by Kirchhoff.

```
regex [ "(+voice,+cont,apical)" -> "(-voice,+cont,apical)"
|| StemInitial_ ];
```

- **rule02.fst** *Reduction of affricates*

  6 states, 44 arcs, Circular.
  File length: 391 bytes

  This rule was implemented just as stated by Kirchhoff. It consists of two subrules:

  - **r2_1pf.fst** For the *reduction of /pf/*

```
regex [ "(-voice,affr,labial)" -> "(-voice,+cont,labial)"
|| StemInitial_ ];
2 states, 10 arcs, Circular.
File length:      276 bytes
```

  - **r2_2ts.fst** For the *reduction of /ts/*

```
regex [ "(-voice,affr,apical)" -> "(-voice,+cont,apical)"
|| StemInitial_ ];
2 states, 10 arcs, Circular.
File length:      276 bytes
```

- **rule03.fst** *Glottal Stop Dropping*

  2 states, 10 arcs, Circular.
  File length: 198 bytes

  This rule was implemented as stated by Kirchhoff.

```
regex [ "?" -> 0 // Onset_ ];
```

- **rule04.fst** *Shortening*

  255 states, 5351 arcs, Circular.
  File length: 16111 bytes

  This rule was implemented just as stated by Kirchhoff. It consists of eight subrules:

  - **r4_1i:.fst** For the *shortening of /i:/*

```
regex [ "(+voice,vowel,[long,high,front,-round])"
-> "(+voice,vowel,[high,front,-round])" ];
1 state, 3 arcs, Circular.
File length:      317 bytes
```

  - **r4_2y:.fst** *For the shortening of /y:/*

```
regex [ "(+voice,vowel,[long,high,front,+round])"
-> "(+voice,vowel,[high,front,+round])" ];
1 state, 3 arcs, Circular.
File length:      317 bytes
```

– **r4_3e:.fst** For the *shortening of /e:/*

```
regex [ "(+voice,vowel,[long,mid,front,-round])"
-> "(+voice,vowel,[mid,front,-round])" ];
1 state, 3 arcs, Circular.
File length:      313 bytes
```

– **r4_42:.fst** *For the shortening of /2:/*

```
regex [ "(+voice,vowel,[long,mid,front,+round])"
-> "(+voice,vowel,[mid,front,+round])" ];
1 state, 3 arcs, Circular.
File length:      313 bytes
```

– **r4_5E:.fst** *For the shortening of /E:/*

```
regex [ "(+voice,vowel,[long,low,front,-round])"
-> "(+voice,vowel,[low,front,-round])" ];
1 state, 3 arcs, Circular.
File length:      313 bytes
```

– **r4_6u:.fst** *For the shortening of /u:/*

```
regex [ "(+voice,vowel,[long,high,back,+round])"
-> "(+voice,vowel,[high,back,+round])" ];
1 state, 3 arcs, Circular.
File length:      313 bytes
```

– **r4_7o:.fst** *For the shortening of /o:/*

```
regex [ "(+voice,vowel,[long,mid,back,+round])"
-> "(+voice,vowel,[mid,back,+round])" ];
1 state, 3 arcs, Circular.
File length:      309 bytes
```

– **r4_8a:.fst** *For the shortening of /a:/*

```
regex [ "(+voice,vowel,[long,low,back,-round])"
-> "(+voice,vowel,[low,back,-round])" ];
1 state, 3 arcs, Circular.
File length:      309 bytes
```

rule04.fst was then produced through union building of all eight subrules. It was implemented one-plus (**1+rule04.fst**):

```
xfst[0]: load rule04.fst
xfst[1]: one-plus
1 state, 25 arcs, Circular.
File length:      1379 bytes
```

- **regel05.fst** *Laxing*

  1 state, 3 arcs, Circular.
  File length: 311 bytes

  This rule was implemented only for the substituion /e:/ → /E/, the other cases stated by Kirchhoff were not considered.

  ```
  regex [ "(+voice,vowel,[long,mid,front,-round])"
  -> "(+voice,vowel,[low,front,-round])" ];
  ```

- **rule06.fst** *E: - e: Alternation*

  6 states, 124 arcs, Circular.
  File length: 2139 bytes

  This rule was implemented through the union building of two subrules, **r6_1E:.fst** and **r6_2E.fst**. The first one corresponds to the definition of the rule given by Kirchhoff in her rule 6. The second subrule is a completion of it, as stated in the explanatory text of her rule 11. The second subrule does not apply within a diphthong.

  – **r6_1E:.fst** For the */E:/→/e:/ Alternation*

  ```
   regex [ "(+voice,vowel,[long,low,front,-round])" \\
  -> "(+voice,vowel,[long,mid,front,-round])" ];
  1 state, 3 arcs, Circular.
  File length:      323 bytes
  ```

  – **r6_2E.fst** For the */E/→/e/ Alternation* (stated in rule 11)

  ```
  regex [ "(+voice,vowel,[low,front,-round])"\\
   -> "(+voice,vowel,[mid,front,-round])"\\
  || _Vowel ];
  3 states, 50 arcs, Circular.
  ```

- **rule07.fst** *I-Rounding*

  4 states, 14 arcs, Circular.
  File length: 417 bytes

  This rule was specified as stated in Kirchhoff's report [Kir95]:

  ```
  regex [ "(+voice,vowel,[high,front,central,-round])"
  -> "(+voice,vowel,[high,front,central,+round])"
  || ?_"(+voice,vowel,[low,central])" ];
  ```

- **rule08.fst** *Schwa-reduction*

  4 states, 112 arcs, Circular.
  File length: 2176 bytes

  This rule was specified as stated in Kirchhoff's report [Kir95]:

```
regex [ Vowel - Central -> "(+voice,vowel,[mid,central])"
|| \Stress_\Vowel ];
```

This rule was implemented one-plus (**1+rule08.fst**), so that it can change the input string at any potentially possible position:

```
xfst[0]: load rule08.fst
xfst[1]: one-plus
3 states, 72 arcs, Circular.
File length:      2112 bytes
```

- **rule09.fst** *Diphthong reduction: 6-dropping*

  4 states, 74 arcs, Circular.
  File length: 2061 bytes

  This rule was specified as stated in Kirchhoff's report [Kir95]:

```
regex [ "(+voice,vowel,[low,central])" -> 0
|| Vowel_? ];
```

  This rule was implemented one-plus (**1+rule09.fst**), so that it can change the input string at any potentially possible position:

```
xfst[0]: load rule09.fst
xfst[1]: one-plus
3 states, 74 arcs, Circular.
File length:      2061 bytes
```

- **rule10.fst** *Diphthong reduction: reduction to schwa*

  4 states, 121 arcs, Circular.
  File length: 2213 bytes

  This rule was implemented through the composition of two subrules: the first one, **r10_16.fst**, for the substituion /6/ → /@/, and the second one, **r10_2del.fst**, for the deletion of the vowel.

  - **r10_16.fst** for the *substitution /6/ → /@/*

```
regex [ "(+voice,vowel,[low,central])" -> "(+voice,vowel,[mid,central])" \\
|| \PrimStress Vowel_ ];
3 states, 75 arcs, Circular.
File length:      2069 bytes
```

  - **r10_2del.fst** for the *deletion of the vowel*

```
regex [ Vowel -> 0 || \PrimStress_"(+voice,vowel,[mid,central])" ];
4 states, 121 arcs, Circular.
File length:      2250 bytes
```

  This rule was implemented one-plus (**1+rule10.fst**), so that it can process the input string at any potentially possible position:

```
xfst[0]: load rule10.fst
xfst[1]: one-plus
4 states, 123 arcs, Circular.
File length:      2213 bytes
```

- **rule11.fst** *Diphthong reduction: reduction to a-schwa*

  4 states, 29 arcs, Circular.
  File length: 580 bytes

  For the implementation of this rule, a special class was created, *ReductionASchwa*, which containing exactly those phonemes that are affected by the rule. They are listed in the cited report.

  ```
  regex [ ReductionASchwa -> 0 // \PrimStress_ASchwa ];
  ```

- **rule12.fst** *Nasal assimilation*

  4 states, 24 arcs, Circular.
  File length: 698 bytes

  This rule was implemented through union building over two subrules: **r12_1nm.fst**, for the labial nasals and **r12_2nN.fst**, for the velar nasals.

  - **r12_1nm.fst** for the labial nasals

    ```
    regex [ "(+voice,nasal,apical)"
    -> "(+voice,nasal,labial)" // _Labial.#. ];
    5 states, 32 arcs, Circular.
    File length:      535 bytes
    ```

  - **r12_2nN.fst** for the velar nasals

    ```
    regex [ "(+voice,nasal,apical)"
    -> "(+voice,nasal,velar)" || _Back.#. ];
    5 states, 28 arcs, Circular.
    File length:      474 bytes
    ```

- **rule13.fst** *Plosive epenthesis*

  6 states, 61 arcs, Circular.
  File length: 780 bytes

  This rule was implemented through union building of three subrules: the first one, **r13_1mpf.fst**, for the labial plosives, the second one, **r13_2nts.fst**, for the apical plosives and the third one, **r13_3Nks.fst**, for the velar plosives .

  - **r13_1mpf.fst** for the labial plosives

    ```
    regex [ 0 -> "(-voice,-cont,labial)" // m_MinusVoiPlusCont.#. ];
    4 states, 25 arcs, Circular.
    File length:      571 bytes
    ```

  - **r13_2nts.fst** for the apical plosives

```
regex [ 0 -> "(-voice,-cont,apical)" // n_MinusVoiPlusCont.#. ];
4 states, 25 arcs, Circular.
File length:      569 bytes
```

  - **r13_3Nks.fst** for the velar plosives

```
regex [ 0 -> "(-voice,-cont,velar)" // N_MinusVoiPlusCont.#. ];
4 states, 25 arcs, Circular.
File length:      565 bytes
```

- **rule14.fst** *Plosive Deletion*

  5 states, 90 arcs, Circular.
  File length: 1282 bytes

  Rule 14 was implemented through union building of two subrules. The first one, **r14_1all.fst**, the most general one, stands for the deletion of all plosives between a nasal and a continuant consonant. The second one, **r14_2nts.fst**, is for the specific case /nts/→/ns/. The extra specification of this particular case was necessary, as /ts/ is declared as one phoneme and is therefore recognized by the *XFST* system as an atom. However, the union of both these rules is equivalent to the declaration of the rule in the report [Kir95]

  - **r14_1all.fst** the general rule

```
regex [ ContMinus -> 0 // Nasal_ContPlus.#. ];
6 states, 101 arcs, Circular.
File length:      1292 bytes
```

  - **r14_2nts.fst** for the specific case /nts/→/ns/

```
regex [ "(-voice,affr,apical)" -> "(-voice,+cont,apical)"
|| "(+voice,nasal,apical)"_.#. ];
4 states, 13 arcs, Circular.
File length:      316 bytes
```

Rule 15 was not implemented in the final version of the automaton.

- **rule16.fst** *R-Alternation*

  6 states, 104 arcs, Circular.
  File length: 2176 bytes

```
regex [ "(+voice,vowel,[low,central])"
-> "(+voice,+cont,uvular)"
// Vowel_\Vowel.#. ];
```

- **rule17.fst** *Velar fricative weakening*

  7 states, 137 arcs, Circular.
  File length: 2302 bytes

```
regex [ "(-voice,+cont,velar)"
-> "(-voice,+cont,glottal)"
// ?_\Vowel.#. ];
```

- **regel18.fst** *L-vocalisation*

  4 states, 25 arcs, Circular.
  File length: 669 bytes

  ```
  regex [ "(+voice,lateral,apical)"
  -> "(+voice,vowel,[high,front,central,-round])"
  || PlusLowPlusFront_FrontPlus ];
  ```

- **rule19.fst** *Schwa-deletion*

  6 states, 130 arcs, Circular.
  File length: 3251 bytes

  This rule was implemented stem-final, as stated in Kirchhoff's report [Kir95].

  ```
  regex [ "(+voice,vowel,[mid,central])" -> 0
  || Consonant_Nasal.#. | Lateral.#. ];
  ```

- **rule20.fst** *Schwa-deletion and progressive place assimilation*

  12 states, 333 arcs, Circular.
  File length: 3251 bytes

  Rule 20 was implemented through composition of two subrules: **r20_1Schwa.fst** for the Schwa-deletion and **r20_2Assim.fst** for the progressive place assimilation.

  – **r20_1Schwa.fst** *Schwa-deletion*

    ```
    regex [ "(+voice,vowel,[mid,central])" -> 0
    || Consonant_Nasal.#. ];
    6 states, 125 arcs, Circular.
    File length:      2341 bytes
    ```

  – **r20_2Assim.fst** *Progressive place assimilation*

    4 states, 41 arcs, Circular.
    File length: 732 bytes

    r20_2Assim.fst was produced through union building over two subrules:

    * **r20_2aLabial.fst** for the labials

      ```
      regex [ "(+voice,nasal,apical)"
      -> "(+voice,nasal,labial)" || Labial_.#. ];
      4 states, 25 arcs, Circular.
      File length:      520 bytes
      ```

    * **r20_2bVelar.fst** for the velars

```
regex [ "(+voice,nasal,apical)"
-> "(+voice,nasal,velar)" || Velar_.#. ];
4 states, 22 arcs, Circular.
File length:      463 bytes
```

- **rule21.fst** *Schwa-deletion and regressive place assimilation* 11 states, 48 arcs, Circular.
  File length: 401 bytes

  Rule 21 was implemented through composition of two subrules: **r21_1Schwa.fst** for the Schwa-deletion and **r21_2Assim.fst** for the regressive place assimilation.

  - **r21_1Schwa.fst** *Schwa-deletion*

    ```
     regex [ "(+voice,vowel,[mid,central])" -> 0 || Nasal_Nasal.#. ];
    6 states, 25 arcs, Circular.
    File length:      397 bytes
    ```

  - **r21_2Assim.fst** *Schwa-deletion*

    ```
    regex [ "(+voice,nasal,apical)" -> "(+voice,nasal,labial)"
    || _"(+voice,nasal,labial)".#. ];
    5 states, 13 arcs, Circular.
    File length:      272 bytes
    ```

  rule21.fst was then produced through composition over the two subrules.

- **rule22.fst** *Schwa-deletion and consonant assimilation*

  12 states, 87 arcs, Circular.
  File length: 949 bytes

  Rule 22 was implemented with the union building of two rules, **rule22a.fst** and **rule22b.fst**, as stated in Kirchhoff's report [Kir95]. The first rule is for the deletion of one [+voi,-cont] consonant, the second one for the deletion of the /n/. The rule was implemented stem-final.

  - **rule22a.fst** *Schwa-deletion and consonant assimilation*
    15 states, 173 arcs, Circular.
    File length: 1125 bytes
    Rule 22(a) was implemented through composition of three subrules: **r22a_1Schwa.fst**, **r22a_2Assim.fst** and **r22a_3Consonant.fst**.

    * **r22a_1Schwa.fst** for the Schwa-deletion

      ```
      regex [ "(+voice,vowel,[mid,central])"
      -> 0 || PlusVoiMinusCont_Nasal.#. ];
      6 states, 36 arcs, Circular.
      File length:      552 bytes
      ```

    * **r22a_2Assim.fst** for the assimilation of the consonant
      4 states, 41 arcs, Circular.
      File length: 732 bytes
      This subrule was implemented through union bulding of two subrules: **r22a_2Labial.fst** and **r22a_2Velar.fst**.

· **r22a_2Labial.fst** for the assimilation of /n/ to a labial consonant

```
regex [ "(+voice,nasal,apical)"
-> "(+voice,nasal,labial)" || Labial_.#. ];
4 states, 25 arcs, Circular.
File length:      520 bytes
```

· **r22a_2Velar.fst** for the assimilation of /n/ to a velar consonant

```
regex  [ "(+voice,nasal,apical)"
 -> "(+voice,nasal,velar)" || Velar_.#. ];
4 states, 22 arcs, Circular.
File length:      520 bytes
```

Through union building **r22_2Assim** was created.

∗ **r22a_3Consonant.fst** for the consonant deletion

```
regex [ PlusVoiMinusCont -> 0 || _Nasal.#. ];
5 states, 33 arcs, Circular.
File length:      492 bytes
```

Through composition of the three subrules rule22a.fst was produced.

– **rule22b.fst** for the /n/-deletion
7 states, 23 arcs, Circular.
File length: 312 bytes

rule 22(b) was produced through the compostion of two subrules: **r22b_1Schwa.fst** for the Schwa-deletion and **r22b_2n.fst** for the deletion of the /n/ before a /m/.

∗ **r22b_1Schwa.fst** for the *Schwa-deletion*

```
regex [ "(+voice,vowel,[mid,central])" -> 0
|| "(+voice,nasal,apical)"_"(+voice,nasal,labial)".#. ];
6 states, 18 arcs, Circular.
File length:      340 bytes
```

∗ **r22b_2n.fst** for the */n/-deletion*

```
regex [ "(+voice,nasal,apical)" -> 0
|| _"(+voice,nasal,labial)".#. ];
5 states, 13 arcs, Circular.
File length:      272 bytes
```

Through composition of the two subrules **rule22b.fst** was produced.

rule22.fst was then produced through union building of **rule22a.fst** and **rule22b.fst**.

• **rule23.fst** *Schwa-deletion and consonant assimilation*

16 states, 147 arcs, Circular.
File length: 919 bytes

Rule 23 was implemented with the composition of three subrules: the first one for the schwa-deletion, the second one for the assimilation of the nasal and the third one for the nasalisation of the consonant.

– **r23_1Schwa.fst** for the schwa-deletion

```
regex [ "(+voice,vowel,[mid,central])" -> 0
|| ContMinus_Nasal.#. ];
6 states, 48 arcs, Circular.
File length:      709 bytes
```

- **r23_2Assim.fst** for the assimilation of the nasal

  5 states, 46 arcs, Circular.
  File length: 618 bytes

  This subrule was built through union building over three subrules:

  - **r23_2N.fst** for the schwa-deletion

    ```
    regex [ Nasal -> "(+voice,nasal,velar)"
    || MinusContPlusVel_.#. ];
    4 states, 20 arcs, Circular.
    File length:      418 bytes
    ```

  - **r23_2n.fst** for the schwa-deletion

    ```
    regex [ Nasal -> "(+voice,nasal,apical)"
    || MinusContPlusApi_.#. ];
    4 states, 20 arcs, Circular.
    File length:      422 bytes
    ```

  - **r23_2m.fst** for the schwa-deletion

    ```
    regex [ Nasal -> "(+voice,nasal,labial)"
    || MinusContPlusLab_.#. ];
    4 states, 20 arcs, Circular.
    File length:      422 bytes
    ```

- **r23_3Nasal.fst** for the assimilation of the nasal

  5 states, 19 arcs, Circular. File length: 564 bytes

  This subrule was built through union building over three subrules:

  - **r23_3aLabial.fst** for the schwa-deletion

  ```
   regex [ MinusContPlusLab ->  "(+voice,nasal,labial)"
  || _"(+voice,nasal,labial)".#. ];
  5 states, 19 arcs, Circular.
  File length:      331 bytes
  ```

  - **r23_3bApical.fst** for the schwa-deletion

  ```
  regex [ MinusContPlusApi -> "(+voice,nasal,apical)"
  || _"(+voice,nasal,apical)".#. ];
  5 states, 19 arcs, Circular.
  File length:      331 bytes
  ```

– **r23_3cVelar.fst** for the schwa-deletion

```
regex [ MinusContPlusVel -> "(+voice,nasal,velar)"
|| _"(+voice,nasal,velar)".#. ];
5 states, 19 arcs, Circular.
File length:      325 bytes
```

The subrules **r23_2Assim.fst** and **r23_3Nasal.fst** were each built through union building of the corresponding subrules listed above. **rule23.fst** was then implemented with composition of the three subrules, **r23_1Schwa.fst**, **r23_2Assim.fst** and **r23_3Nasal.fst** in this order.

- **rule24.fst** *Voicing assimilation*

  709 states, 30634 arcs, Circular.
  File length: 107534 bytes

  Rule 24 was implemented through union building of seven subrules, which follow:

  – **r24_1pb.fst** for the substitution /p/ → /b/

```
regex [ "(-voice,-cont,labial)"
-> "(+voice,-cont,labial)" || PlusVoi_PlusVoi ];
4 states, 112 arcs, Circular.
File length:      2761 bytes
```

  – **r24_2td.fst** for the substitution /t/ → /d/

```
regex ["(-voice,-cont,apical)"
-> "(+voice,-cont,apical)" || PlusVoi_PlusVoi ];
4 states, 112 arcs, Circular.
File length:      2761 bytes
```

  – **r24_3kg.fst** for the substitution /k/ → /g/

```
regex [ "(-voice,-cont,velar)"
-> "(+voice,-cont,velar)" || PlusVoi_PlusVoi ];
4 states, 112 arcs, Circular.
File length:      2759 bytes
```

  – **r24_4fv.fst** for the substitution /f/ → /v/

```
regex [ "(-voice,+cont,labial)"
-> "(+voice,+cont,labial)" || PlusVoi_PlusVoi ];
4 states, 112 arcs, Circular.
File length:      2761 bytes
```

  – **r24_5sz.fst** for the substitution /s/ → /z/

```
regex [ "(-voice,+cont,apical)"
-> "(+voice,+cont,apical)" || PlusVoi_PlusVoi ];
4 states, 112 arcs, Circular.
File length:      2761 bytes
```

– **r24_6SZ.fst** for the substitution /S/ → /Z/

```
regex [ "(-voice,+cont,palato)"
-> "(+voice,+cont,palato)" || PlusVoi_PlusVoi ];
4 states, 112 arcs, Circular.
File length:      2761 bytes
```

– **r24_7Cj.fst** for the substitution /C/ → /j/

```
regex ["(-voice,+cont,palatal)"
-> "(+voice,+cont,palatal)" || PlusVoi_PlusVoi ];
4 states, 112 arcs, Circular.
File length:      2763 bytes
```

Rule24.fst was then built through union buiding over all seven subrules.

Rule 25 (*Laxing of Constriction: labials*) and rule 26 (*Laxing of Constriction: velars*) were not implemented.

- **rule27.fst** *Cross-syllable plosive assimilation/deletion*

  4 states, 69 arcs, Circular.
  File length: 1224 bytes

  From rule 27 only the subrule (c) (as described in [Kir95]) was implemented (*Deletion before fricatives*). This subrule is described below:

  – **r27_c.fst**

```
regex [ ContMinus -> 0 || Nasal_ContPlus ];
```

Rule 28 (*Cluster simplification*) was not implemented.

- **rule29.fst** *Schwa-Lowering*

  4 states, 85 arcs, Circular.
  File length: 2153 bytes

```
regex [ "(+voice,vowel,[mid,central])"
-> "(+voice,vowel,[low,central])"
|| Consonant_"(+voice,+cont,uvular)" ];
```

- **rule30.fst** *Stem-final mixed junction*

  891 states, 24845 arcs, Circular.
  File length: 83721 bytes

  This rule was implemented here consisting of three subrules, which were then composed: the first rule is the /r/ deletion, the second rule is the substitution of /@/ into /6/ an the third is the substitution of the long vowels by central ones. This last subrule is itself built through union building of five subrules.

  - **r30_1r.fst** for the /r/ deletion

```
regex [ "(+voice,+cont,uvular)" -> 0
// LongVowel_Schwa "(+voice,nasal,apical)" .#. ];
8 states, 63 arcs, Circular.
File length:      1060 bytes
```

  - **r30_2@.fst** for the substitution of /@/ by /6/

```
regex [ "(+voice,vowel,[mid,central])" -> "(+voice,vowel,[low,central])"
|| LongVowel_"(+voice,nasal,apical)" ];
4 states, 37 arcs, Circular.
File length:      1025 bytes
```

  - **r30_3ai:.fst** for the substitution of /i:/ by /I/

```
regex [ "(+voice,vowel,[long,high,front,-round])" ->
"(+voice,vowel,[high,front,central,-round])"
|| Consonant_"(+voice,vowel,[low,central])" "(+voice,nasal,apical)" ];
6 states, 114 arcs, Circular.
File length:      2221 bytes
```

  - **r30_3by:.fst** for the substituion of /y:/ by /Y/

```
regex [ "(+voice,vowel,[long,high,front,+round])" ->
"(+voice,vowel,[high,front,central,+round])"
|| Consonant_"(+voice,vowel,[low,central])" "(+voice,nasal,apical)" ];
6 states, 114 arcs, Circular.
File length:      2221 bytes
```

  - **r30_3cu:.fst** for the substituion of /u:/ by /U/

```
regex [ "(+voice,vowel,[long,high,back,+round])" ->
"(+voice,vowel,[high,back,central,+round])"
|| Consonant_"(+voice,vowel,[low,central])" "(+voice,nasal,apical)" ];
6 states, 114 arcs, Circular.
File length:      2221 bytes
```

  - **r30_3do:.fst** for the substituion of /o:/ by /O/

```
regex [ "(+voice,vowel,[long,mid,back,+round])" ->
"(+voice,vowel,[mid,back,central,+round])"
|| Consonant_"(+voice,vowel,[low,central])" "(+voice,nasal,apical)" ];
6 states, 114 arcs, Circular.
File length:      2221 bytes
```

– **r30_3eE:.fst** for the substitution of /E:/ by /E/

```
regex [ "(+voice,vowel,[long,low,front,-round])" ->
"(+voice,vowel,[low,front,-round])"
|| Consonant_"(+voice,vowel,[low,central])" "(+voice,nasal,apical)" ];
6 states, 114 arcs, Circular.
File length:      2221 bytes
```

– **r30_3f2:.fst** for the substitution of /2:/ by /9/

```
regex [ "(+voice,vowel,[long,mid,front,+round])"
-> "(+voice,vowel,[mid,front,central,+round])"
|| Consonant_"(+voice,vowel,[low,central])" "(+voice,nasal,apical)" ];
6 states, 114 arcs, Circular.
File length:      2221 bytes
```

The rule **r30_3central.fst** was generated through union building of the rules **r30_3(...).fst** defined above.

522 states, 14478 arcs, Circular.
File length: 48258 bytes

rule30.fst was then implemented composing the rules **r30_1r.fst**, **r30_2@.fst** and **r30_3central.fst**.

- **rule31.fst** *Stem-final cluster simplification*

  9 states, 43 arcs, Circular.
  File length: 360 bytes

  In Kirchhoff's report, this rule serves the schwa-deletion and cluster reduction taking place in one pass. It was implemented there like that because the Two-Level formalism does not permit rules to apply sequentially. But working with finite state transducers permits us to compose two defined rules, forcing them to apply sequentially. This is the reason why rule31.fst was implemented here consisting of two subrules: the first one for the cluster reduction and the second one for the schwa-deletion. The order of application of the subrules doesn't actually matter in this case.

  – **r31_1t.fst** for the cluster reduction (/t/ deletion)

```
regex [ "(-voice,-cont,apical)" -> 0
|| "(-voice,+cont,apical)"_"(+voice,vowel,[mid,central])" ];
6 states, 29 arcs, Circular.
File length:      371 bytes
```

  – **r31_2@.fst** for schwa deletion

```
regex [ "(+voice,vowel,[mid,central])" -> 0 || "(-voice,+cont,apical)"_? ];
5 states, 21 arcs, Circular.
File length:      312 bytes
```

rule31.fst was then implemented composing both subrules defined above.

Rule 32 (Velar reinforcement) was not implemented because of its strong dialectal character.

- **rule33.fst** *Velar fricativisation*

```
13 states, 301 arcs, Circular.
\newline File length:      2592 bytes
```

This rule was implemented stem-final, as stated. It consists of two subrules: the first one for the back vowels, the second one for the front vowels. The subrules were implemented as follows:

- **r33_1back.fst** for the back vowels

```
regex [ "(-voice,-cont,velar)" -> "(-voice,+cont,velar)"
|| BackVowel_StemInitial ];
4 states, 40 arcs, Circular.
File length:      936 bytes
```

- **r33_2front.fst** for the back vowels

```
regex [ "(-voice,-cont,velar)" -> "(-voice,+cont,palatal)"
|| FrontVowel_StemInitial ];
4 states, 55 arcs, Circular.
File length:      1380 bytes
```

rule33.fst was implemented through union building over both subrules.

- **rule34.fst** *Final /C/ → /k/ alternation*

This rule was implemented stem-final, as required by Kirchhoff. In order to avoid overgeneration of forms like

```
?'E:n.+lIk
```

(for *ähnlich*), it was restricted to the suffix *+ig/+IC*

```
regex [ "(-voice,+cont,palatal)" -> "(-voice,-cont,velar)"
|| "+" "(+voice,vowel,[high,front,central,-round])"_StemInitial+ ];
5 states, 25 arcs, Circular.
File length:      383 bytes
```

Rule 35 (*assimilation of voicelessness*) was left out when building of the final version of *x-vario*.

- **rule36.fst** *Cross-syllable nasal assimilation*

  13 states, 155 arcs, Circular.
  File length: 967 bytes

  This rule was implemented consisting of two subrules: the first one for the labial nasals, the second one for the velar nasals. The apical nasal could be omitted here, as no substitution takes place for it. The subrules were implemented as follows:

  – **r36_1labial.fst** for the labial nasals

```
regex [ "(+voice,nasal,apical)" -> "(+voice,nasal,labial)"
|| _StemBoundary Labial ];
5 states, 37 arcs, Circular.
File length:      552 bytes
```

  – **r36_2velar.fst** for the velar nasals

```
regex [ "(+voice,nasal,apical)" -> "(+voice,nasal,velar)"
|| _StemBoundary Velar ];
5 states, 34 arcs, Circular.
File length:      495 bytes
```

  rule36.fst was then implemented through union building of the two subrules.

Rules 37 (*Geminate simplification*) was not implemented, because it applies over words boundaries and is therefore irrelevant for the purposes of processing lexical representations. Rule 38 (*place assimilation of /t/*) was not implemented because it seems to occur only seldom.

7 Runtime Behaviour

The data presented here are derived from an application of the *x-vario* version presented and discussed in this paper to the *Bielefeld Lexical Database* as of August 1999. The program ran on a Sun SPARCstation Ultra 1 140, 128 RAM, under Solaris 7. Here are the average runtime data for *x-vario*:

```
real    23:07.4
user     9:14.3
sys        37.8
```

The *Bielefeld Lexical Database* contained 70911 canonical phonological representations of 70911 word forms, which were used as the input strings. On average, 4.35 speech variants per input string were generated.

8 Summary

In this paper, we have discussed one particular implementation of a knowledge base for the generation and recognition of speech variants. The phonetic rules, which were taken from the knowledge base described by Kirchhoff ([Kir95]), were first re-defined by means of regular expressions and then re-implemented in the form of finite state transducers. Uunion building of the single rule transducers yielded four large transducers. A script controls the access to the four transducers and combines their output into one single file, so that *x-vario* can be operated as if it were consisted of one single transducer. *x-vario* was applied to the *Bielefeld Lexical Database*, cf. [LGES98] . One excerpt of the generated speech variants database can be found in the appendix. This particular database was designed to be used in speech recognition. *x-vario* can be used equally for both purposes, the generation of all possible variants for a given lexical database, or the recognition of variant forms and their reduction to their lexical representation.

For some purposes it may be benefical not to apply all of the rules given in this report, but only a selection of them. Such a pre-selection might improve the recognition rate, when otherwise too many variants can be confused with other canonical or non-canonical forms. Moreover, if the existence of many variant forms enlargens the the lexicon drastically, the search space for each individual form gets unnecessarily big.

It is possible to exclude some of the rules of *x-vario*. For this purpose, one needs to build a new transducer consisting of only those rules that are needed, and apply it to the database using a script for the *XFST*-system similar to the one shown in the appendix.

All the rules stated above are pre-implemented, so that one just has to union the selected ones into one single transducer an to compose this last one with the converter and the inverter. To apply the *XFST*-script, just type in:

```
xfst -f script.file > output.file
```

If the input strings are put in each in one line with blank lines in between them, the transducer will produce the different variants in one block (seperated from the other blocks by blank lines) for each input string.

The use of speech variation rules will sometimes lead to overgeneration. If used generatively, they will also yield forms which are theoretically possible, but rather unlikely to occur. The very best thing to do in such cases is to combine the generation rules with other constraints, like frequency of occurrence of a certain rule in a representative corpus or within a lexical database. In its present form, *x-vario* will always generate all possible variants exhaustively.

References

[Gib95] Gibbon Gibbon. Verbmobil lexicon: Conventions for spelling and pronunciation. Verbmobil Technisches Dokument 31. Universität Bielefeld, 1995.

[HU93] John E. Hopcroft and Jeffrey D. Ullman. *Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie*. Addison-Wesley, 1993.

[Joh72] C. D. Johnson. *Formal Asoects of Phonological Description*. Mouton, 1972.

[Kar97] Lauri Karttunen. Syntax and semantics of regular expressions. *http://www.xrce.xerox.com/research/mltt/fst/fssyntax.html*, Fri Aug 13 13:04:37 GMT DST 1999, October 1997.

[Ker93] András Kertész. *Heuristik der Deutschen Phonologie*. Académiai Kiadó, 1993.

[KGK98] Lauri Karttunen, Tamás Gaál, and André Kempe. Xerox finite-state tool. *http://www.xrce.xerox.com/research/mltt/fsSoft/docs/fst-97/xfst97.html*, Fri Aug 13 13:01:45 GMT DST 1999, October 1998.

[Kir95] Katrin Kirchhoff. Two-level modelling of speech variant rules. *Verbmobil*, Report 82, May 1995.

[KK94] Ronald M. Kaplan and Martin Kay. Regular models of phonological rile systems. *Computational Linguistics*, 20, NUmber 3, 331-378, 1994.

[Koh77] Klaus J. Kohler. *Einführung in die Phonetik des Deutschen*. Ercih Schmidt Verlag, 1977.

[Koh94] Klaus J. Kohler. Glottal stops and glottalization in german. *Phonetica*, 51, 38-51, 1994.

[Koh96] Klaus J. Kohler. Phonetic realization of german / /-syllables. *Institut für Phonetik und Digitale Sprachverarbeitung, Universität Kiel*, 30, 159-194, April 1996.

[LGES98] H. Lüngen, D. Gibbon, K. Ehlebracht, and A.P. Quirino Simoes. Bielefelder lexikon und morphologie in verbmobil phase-ii. Verbmobil Report 233. Universität Bielefeld, 1998.

[LJPGA97] L.Karttunen, J-P.Chanod, G.Grefenstette, and A.Schiller. Regular expressions for language engineering. *http://www.xrce.xerox.com/research/mltt/fst/articles/jnle-97/rele.html*, Fri Aug 13 13:07:42 GMT DST 1999, January 1997.

A x-vario

The script for *x-vario*:

```
NUMORTH=$DATDIR/numOrth.txt
SORTED=$DATDIR/sortedvariantsLexDB.results
NONUMBERS=$SOFTDIR/noNumbers.awk
VARRESULTS=$DATDIR/variantsLexDB.results
NOMARKING=$SOFTDIR/noVariantMarking.awk
RESULTS=$RESULTSDIR/LexDB.results
ONELINE=$SOFTDIR/noVariantsOneLine.awk
ONELINED=$DATDIR/oneLined_LexDB.txt
PASTED=$DATDIR/phonseg_pasted.txt
FIRSTONELINE=$SOFTDIR/oneLine.awk
FIRSTONELINED=$DATDIR/firstOneLined.txt
KLAMMERN=$SOFTDIR/prettyPhonseg.awk
ORTHSEGPRETTY=$DATDIR/PrettyOrthSeg.txt
ORTHSEGREADY=$DATDIR/OrthsegReady.txt
PHONSEGREADY=$DATDIR/PhonsegReady.txt
NOPARENTH=$SOFTDIR/noParenth.awk
ADJUST=$SOFTDIR/lineAdjust.awk
ORTHADJUSTED=$DATDIR/orthAdjusted.txt


#gawk '{print $3}' $ORTHP > phonseg.orthp.wl
#gawk '{print $3}' $LEXDB > phonseg.lexdb.wl

#cat phonseg.lexdb.wl phonseg.orthp.wl |
#sort -u > $PHONSEG

gawk 'NR > 2 {print $4}' /homes/vmobil/VM/SOFTWARE/LEXDB-INTEGRATION/LexDb.Integrated > $PHONSEG
gawk 'NR > 2 {print $4}' /homes/vmobil/VM/SOFTWARE/LEXDB-INTEGRATION/LexDb.Integrated > $PRETTYPHONSEG
gawk 'NR > 2 {print $1}' /homes/vmobil/VM/SOFTWARE/LEXDB-INTEGRATION/LexDb.Integrated > $ORTHSEG

gawk -f $TWOLINES $PHONSEG > $TWOLINED
gawk -f $TWOLINES $PRETTYPHONSEG > $PRETTYTWOLINED
gawk -f $TWOLINES $ORTHSEG > $ORTHTWOLINED

gawk -f $LINESPLITT $TWOLINED > $LINESPLITTED
gawk -f $LINESPLITT $PRETTYTWOLINED > $PRETTYLINESPLITTED
gawk -f $LINESPLITT $ORTHTWOLINED > $ORTHSPLITTED

gawk -f $KLAMMERN $PRETTYLINESPLITTED > $PHONSEGPRETTY
gawk -f $KLAMMERN $ORTHSPLITTED > $ORTHSEGPRETTY

gawk -f $NOPARENTH $PHONSEGPRETTY > $PHONSEGREADY
```

```
gawk -f $NOPARENTH $ORTHSEGPRETTY > $ORTHSEGREADY
gawk -f $ADJUST $ORTHSEGREADY > $ORTHADJUSTED

xfst -f $XFST01 > $RESULTS01
xfst -f $XFST02 > $RESULTS02
xfst -f $XFST03 > $RESULTS03
xfst -f $XFST04 > $RESULTS04

gawk -f $VARMARKING $RESULTS01 > $VARRESULTS01
gawk -f $VARMARKING $RESULTS02 > $VARRESULTS02
gawk -f $VARMARKING $RESULTS03 > $VARRESULTS03
gawk -f $VARMARKING $RESULTS04 > $VARRESULTS04

gawk -f $BLOCKNUM $VARRESULTS01 > $NUMRESULTS01
gawk -f $BLOCKNUM $VARRESULTS02 > $NUMRESULTS02
gawk -f $BLOCKNUM $VARRESULTS03 > $NUMRESULTS03
gawk -f $BLOCKNUM $VARRESULTS04 > $NUMRESULTS04
#gawk -f $BLOCKNUM $PRETTYLINESPLITTED > $NUMPRETTY
#gawk -f $BLOCKNUM $ORTHSPLITTED > $NUMORTH



cat $NUMRESULTS04 >> $NUMRESULTS03
cat $NUMRESULTS03 >> $NUMRESULTS02
cat $NUMRESULTS02 >> $NUMRESULTS01

sort -u $NUMRESULTS01 | sort -n > $SORTED

gawk -f $NONUMBERS $SORTED > $VARRESULTS

gawk -f $FIRSTONELINE $VARRESULTS > $FIRSTONELINED

paste -d"  " $PHONSEGREADY  $FIRSTONELINED > $PASTED

gawk -f $ONELINE $PASTED > $ONELINED

paste -d"  " $ORTHADJUSTED $ONELINED > $RESULTS
```

B *convert.pl*

Prolog-Code for the converter. It was generated through the Prolog interface of the *XFST*-system:

```
network(net_df130).
arc(net_df130, 0, 1, "~").
arc(net_df130, 0, 1, "\").
arc(net_df130, 0, 1, " ").
arc(net_df130, 0, 1, "''").
arc(net_df130, 0, 1, "%?").
arc(net_df130, 0, 1, "^").
arc(net_df130, 0, 1, "+").
arc(net_df130, 0, 1, "#").
arc(net_df130, 0, 1, ".").
arc(net_df130, 0, 1, ":").
arc(net_df130, 0, 1, "'").
arc(net_df130, 0, 1, "I":"(+voice,vowel,[high,front,central,-round])").
arc(net_df130, 0, 1, "i":"(+voice,vowel,[high,front,-round])").
arc(net_df130, 0, 1, "i:":"(+voice,vowel,[long,high,front,-round])").
arc(net_df130, 0, 1, "y":"(+voice,vowel,[high,front,+round])").
arc(net_df130, 0, 1, "y:":"(+voice,vowel,[long,high,front,+round])").
arc(net_df130, 0, 1, "Y":"(+voice,vowel,[high,front,central,+round])").
arc(net_df130, 0, 1, "e":"(+voice,vowel,[mid,front,-round])").
arc(net_df130, 0, 1, "e:":"(+voice,vowel,[long,mid,front,-round])").
arc(net_df130, 0, 1, "E":"(+voice,vowel,[low,front,-round])").
arc(net_df130, 0, 1, "E:":"(+voice,vowel,[long,low,front,-round])").
arc(net_df130, 0, 1, "2":"(+voice,vowel,[mid,front,+round])").
arc(net_df130, 0, 1, "2:":"(+voice,vowel,[long,mid,front,+round])").
arc(net_df130, 0, 1, "9":"(+voice,vowel,[mid,front,central,+round])").
arc(net_df130, 0, 1, "u":"(+voice,vowel,[high,back,+round])").
arc(net_df130, 0, 1, "u:":"(+voice,vowel,[long,high,back,+round])").
arc(net_df130, 0, 1, "U":"(+voice,vowel,[high,back,central,+round])").
arc(net_df130, 0, 1, "o":"(+voice,vowel,[mid,back,+round])").
arc(net_df130, 0, 1, "o:":"(+voice,vowel,[long,mid,back,+round])").
arc(net_df130, 0, 1, "O":"(+voice,vowel,[mid,back,central,+round])").
arc(net_df130, 0, 1, "a":"(+voice,vowel,[low,back,-round])").
arc(net_df130, 0, 1, "a:":"(+voice,vowel,[long,low,back,-round])").
arc(net_df130, 0, 1, "6":"(+voice,vowel,[low,central])").
arc(net_df130, 0, 1, "@":"(+voice,vowel,[mid,central])").
arc(net_df130, 0, 1, "p":"(-voice,-cont,labial)").
arc(net_df130, 0, 1, "t":"(-voice,-cont,apical)").
arc(net_df130, 0, 1, "k":"(-voice,-cont,velar)").
arc(net_df130, 0, 1, "b":"(+voice,-cont,labial)").
arc(net_df130, 0, 1, "d":"(+voice,-cont,apical)").
arc(net_df130, 0, 1, "g":"(+voice,-cont,velar)").
arc(net_df130, 0, 1, "%?":"(-voice,-cont,glottal)").
```

```
arc(net_df130, 0, 1, "f":"(-voice,+cont,labial)").
arc(net_df130, 0, 1, "v":"(+voice,+cont,labial)").
arc(net_df130, 0, 1, "s":"(-voice,+cont,apical)").
arc(net_df130, 0, 1, "z":"(+voice,+cont,apical)").
arc(net_df130, 0, 1, "S":"(-voice,+cont,palato)").
arc(net_df130, 0, 1, "Z":"(+voice,+cont,palato)").
arc(net_df130, 0, 1, "j":"(+voice,+cont,palatal)").
arc(net_df130, 0, 1, "C":"(-voice,+cont,palatal)").
arc(net_df130, 0, 1, "x":"(-voice,+cont,velar)").
arc(net_df130, 0, 1, "h":"(-voice,+cont,glottal)").
arc(net_df130, 0, 1, "l":"(+voice,lateral,apical)").
arc(net_df130, 0, 1, "r":"(+voice,+cont,uvular)").
arc(net_df130, 0, 1, "m":"(+voice,nasal,labial)").
arc(net_df130, 0, 1, "n":"(+voice,nasal,apical)").
arc(net_df130, 0, 1, "N":"(+voice,nasal,velar)").
arc(net_df130, 0, 1, "ts":"(-voice,affr,apical)").
arc(net_df130, 0, 1, "pf":"(-voice,affr,labial)").
arc(net_df130, 0, 1, "tS":"(-voice,affr,palato)").
arc(net_df130, 1, 1, "tS":"(-voice,affr,palato)").
arc(net_df130, 1, 1, "pf":"(-voice,affr,labial)").
arc(net_df130, 1, 1, "ts":"(-voice,affr,apical)").
arc(net_df130, 1, 1, "N":"(+voice,nasal,velar)").
arc(net_df130, 1, 1, "n":"(+voice,nasal,apical)").
arc(net_df130, 1, 1, "m":"(+voice,nasal,labial)").
arc(net_df130, 1, 1, "r":"(+voice,+cont,uvular)").
arc(net_df130, 1, 1, "l":"(+voice,lateral,apical)").
arc(net_df130, 1, 1, "h":"(-voice,+cont,glottal)").
arc(net_df130, 1, 1, "x":"(-voice,+cont,velar)").
arc(net_df130, 1, 1, "C":"(-voice,+cont,palatal)").
arc(net_df130, 1, 1, "j":"(+voice,+cont,palatal)").
arc(net_df130, 1, 1, "Z":"(+voice,+cont,palato)").
arc(net_df130, 1, 1, "S":"(-voice,+cont,palato)").
arc(net_df130, 1, 1, "z":"(+voice,+cont,apical)").
arc(net_df130, 1, 1, "s":"(-voice,+cont,apical)").
arc(net_df130, 1, 1, "v":"(+voice,+cont,labial)").
arc(net_df130, 1, 1, "f":"(-voice,+cont,labial)").
arc(net_df130, 1, 1, "%?":"(-voice,-cont,glottal)").
arc(net_df130, 1, 1, "g":"(+voice,-cont,velar)").
arc(net_df130, 1, 1, "d":"(+voice,-cont,apical)").
arc(net_df130, 1, 1, "b":"(+voice,-cont,labial)").
arc(net_df130, 1, 1, "k":"(-voice,-cont,velar)").
arc(net_df130, 1, 1, "t":"(-voice,-cont,apical)").
arc(net_df130, 1, 1, "p":"(-voice,-cont,labial)").
arc(net_df130, 1, 1, "@":"(+voice,vowel,[mid,central])").
arc(net_df130, 1, 1, "6":"(+voice,vowel,[low,central])").
arc(net_df130, 1, 1, "a:":"(+voice,vowel,[long,low,back,-round])").
arc(net_df130, 1, 1, "a":"(+voice,vowel,[low,back,-round])").
arc(net_df130, 1, 1, "O":"(+voice,vowel,[mid,back,central,+round])").
```

```
arc(net_df130, 1, 1, "o:":"(+voice,vowel,[long,mid,back,+round])").
arc(net_df130, 1, 1, "o":"(+voice,vowel,[mid,back,+round])").
arc(net_df130, 1, 1, "U":"(+voice,vowel,[high,back,central,+round])").
arc(net_df130, 1, 1, "u:":"(+voice,vowel,[long,high,back,+round])").
arc(net_df130, 1, 1, "u":"(+voice,vowel,[high,back,+round])").
arc(net_df130, 1, 1, "9":"(+voice,vowel,[mid,front,central,+round])").
arc(net_df130, 1, 1, "2:":"(+voice,vowel,[long,mid,front,+round])").
arc(net_df130, 1, 1, "2":"(+voice,vowel,[mid,front,+round])").
arc(net_df130, 1, 1, "E:":"(+voice,vowel,[long,low,front,-round])").
arc(net_df130, 1, 1, "E":"(+voice,vowel,[low,front,-round])").
arc(net_df130, 1, 1, "e:":"(+voice,vowel,[long,mid,front,-round])").
arc(net_df130, 1, 1, "e":"(+voice,vowel,[mid,front,-round])").
arc(net_df130, 1, 1, "Y":"(+voice,vowel,[high,front,central,+round])").
arc(net_df130, 1, 1, "y:":"(+voice,vowel,[long,high,front,+round])").
arc(net_df130, 1, 1, "y":"(+voice,vowel,[high,front,+round])").
arc(net_df130, 1, 1, "i:":"(+voice,vowel,[long,high,front,-round])").
arc(net_df130, 1, 1, "i":"(+voice,vowel,[high,front,-round])").
arc(net_df130, 1, 1, "I":"(+voice,vowel,[high,front,central,-round])").
arc(net_df130, 1, 1, "'").
arc(net_df130, 1, 1, ":").
arc(net_df130, 1, 1, ".").
arc(net_df130, 1, 1, "#").
arc(net_df130, 1, 1, "+").
arc(net_df130, 1, 1, "^").
arc(net_df130, 1, 1, "%?").
arc(net_df130, 1, 1, "''").
arc(net_df130, 1, 1, " ").
arc(net_df130, 1, 1, "\").
arc(net_df130, 1, 1, "~").
final(net_df130, 1).
```

C The input

The two input files for *x-vario*: one with the orthographic representation of the word forms, the second one with the corresponding canonical forms in SAMPA-representation (cf. [Gib95]). Both files are generated by *x-vario* directly from the *Bielefeld Lexical Database*.

C.1 Orthographic representation

```
"Agypten
"Agyptens
"Ahnlich
"Ahnliche
"Ahnlichem
"Ahnlichen
"Ahnlicher
"Ahnliches
"Amter
"Amtern
"Andern
"Anderns
"Anderung
"Anderungen
"Angste
"Angsten
"Apfel
"Apfeln
"Appeln
"Appelns
"Arger
"Argern
"Argerns
"Argers
"Arzte
"Arzten
"Ather
"Athern
"Athers
"Au"sern
"Au"serns
"Odipus
"Odipusse
"Odipussen
"Offnen
"Offnens
"Offnungszeit
"Offnungszeiten
```

"Ogai
"Ogai-Jahrestagung
"Ogai-Jahrestagungen
"Ogai-Tagung
"Ogai-Tagungen
"Ogais
"Ol
"Ole
"Olen
"Ols
"Ortern
"Orterns
"Ortlichkeit
"Ortlichkeiten
"Ubel
"Ubeln
"Ubels
"Uberarbeiten
"Uberarbeitens
"Uberblick
"Uberblicke
"Uberblicken
"Uberblickens
"Uberblicks
"Uberblicksbericht
"Uberblicksberichte
"Uberblicksberichten
"Uberblicksberichts
"Uberbr"ucken
"Uberbr"uckens
"Uberbringen
"Uberbringens
"Ubereink"unfte
"Ubereink"unften
"Ubereinkommen
"Ubereinkommens
"Ubereinkunft
"Ubereinstimmen
"Ubereinstimmens
"Ubereinstimmung
"Ubereinstimmungen
"Uberf"ullen
"Uberf"ullens
"Uberfordern
"Uberforderns
"Uberg"ange
"Uberg"angen
"Ubergang

```
"Ubergange
"Uberganges
"Ubergeben
"Ubergebens
"Ubergehen
"Ubergehens
"Uberlassen
"Uberlassens
"Uberlaufen
"Uberlaufens
"Uberleben
"Uberlebens
"Uberlegen
"Uberlegens
"Uberlegung
"Uberlegungen
"Ubernachten
"Ubernachtens
"Ubernachtung
"Ubernachtungen
"Ubernachtungsgelegenheit
"Ubernachtungsgelegenheiten
"Ubernachtungskosten
"Ubernachtungsm"oglichkeit
"Ubernachtungsm"oglichkeiten
"Ubernachtungspreis
"Ubernachtungspreise
"Ubernachtungspreisen
"Ubernachtungspreises
```

C.2 Canonical phonological representation

```
?E:.g'Yp.t+@n
?E:.g'Yp.t+@n#+s
?'E:n.+lIC
?'E:n.+lI.C#+@
?'E:n.+lI.C#+@m
?'E:n.+lI.C#+@n
?'E:n.+lI.C#+6
?'E:n.+lI.C#+@s
?'Em.t#+6
?'Em.t#+6n
?'En.d6+n
?'En.d6+n#+s
?'En.d@.r+UN
```

?’En.d@.r+U.N#+@n
?’ENs.t#+@
?’ENs.t#+@n
?’E.pf@l
?’E.pf@l#+n
?’E.p@l+n
?’E.p@l+n#+s
?’E6.g6
?’E6.g6+n
?’E6.g6+n#+s
?’E6.g6#+s
?’E6ts.t#+@
?’E6ts.t#+@n
?’E:.t+6
?’E:.t+6#+n
?’E:.t+6#+s
?’OY.s6+n
?’OY.s6+n#+s
?’2:.dI.p+Us
?’2:.dI.p+U.s#+@
?’2:.dI.p+U.s#+@n
?’9f.n+@n
?’9f.n+@n#+s
?’9f.n+UN#+s#ts’’aIt
?’9f.n+UN#+s#ts’’aI.t#+@n
?’2:.gaI
?’’2:.gaI#j’a:.r#+@s#t’’a:.g+UN
?’’2:.gaI#j’a:.r#+@s#t’’a:.g+U.N#+@n
?’2:.gaI#t’’a:.g+UN
?’2:.gaI#t’’a:.g+U.N#+@n
?’2:.gaI#+s
?’2:l
?’2:.l#+@
?’2:.l#+@n
?’2:l#+s
?’96.t6+n
?’96.t6+n#+s
?’96t.+lIC.+kaIt
?’96t.+lIC.+kaI.t#+@n
?’y:.b@l
?’y:.b@l#+n
?’y:.b@l#+s
?y:.b6.+?’a6.baI.t+@n
?y:.b6.+?’a6.baI.t+@n#+s
?’y:.b6#bl’’Ik
?’y:.b6#bl’’I.k#+@
?’y:.b6#bl’’I.k#+@n;?y:.b6.+bl’I.k+@n
?y:.b6.+bl’I.k+@n#+s

```
?'y:.b6#bl''Ik#+s
?'y:.b6#bl''Ik#+s#b@.+r''ICt
?'y:.b6#bl''Ik#+s#b@.+r''IC.t#+@
?'y:.b6#bl''Ik#+s#b@.+r''IC.t#+@n
?'y:.b6#bl''Ik#+s#b@.+r''ICt#+s
?y:.b6.+br'Y.k+@n
?y:.b6.+br'Y.k+@n#+s
?y:.b6.+br'I.N+@n
?y:.b6.+br'I.N+@n#+s
?''y:.b6#?'aIn#k''Ynf.t#+@
?''y:.b6#?'aIn#k''Ynf.t#+@n
?y:.b6#?'aIn#k''O.m+@n
?y:.b6#?'aIn#k''O.m+@n#+s
?''y:.b6#?'aIn#k''Unft
?y:.b6#?'aIn#St''I.m+@n
?y:.b6#?'aIn#St''I.m+@n#+s
?y:.b6#?'aIn#St''I.m+UN
?y:.b6#?'aIn#St''I.m+U.N#+@n
?y:.b6.+f'Y.l+@n
?y:.b6.+f'Y.l+@n#+s
?y:.b6.+f'O6.d6+n
?y:.b6.+f'O6.d6+n#+s
?'y:.b6#g''E.N#+@
?'y:.b6#g''E.N#+@n
?'y:.b6#g''aN
?'y:.b6#g''a.N#+@
?'y:.b6#g''a.N#+@s
?y:.b6.+g'e:.b+@n
?y:.b6.+g'e:.b+@n#+s
?'y:.b6#g''e:.+@n;?y:.b6.+g'e:.+@n
?'y:.b6#g''e:.+@n#+s;?y:.b6.+g'e:.+@n#+s
?y:.b6.+l'a.s+@n
?y:.b6.+l'a.s+@n#+s
?'y:.b6#l''aU.f+@n;?y:.b6.+l'aU.f+@n
?'y:.b6#l''aU.f+@n#+s;?y:.b6.+l'aU.f+@n#+s
?y:.b6.+l'e:.b+@n
?y:.b6.+l'e:.b+@n#+s
?y:.b6.+l'e:.g+@n
?y:.b6.+l'e:.g+@n#+s
?y:.b6.+l'e:.g+UN
?y:.b6.+l'e:.g+U.N#+@n
?y:.b6.+n'ax.t+@n
?y:.b6.+n'ax.t+@n#+s
?y:.b6.+n'ax.t+UN
?y:.b6.+n'ax.t+U.N#+@n
?y:.b6.+n'ax.t+UN#+s#g@.+l''e:.g+@n.+haIt
?y:.b6.+n'ax.t+UN#+s#g@.+l''e:.g+@n.+haI.t#+@n
?y:.b6.+n'ax.t+UN#+s#k''Os.t#+@n
```

```
?y:.b6.+n'ax.t+UN#+s#m''2:k.+lIC.+kaIt
?y:.b6.+n'ax.t+UN#+s#m''2:k.+lIC.+kaI.t#+@n
?''y:.b6#n'ax.t+UN#+s#pr''aIs
?''y:.b6#n'ax.t+UN#+s#pr''aI.z#+@
?''y:.b6#n'ax.t+UN#+s#pr''aI.z#+@n
?''y:.b6#n'ax.t+UN#+s#pr''aI.z#+@s
```

D The output

```
< "Agypten >  < ?E:.g'Yp.t+@n > <  ?@.g'Yp.t+@n ?E.g'Yp.t+@n ?E:.g'Yp.t+@n ?e:.g'Yp.t+@n E:.g'Yp.t+@n   >
< "Agyptens >  < ?E:.g'Yp.t+@n#+s > <  ?@.g'Yp.t+@n#+s ?E.g'Yp.t+@n#+s ?E:.g'Yp.t+@n#+s ?e:.g'Yp.t+@n#+s E:.g'Yp.t+@n#+s  >
< "Ahnlich >  < ?'E:n.+lIC > <  'E:n.+lIC ?'E:n.+lIC ?'En.+lIC ?'e:n.+lIC  >
< "Ahnliche >  < ?'E:n.+lI.C#+@ > <  'E:n.+lI.C#+@ ?'E:n.+lI.C#+@ ?'En.+lI.C#+@ ?'e:n.+lI.C#+@  >
< "Ahnlichem >  < ?'E:n.+lI.C#+@m > <  'E:n.+lI.C#+@m ?'E:n.+lI.C#+@m ?'En.+lI.C#+@m ?'e:n.+lI.C#+@m  >
< "Ahnlichen >  < ?'E:n.+lI.C#+@n > <  'E:n.+lI.C#+@n ?'E:n.+lI.C#+@n ?'En.+lI.C#+@n ?'e:n.+lI.C#+@n  >
< "Ahnlicher >  < ?'E:n.+lI.C#+6 > <  'E:n.+lI.C#+6 ?'E:n.+lI.C#+6 ?'En.+lI.C#+6 ?'e:n.+lI.C#+6  >
< "Ahnliches >  < ?'E:n.+lI.C#+@s > <  'E:n.+lI.C#+@s ?'E:n.+lI.C#+@s ?'En.+lI.C#+@s ?'e:n.+lI.C#+@s  >
< "Amter >  < ?'Em.t#+6 > <  'Em.t#+6 ?'Em.t#+6  >
< "Amtern >  < ?'Em.t#+6n > <  'Em.t#+6n ?'Em.t#+6n  >
< "Andern >  < ?'En.d6+n > <  'En.d6+n ?'En.d6+n  >
< "Anderns >  < ?'En.d6+n#+s > <  'En.d6+n#+s ?'En.d6+n#+s  >
< "Anderung >  < ?'En.d@.r+UN > <  'En.d@.r+UN ?'En.d@.r+UN  >
< "Anderungen >  < ?'En.d@.r+U.N#+@n > <  'En.d@.r+U.N#+@n ?'En.d@.r+U.N#+@n  >
< "Angste >  < ?'ENs.t#+@ > <  'ENs.t#+@ ?'ENs.t#+@  >
< "Angsten >  < ?'ENs.t#+@n > <  'ENs.t#+@n ?'ENs.t#+@n  >
< "Apfel >  < ?'E.pf@l > <  'E.pf@l ?'E.pf@l ?'E.pfl  >
< "Apfeln >  < ?'E.pf@l#+n > <  'E.pf@l#+n ?'E.pf@l#+n  >
< "Appeln >  < ?'E.p@l+n > <  'E.p@l+n ?'E.p@l+n  >
< "Appelns >  < ?'E.p@l+n#+s > <  'E.p@l+n#+s ?'E.p@l+n#+s  >
< "Arger >  < ?'E6.g6 > <  'E6.g6 ?'E.g6 ?'E6.g6 ?'e6.g6  >
< "Argern >  < ?'E6.g6+n > <  'E6.g6+n ?'E.g6+n ?'E6.g6+n ?'e6.g6+n  >
< "Argerns >  < ?'E6.g6+n#+s > <  'E6.g6+n#+s ?'E.g6+n#+s ?'E6.g6+n#+s ?'e6.g6+n#+s  >
< "Argers >  < ?'E6.g6#+s > <  'E6.g6#+s ?'E.g6#+s ?'E6.g6#+s ?'e6.g6#+s  >
< "Arzte >  < ?'E6ts.t#+@ > <  'E6ts.t#+@ ?'E6ts.t#+@ ?'Ets.t#+@ ?'e6ts.t#+@  >
< "Arzten >  < ?'E6ts.t#+@n > <  'E6ts.t#+@n ?'E6ts.t#+@n ?'Ets.t#+@n ?'e6ts.t#+@n  >
< "Ather >  < ?'E:.t+6 > <  'E:.t+6 ?'E.t+6 ?'E:.t+6 ?'e:.t+6  >
< "Athern >  < ?'E:.t+6#+n > <  'E:.t+6#+n ?'E.t+6#+n ?'E:.t+6#+n ?'e:.t+6#+n  >
< "Athers >  < ?'E:.t+6#+s > <  'E:.t+6#+s ?'E.t+6#+s ?'E:.t+6#+s ?'e:.t+6#+s  >
< "Au"sern >  < ?'OY.s6+n > <  'OY.s6+n ?'OY.s6+n  >
< "Au"serns >  < ?'OY.s6+n#+s > <  'OY.s6+n#+s ?'OY.s6+n#+s  >
< "Odipus >  < ?'2:.dI.p+Us > <  '2:.dI.p+Us ?'2.dI.p+Us ?'2:.dI.p+Us  >
< "Odipusse >  < ?'2:.dI.p+U.s#+@ > <  '2:.dI.p+U.s#+@ ?'2.dI.p+U.s#+@ ?'2:.dI.p+U.s#+@  >
< "Odipussen >  < ?'2:.dI.p+U.s#+@n > <  '2:.dI.p+U.s#+@n ?'2.dI.p+U.s#+@n ?'2:.dI.p+U.s#+@n ?'2:.dI.p+U.s#+n  >
< "Offnen >  < ?'9f.n+@n > <  '9f.n+@n ?'9f.n+@n  >
< "Offnens >  < ?'9f.n+@n#+s > <  '9f.n+@n#+s ?'9f.n+@n#+s  >
< "Offnungszeit >  < ?'9f.n+UN#+s#ts''aIt > <  '9f.n+UN#+s#ts''aIt ?'9f.n+UN#+s#s''aIt ?'9f.n+UN#+s#ts''aIt  >
< "Offnungszeiten >  < ?'9f.n+UN#+s#ts''aI.t#+@n > <  '9f.n+UN#+s#ts''aI.t#+@n ?'9f.n+UN#+s#s''aI.t#+@n ?'9f.n+UN#+s#ts''aI.t#+@n  >
< "Ogai >  < ?'2:.gaI > <  '2:.gaI ?'2.gaI ?'2:.gaI  >
< "Ogai-Jahrestagung >  < ?''2:.gaI#j'a:.r#+@s#t''a:.g+UN > <  ''2:.gaI#j'a:.r#+@s#t''a:.g+UN ?''2.gaI#j'a.r#+@s#t''a.g+UN ?''2.gaI#j'a.r#+@s#t''a:.g+UN ?''2.gaI#j'a:.r#+@s#t''a.g+UN ?''2.gaI#j'a:.r#+@s#t''a:.g+UN ?''2:.gaI#j'a.r#+@s#t''a.g+UN ?''2:.gaI#j'a.r#+@s#t''a:.g+UN ?''2:.gaI#j'a:.r#+@s#t''a.g+UN ?''2:.gaI#j'a:.r#+@s#t''a:.g+UN  >
< "Ogai-Jahrestagungen >  < ?''2:.gaI#j'a:.r#+@s#t''a:.g+U.N#+@n > <  ''2:.gaI#j'a:.r#+@s#t''a:.g+U.N#+@n ?''2.gaI#j'a.r#+@s#t''a.g+U.N#+@n ?''2.gaI#j'a.r#+@s#t''a:.g+U.N#+@n ?''2.gaI#j'a:.r#+@s#t''a.g+U.N#+@n ?''2.gaI#j'a:.r#+@s#t''a:.g+U.N#+@n ?''2:.gaI#j'a.r#+@s#t''a.g+U.N#+@n ?''2:.gaI#j'a.r#+@s#t''a:.g+U.N#+@n ?''2:.gaI#j'a:.r#+@s#t''a.g+U.N#+@n ?''2:.gaI#j'a:.r#+@s#t''a:.g+U.N#+@n  >
< "Ogai-Tagung >  < ?'2:.gaI#t''a:.g+UN > <  '2:.gaI#t''a:.g+UN ?'2.gaI#t''a.g+UN ?'2.gaI#t''a:.g+UN ?'2:.gaI#t''a.g+UN ?'2:.gaI#t''a:.g+UN  >
< "Ogai-Tagungen >  < ?'2:.gaI#t''a:.g+U.N#+@n > <  '2:.gaI#t''a:.g+U.N#+@n ?'2.gaI#t''a.g+U.N#+@n ?'2.gaI#t''a:.g+U.N#+@n ?'2:.gaI#t''a.g+U.N#+@n ?'2:.gaI#t''a:.g+U.N#+@n  >
< "Ogais >  < ?'2:.gaI#+s > <  '2:.gaI#+s ?'2.gaI#+s ?'2:.gaI#+s  >
< "Ol >  < ?'2:l > <  '2:l ?'2:l ?'2l  >
< "Ole >  < ?'2:.l#+@ > <  '2:.l#+@ ?'2.l#+@ ?'2:.l#+@  >
< "Olen >  < ?'2:.l#+@n > <  '2:.l#+@n ?'2.l#+@n ?'2:.l#+@n  >
< "Ols >  < ?'2:l#+s > <  '2:l#+s ?'2:l#+s ?'2l#+s  >
< "Ortern >  < ?'96.t6+n > <  '96.t6+n ?'9.t6+n ?'96.t6+n  >
< "Orterns >  < ?'96.t6+n#+s > <  '96.t6+n#+s ?'9.t6+n#+s ?'96.t6+n#+s  >
< "Ortlichkeit >  < ?'96t.+lIC.+kaIt > <  '96t.+lIC.+kaIt ?'96t.+lIC.+kaIt ?'9t.+lIC.+kaIt  >
< "Ortlichkeiten >  < ?'96t.+lIC.+kaI.t#+@n > <  '96t.+lIC.+kaI.t#+@n ?'96t.+lIC.+kaI.t#+@n ?'9t.+lIC.+kaI.t#+@n  >
< "Ubel >  < ?'y:.b@l > <  'y:.b@l ?'Y.b@l ?'y.b@l ?'y:.b@l ?'y:.bl  >
< "Ubeln >  < ?'y:.b@l#+n > <  'y:.b@l#+n ?'Y.b@l#+n ?'y.b@l#+n ?'y:.b@l#+n  >
< "Ubels >  < ?'y:.b@l#+s > <  'y:.b@l#+s ?'Y.b@l#+s ?'y.b@l#+s ?'y:.b@l#+s  >
< "Uberarbeiten >  < ?y:.b6.+?'a6.baI.t+@n > <  ?@.b6.+?'a6.baI.t+@n ?Y.b6.+?'a6.baI.t+@n ?y.b6.+?'a6.baI.t+@n ?y:.b6.+'a6.baI.t+@n ?y:.b6.+?'a.baI.t+@n ?y:.b6.+?'a6.baI.t+@n y:.b6.+'a6.baI.t+@n y:.b6.+?'a6.baI.t+@n  >
< "Uberarbeitens >  < ?y:.b6.+?'a6.baI.t+@n#+s > <  ?@.b6.+?'a6.baI.t+@n#+s ?Y.b6.+?'a6.baI.t+@n#+s ?y.b6.+?'a6.baI.t+@n#+s ?y:.b6.+'a6.baI.t+@n#+s ?y:.b6.+?'a.baI.t+@n#+s ?y:.b6.+?'a6.baI.t+@n#+s y:.b6.+'a6.baI.t+@n#+s y:.b6.+?'a6.baI.t+@n#+s  >
< "Uberblick >  < ?'y:.b6#bl''Ik > <  'y:.b6#bl''Ik ?'Y.b6#bl''Ik ?'y.b6#bl''Ik ?'y:.b6#bl''IC ?'y:.b6#bl''Ik  >
< "Uberblicke >  < ?'y:.b6#bl''I.k#+@ > <  'y:.b6#bl''I.k#+@ ?'Y.b6#bl''I.k#+@ ?'y.b6#bl''I.k#+@ ?'y:.b6#bl''I.k#+@  >
< "Uberblicken >  < ?'y:.b6#bl''I.k#+@n > <  'y:.b6#bl''I.k#+@n ?'Y.b6#bl''I.k#+@n ?'y.b6#bl''I.k#+@n ?'y:.b6#bl''I.k#+@n  >< ?y:.b6.+bl'I.k+@n > <    ?@.b6.+bl'I.k+@n   ?Y.b6.+bl'I.k+@n   ?y.b6.+bl'I.k+@n   ?y:.b6.+bl'I.k+@n   y:.b6.+bl'I.k+@n  >
< "Uberblickens >  < ?y:.b6.+bl'I.k+@n#+s > <  ?@.b6.+bl'I.k+@n#+s ?Y.b6.+bl'I.k+@n#+s ?y.b6.+bl'I.k+@n#+s ?y:.b6.+bl'I.k+@n#+s y:.b6.+bl'I.k+@n#+s  >
< "Uberblicks >  < ?'y:.b6#bl''Ik#+s > <  'y:.b6#bl''Ik#+s ?'Y.b6#bl''Ik#+s ?'y.b6#bl''Ik#+s ?'y:.b6#bl''IC#+s ?'y:.b6#bl''Ik#+s  >
< "Uberblicksbericht >  < ?'y:.b6#bl''Ik#+s#b@.+r''ICt > <  'y:.b6#bl''Ik#+s#b@.+r''ICt ?'Y.b6#bl''Ik#+s#b@.+r''ICt ?'y.b6#bl''Ik#+s#b@.+r''ICt ?'y:.b6#bl''IC#+s#b@.+r''ICt ?'y:.b6#bl''Ik#+s#b@.+r''ICt  >
< "Uberblicksberichte >  < ?'y:.b6#bl''Ik#+s#b@.+r''IC.t#+@ > <  'y:.b6#bl''Ik#+s#b@.+r''IC.t#+@ ?'Y.b6#bl''Ik#+s#b@.+r''IC.t#+@ ?'y.b6#bl''Ik#+s#b@.+r''IC.t#+@ ?'y:.b6#bl''IC#+s#b@.+r''IC.t#+@ ?'y:.b6#bl''Ik#+s#b@.+r''IC.t#+@  >
< "Uberblicksberichten >  < ?'y:.b6#bl''Ik#+s#b@.+r''IC.t#+@n > <  'y:.b6#bl''Ik#+s#b@.+r''IC.t#+@n ?'Y.b6#bl''Ik#+s#b@.+r''IC.t#+@n ?'y.b6#bl''Ik#+s#b@.+r''IC.t#+@n ?'y:.b6#bl''IC#+s#b@.+r''IC.t#+@n ?'y:.b6#bl''Ik#+s#b@.+r''IC.t#+@n  >
< "Uberblicksberichts >  < ?'y:.b6#bl''Ik#+s#b@.+r''ICt#+s > <  'y:.b6#bl''Ik#+s#b@.+r''ICt#+s ?'Y.b6#bl''Ik#+s#b@.+r''ICt#+s ?'y.b6#bl''Ik#+s#b@.+r''ICt#+s ?'y:.b6#bl''IC#+s#b@.+r''ICt#+s ?'y:.b6#bl''Ik#+s#b@.+r''ICt#+s  >
< "Uberbr"ucken >  < ?y:.b6.+br'Y.k+@n > <  ?@.b6.+br'Y.k+@n ?Y.b6.+br'Y.k+@n ?y.b6.+br'Y.k+@n ?y:.b6.+br'Y.k+@n y:.b6.+br'Y.k+@n  >
< "Uberbr"uckens >  < ?y:.b6.+br'Y.k+@n#+s > <  ?@.b6.+br'Y.k+@n#+s ?Y.b6.+br'Y.k+@n#+s ?y.b6.+br'Y.k+@n#+s ?y:.b6.+br'Y.k+@n#+s y:.b6.+br'Y.k+@n#+s  >
< "Uberbringen >  < ?y:.b6.+br'I.N+@n > <  ?@.b6.+br'I.N+@n ?Y.b6.+br'I.N+@n ?y.b6.+br'I.N+@n ?y:.b6.+br'I.N+@n y:.b6.+br'I.N+@n  >
< "Uberbringens >  < ?y:.b6.+br'I.N+@n#+s > <  ?@.b6.+br'I.N+@n#+s ?Y.b6.+br'I.N+@n#+s ?y.b6.+br'I.N+@n#+s ?y:.b6.+br'I.N+@n#+s y:.b6.+br'I.N+@n#+s  >
< "Ubereink"unfte >  < ?''y:.b6#?'aIn#k''Ynf.t#+@ > <  ''y:.b6#'aIn#k''Ynf.t#+@ ''y:.b6#?'aIn#k''Ynf.t#+@ ?''Y.b6#?'aIn#k''Ynf.t#+@ ?''y.b6#?'aIn#k''Ynf.t#+@ ?''y:.b6#'aIn#k''Ynf.t#+@ ?''y:.b6#?'aIN#k''Ynf.t#+@ ?''y:.b6#?'aIn#k''Ynf.t#+@  >
< "Ubereink"unften >  < ?''y:.b6#?'aIn#k''Ynf.t#+@n > <  ''y:.b6#'aIn#k''Ynf.t#+@n ''y:.b6#?'aIn#k''Ynf.t#+@n ?''Y.b6#?'aIn#k''Ynf.t#+@n ?''y.b6#?'aIn#k''Ynf.t#+@n ?''y:.b6#'aIn#k''Ynf.t#+@n ?''y:.b6#?'aIN#k''Ynf.t#+@n ?''y:.b6#?'aIn#k''Ynf.t#+@n  >
< "Ubereinkommen >  < ?y:.b6#?'aIn#k''O.m+@n > <  ?@.b6#?'aIn#k''O.m+@n ?Y.b6#?'aIn#k''O.m+@n ?y.b6#?'aIn#k''O.m+@n ?y:.b6#'aIn#k''O.m+@n ?y:.b6#?'aIN#k''O.m+@n ?y:.b6#?'aIn#k''O.m+@n y:.b6#'aIn#k''O.m+@n y:.b6#?'aIn#k''O.m+@n  >
< "Ubereinkommens >  < ?y:.b6#?'aIn#k''O.m+@n#+s > <  ?@.b6#?'aIn#k''O.m+@n#+s ?Y.b6#?'aIn#k''O.m+@n#+s ?y.b6#?'aIn#k''O.m+@n#+s ?y:.b6#'aIn#k''O.m+@n#+s ?y:.b6#?'aIN#k''O.m+@n#+s ?y:.b6#?'aIn#k''O.m+@n#+s y:.b6#'aIn#k''O.m+@n#+s y:.b6#?'aIn#k''O.m+@n#+s  >
< "Ubereinkunft >  < ?''y:.b6#?'aIn#k''Unft > <  ''y:.b6#'aIn#k''Unft ''y:.b6#?'aIn#k''Unft ?''Y.b6#?'aIn#k''Unft ?''y.b6#?'aIn#k''Unft ?''y:.b6#'aIn#k''Unft ?''y:.b6#?'aIN#k''Unft ?''y:.b6#?'aIn#k''Unft  >
< "Ubereinstimmen >  < ?y:.b6#?'aIn#St''I.m+@n > <  ?@.b6#?'aIn#St''I.m+@n ?Y.b6#?'aIn#St''I.m+@n ?y.b6#?'aIn#St''I.m+@n ?y:.b6#'aIn#St''I.m+@n ?y:.b6#?'aIn#St''I.m+@n y:.b6#'aIn#St''I.m+@n y:.b6#?'aIn#St''I.m+@n  >
< "Ubereinstimmens >  < ?y:.b6#?'aIn#St''I.m+@n#+s > <  ?@.b6#?'aIn#St''I.m+@n#+s ?Y.b6#?'aIn#St''I.m+@n#+s ?y.b6#?'aIn#St''I.m+@n#+s ?y:.b6#'aIn#St''I.m+@n#+s ?y:.b6#?'aIn#St''I.m+@n#+s y:.b6#'aIn#St''I.m+@n#+s y:.b6#?'aIn#St''I.m+@n#+s  >
< "Ubereinstimmung >  < ?y:.b6#?'aIn#St''I.m+UN > <  ?@.b6#?'aIn#St''I.m+UN ?Y.b6#?'aIn#St''I.m+UN ?y.b6#?'aIn#St''I.m+UN ?y:.b6#'aIn#St''I.m+UN ?y:.b6#?'aIn#St''I.m+UN y:.b6#'aIn#St''I.m+UN y:.b6#?'aIn#St''I.m+UN  >
< "Ubereinstimmungen >  < ?y:.b6#?'aIn#St''I.m+U.N#+@n > <  ?@.b6#?'aIn#St''I.m+U.N#+@n ?Y.b6#?'aIn#St''I.m+U.N#+@n ?y.b6#?'aIn#St''I.m+U.N#+@n ?y:.b6#'aIn#St''I.m+U.N#+@n ?y:.b6#?'aIn#St''I.m+U.N#+@n y:.b6#'aIn#St''I.m+U.N#+@n y:.b6#?'aIn#St''I.m+U.N#+@n  >
< "Uberf"ullen >  < ?y:.b6.+f'Y.l+@n > <  ?@.b6.+f'Y.l+@n ?Y.b6.+f'Y.l+@n ?y.b6.+f'Y.l+@n ?y:.b6.+f'Y.l+@n y:.b6.+f'Y.l+@n  >
< "Uberf"ullens >  < ?y:.b6.+f'Y.l+@n#+s > <  ?@.b6.+f'Y.l+@n#+s ?Y.b6.+f'Y.l+@n#+s ?y.b6.+f'Y.l+@n#+s ?y:.b6.+f'Y.l+@n#+s y:.b6.+f'Y.l+@n#+s  >
< "Uberfordern >  < ?y:.b6.+f'O6.d6+n > <  ?@.b6.+f'O6.d6+n ?Y.b6.+f'O6.d6+n ?y.b6.+f'O6.d6+n ?y:.b6.+f'O.d6+n ?y:.b6.+f'O6.d6+n y:.b6.+f'O6.d6+n  >
< "Uberforderns >  < ?y:.b6.+f'O6.d6+n#+s > <  ?@.b6.+f'O6.d6+n#+s ?Y.b6.+f'O6.d6+n#+s ?y.b6.+f'O6.d6+n#+s ?y:.b6.+f'O.d6+n#+s ?y:.b6.+f'O6.d6+n#+s y:.b6.+f'O6.d6+n#+s  >
< "Uberg"ange >  < ?'y:.b6#g''E.N#+@ > <  'y:.b6#g''E.N#+@ ?'Y.b6#g''E.N#+@ ?'y.b6#g''E.N#+@ ?'y:.b6#g''E.N#+@  >
< "Uberg"angen >  < ?'y:.b6#g''E.N#+@n > <  'y:.b6#g''E.N#+@n ?'Y.b6#g''E.N#+@n ?'y.b6#g''E.N#+@n ?'y:.b6#g''E.N#+@n  >
< "Ubergang >  < ?'y:.b6#g''aN > <  'y:.b6#g''aN ?'Y.b6#g''aN ?'y.b6#g''aN ?'y:.b6#g''aN  >
< "Ubergange >  < ?'y:.b6#g''a.N#+@ > <  'y:.b6#g''a.N#+@ ?'Y.b6#g''a.N#+@ ?'y.b6#g''a.N#+@ ?'y:.b6#g''a.N#+@  >
< "Uberganges >  < ?'y:.b6#g''a.N#+@s > <  'y:.b6#g''a.N#+@s ?'Y.b6#g''a.N#+@s ?'y.b6#g''a.N#+@s ?'y:.b6#g''a.N#+@s  >
< "Ubergeben >  < ?y:.b6.+g'e:.b+@n > <  ?@.b6.+g'e:.b+@n ?Y.b6.+g'e:.b+@n ?y.b6.+g'e.b+@n ?y.b6.+g'e:.b+@n ?y:.b6.+g'E.b+@n ?y:.b6.+g'e.b+@n ?y:.b6.+g'e:.b+@n y:.b6.+g'e:.b+@n  >
< "Ubergebens >  < ?y:.b6.+g'e:.b+@n#+s > <  ?@.b6.+g'e:.b+@n#+s ?Y.b6.+g'e:.b+@n#+s ?y.b6.+g'e.b+@n#+s ?y.b6.+g'e:.b+@n#+s ?y:.b6.+g'E.b+@n#+s ?y:.b6.+g'e.b+@n#+s ?y:.b6.+g'e:.b+@n#+s y:.b6.+g'e:.b+@n#+s  >
```

```
< "Ubergehen >  < ?'y:.b6#g''e:.+@n > <  'y:.b6#g''e:.+@n ?'Y.b6#g''e:.+@n ?'y.b6#g''e.+@n ?'y.b6#g''e:.+@n ?'y:.b6#g''E.+@n ?'y:.b6#g''e.+@n ?'y:.b6#g''e:.+@n  ><  ?y:.b6.+g'e:.+@n > <    ?@.b6.+g'e:.+@n   ?Y.b6.+g'e:.+@n   ?y.b6.+g'e.+@n   ?y.b6.+g'e:.+@n   ?y:.b6.+g'E.+@n   ?y:.b6.+g'e.+@n   ?y:.b6.+g'e:.+@n   y:.b6.+g'e:.+@n  >
< "Ubergehens >  < ?'y:.b6#g''e:.+@n#+s > <  'y:.b6#g''e:.+@n#+s ?'Y.b6#g''e:.+@n#+s ?'y.b6#g''e.+@n#+s ?'y.b6#g''e:.+@n#+s ?'y:.b6#g''E.+@n#+s ?'y:.b6#g''e.+@n#+s ?'y:.b6#g''e:.+@n#+s  ><  ?y:.b6.+g'e:.+@n#+s > <    ?@.b6.+g'e:.+@n#+s   ?Y.b6.+g'e:.+@n#+s   ?y.b6.+g'e.+@n#+s   ?y.b6.+g'e:.+@n#+s   ?y:.b6.+g'E.+@n#+s   ?y:.b6.+g'e.+@n#+s   ?y:.b6.+g'e:.+@n#+s   y:.b6.+g'e:.+@n#+s  >
< "Uberlassen >  < ?y:.b6.+l'a.s+@n > <  ?@.b6.+l'a.s+@n ?Y.b6.+l'a.s+@n ?y.b6.+l'a.s+@n ?y:.b6.+l'a.s+@n ?y:.b6.+l'a.s+n y:.b6.+l'a.s+@n  >
< "Uberlassens >  < ?y:.b6.+l'a.s+@n#+s > <  ?@.b6.+l'a.s+@n#+s ?Y.b6.+l'a.s+@n#+s ?y.b6.+l'a.s+@n#+s ?y:.b6.+l'a.s+@n#+s ?y:.b6.+l'a.s+n#+s y:.b6.+l'a.s+@n#+s  >
< "Uberlaufen >  < ?'y:.b6#l''aU.f+@n > <  'y:.b6#l''aU.f+@n ?'Y.b6#l''aU.f+@n ?'y.b6#l''aU.f+@n ?'y:.b6#l''aU.f+@n  ><  ?y:.b6.+l'aU.f+@n > <    ?@.b6.+l'aU.f+@n   ?Y.b6.+l'aU.f+@n   ?y.b6.+l'aU.f+@n   ?y:.b6.+l'aU.f+@n   y:.b6.+l'aU.f+@n  >
< "Uberlaufens >  < ?'y:.b6#l''aU.f+@n#+s > <  'y:.b6#l''aU.f+@n#+s ?'Y.b6#l''aU.f+@n#+s ?'y.b6#l''aU.f+@n#+s ?'y:.b6#l''aU.f+@n#+s  ><  ?y:.b6.+l'aU.f+@n#+s > <    ?@.b6.+l'aU.f+@n#+s   ?Y.b6.+l'aU.f+@n#+s   ?y.b6.+l'aU.f+@n#+s   ?y:.b6.+l'aU.f+@n#+s   y:.b6.+l'aU.f+@n#+s  >
< "Uberleben >  < ?y:.b6.+l'e:.b+@n > <  ?@.b6.+l'e:.b+@n ?Y.b6.+l'e:.b+@n ?y.b6.+l'e.b+@n ?y.b6.+l'e:.b+@n ?y:.b6.+l'E.b+@n ?y:.b6.+l'e.b+@n ?y:.b6.+l'e:.b+@n y:.b6.+l'e:.b+@n  >
< "Uberlebens >  < ?y:.b6.+l'e:.b+@n#+s > <  ?@.b6.+l'e:.b+@n#+s ?Y.b6.+l'e:.b+@n#+s ?y.b6.+l'e.b+@n#+s ?y.b6.+l'e:.b+@n#+s ?y:.b6.+l'E.b+@n#+s ?y:.b6.+l'e.b+@n#+s ?y:.b6.+l'e:.b+@n#+s y:.b6.+l'e:.b+@n#+s  >
< "Uberlegen >  < ?y:.b6.+l'e:.g+@n > <  ?@.b6.+l'e:.g+@n ?Y.b6.+l'e:.g+@n ?y.b6.+l'e.g+@n ?y.b6.+l'e:.g+@n ?y:.b6.+l'E.g+@n ?y:.b6.+l'e.g+@n ?y:.b6.+l'e:.g+@n y:.b6.+l'e:.g+@n  >
< "Uberlegens >  < ?y:.b6.+l'e:.g+@n#+s > <  ?@.b6.+l'e:.g+@n#+s ?Y.b6.+l'e:.g+@n#+s ?y.b6.+l'e.g+@n#+s ?y.b6.+l'e:.g+@n#+s ?y:.b6.+l'E.g+@n#+s ?y:.b6.+l'e.g+@n#+s ?y:.b6.+l'e:.g+@n#+s y:.b6.+l'e:.g+@n#+s  >
< "Uberlegung >  < ?y:.b6.+l'e:.g+UN > <  ?@.b6.+l'e:.g+UN ?Y.b6.+l'e:.g+UN ?y.b6.+l'e.g+UN ?y.b6.+l'e:.g+UN ?y:.b6.+l'E.g+UN ?y:.b6.+l'e.g+UN ?y:.b6.+l'e:.g+UN y:.b6.+l'e:.g+UN  >
< "Uberlegungen >  < ?y:.b6.+l'e:.g+U.N#+@n > <  ?@.b6.+l'e:.g+U.N#+@n ?Y.b6.+l'e:.g+U.N#+@n ?y.b6.+l'e.g+U.N#+@n ?y.b6.+l'e:.g+U.N#+@n ?y:.b6.+l'E.g+U.N#+@n ?y:.b6.+l'e.g+U.N#+@n ?y:.b6.+l'e:.g+U.N#+@n y:.b6.+l'e:.g+U.N#+@n  >
< "Ubernachten >  < ?y:.b6.+n'ax.t+@n > <  ?@.b6.+n'ax.t+@n ?Y.b6.+n'ax.t+@n ?y.b6.+n'ax.t+@n ?y:.b6.+n'ax.t+@n y:.b6.+n'ax.t+@n  >
< "Ubernachtens >  < ?y:.b6.+n'ax.t+@n#+s > <  ?@.b6.+n'ax.t+@n#+s ?Y.b6.+n'ax.t+@n#+s ?y.b6.+n'ax.t+@n#+s ?y:.b6.+n'ax.t+@n#+s y:.b6.+n'ax.t+@n#+s  >
< "Ubernachtung >  < ?y:.b6.+n'ax.t+UN > <  ?@.b6.+n'ax.t+UN ?Y.b6.+n'ax.t+UN ?y.b6.+n'ax.t+UN ?y:.b6.+n'ax.t+UN y:.b6.+n'ax.t+UN  >
< "Ubernachtungen >  < ?y:.b6.+n'ax.t+U.N#+@n > <  ?@.b6.+n'ax.t+U.N#+@n ?Y.b6.+n'ax.t+U.N#+@n ?y.b6.+n'ax.t+U.N#+@n ?y:.b6.+n'ax.t+U.N#+@n y:.b6.+n'ax.t+U.N#+@n  >
< "Ubernachtungsgelegenheit >  < ?y:.b6.+n'ax.t+UN#+s#g@.+l''e:.g+@n.+haIt > <  ?@.b6.+n'ax.t+UN#+s#g@.+l''e:.g+@n.+haIt ?Y.b6.+n'ax.t+UN#+s#g@.+l''e:.g+@n.+haIt ?y.b6.+n'ax.t+UN#+s#g@.+l''e.g+@n.+haIt ?y.b6.+n'ax.t+UN#+s#g@.+l''e:.g+@n.+haIt ?y:.b6.+n'ax.t+UN#+s#g@.+l''E.g+@n.+haIt ?y:.b6.+n'ax.t+UN#+s#g@.+l''e.g+@n.+haIt ?y:.b6.+n'ax.t+UN#+s#g@.+l''e:.g+@n.+haIt y:.b6.+n'ax.t+UN#+s#g@.+l''e:.g+@n.+haIt  >
< "Ubernachtungsgelegenheiten >  < ?y:.b6.+n'ax.t+UN#+s#g@.+l''e:.g+@n.+haI.t#+@n > <  ?@.b6.+n'ax.t+UN#+s#g@.+l''e:.g+@n.+haI.t#+@n ?Y.b6.+n'ax.t+UN#+s#g@.+l''e:.g+@n.+haI.t#+@n ?y.b6.+n'ax.t+UN#+s#g@.+l''e.g+@n.+haI.t#+@n ?y.b6.+n'ax.t+UN#+s#g@.+l''e:.g+@n.+haI.t#+@n ?y:.b6.+n'ax.t+UN#+s#g@.+l''E.g+@n.+haI.t#+@n ?y:.b6.+n'ax.t+UN#+s#g@.+l''e.g+@n.+haI.t#+@n ?y:.b6.+n'ax.t+UN#+s#g@.+l''e:.g+@n.+haI.t#+@n y:.b6.+n'ax.t+UN#+s#g@.+l''e:.g+@n.+haI.t#+@n  >
< "Ubernachtungskosten >  < ?y:.b6.+n'ax.t+UN#+s#k''Os.t#+@n > <  ?@.b6.+n'ax.t+UN#+s#k''Os.t#+@n ?Y.b6.+n'ax.t+UN#+s#k''Os.t#+@n ?y.b6.+n'ax.t+UN#+s#k''Os.t#+@n ?y:.b6.+n'ax.t+UN#+s#k''Os.t#+@n y:.b6.+n'ax.t+UN#+s#k''Os.t#+@n  >
< "Ubernachtungsm"oglichkeit >  < ?y:.b6.+n'ax.t+UN#+s#m''2:k.+lIC.+kaIt > <  ?@.b6.+n'ax.t+UN#+s#m''2:k.+lIC.+kaIt ?Y.b6.+n'ax.t+UN#+s#m''2:k.+lIC.+kaIt ?y.b6.+n'ax.t+UN#+s#m''2:k.+lIC.+kaIt ?y.b6.+n'ax.t+UN#+s#m''2k.+lIC.+kaIt ?y:.b6.+n'ax.t+UN#+s#m''2:k.+lIC.+kaIt ?y:.b6.+n'ax.t+UN#+s#m''2k.+lIC.+kaIt y:.b6.+n'ax.t+UN#+s#m''2:k.+lIC.+kaIt  >
< "Ubernachtungsm"oglichkeiten >  < ?y:.b6.+n'ax.t+UN#+s#m''2:k.+lIC.+kaI.t#+@n > <  ?@.b6.+n'ax.t+UN#+s#m''2:k.+lIC.+kaI.t#+@n ?Y.b6.+n'ax.t+UN#+s#m''2:k.+lIC.+kaI.t#+@n ?y.b6.+n'ax.t+UN#+s#m''2:k.+lIC.+kaI.t#+@n ?y.b6.+n'ax.t+UN#+s#m''2k.+lIC.+kaI.t#+@n ?y:.b6.+n'ax.t+UN#+s#m''2:k.+lIC.+kaI.t#+@n ?y:.b6.+n'ax.t+UN#+s#m''2k.+lIC.+kaI.t#+@n y:.b6.+n'ax.t+UN#+s#m''2:k.+lIC.+kaI.t#+@n  >
< "Ubernachtungspreis >  < ?''y:.b6#n'ax.t+UN#+s#pr''aIs > <  ''y:.b6#n'ax.t+UN#+s#pr''aIs ?''Y.b6#n'ax.t+UN#+s#pr''aIs ?''y.b6#n'ax.t+UN#+s#pr''aIs ?''y:.b6#n'ax.t+UN#+s#pr''aIs  >
< "Ubernachtungspreise >  < ?''y:.b6#n'ax.t+UN#+s#pr''aI.z#+@ > <  ''y:.b6#n'ax.t+UN#+s#pr''aI.z#+@ ?''Y.b6#n'ax.t+UN#+s#pr''aI.z#+@ ?''y.b6#n'ax.t+UN#+s#pr''aI.z#+@ ?''y:.b6#n'ax.t+UN#+s#pr''aI.z#+@  >
< "Ubernachtungspreisen >  < ?''y:.b6#n'ax.t+UN#+s#pr''aI.z#+@n > <  ''y:.b6#n'ax.t+UN#+s#pr''aI.z#+@n ?''Y.b6#n'ax.t+UN#+s#pr''aI.z#+@n ?''y.b6#n'ax.t+UN#+s#pr''aI.z#+@n ?''y:.b6#n'ax.t+UN#+s#pr''aI.z#+@n  >
< "Ubernachtungspreises >  < ?''y:.b6#n'ax.t+UN#+s#pr''aI.z#+@s > <  ''y:.b6#n'ax.t+UN#+s#pr''aI.z#+@s ?''Y.b6#n'ax.t+UN#+s#pr''aI.z#+@s ?''y.b6#n'ax.t+UN#+s#pr''aI.z#+@s ?''y:.b6#n'ax.t+UN#+s#pr''aI.z#+@s  >
```

E the awk-Scripts

E.1 blockNumbering.awk

```
########################################################################
#
# blockNumbering.awk
#
# 25.06.99
# Ana Paula Quirino Simoes
#
# Project - X-Vario VERBMOBIL
#
# This script numbers the blocks of the different output files of Vario
# so that the output files can be merged in one single file and
# then be sorted.
# One blank line numbered 1 is added at the beginning of the file.
#
# gawk -f "blockNumbering.awk" "inputfile.txt" > "outputfile.txt"
#
#########################################################################

BEGIN { block = 1 }

{ if ($1==/^\n/) (block = block +1)
                   gsub(/bye\./,"")
                   gsub(/Opening file \/project\/xfst\/Tmp\/lineSplitted\.txt\.\.\./,"")
                   gsub(/Closing file \/project\/xfst\/Tmp\/lineSplitted\.txt\.\.\./,"")
#                  gsub(/Opening*/,"")
#                  gsub(/Closing*/,"")
                   print block, " "  $0

#        else (print block, " " $0)
}
```

E.2 lineAdjust.awk

```
########################################################################
#
# lineAdjust.awk
#
# 10.08.99
# Ana Paula Quirino Simoes
```

```
#
# Project - X-Vario VERBMOBIL
#
# This script adds one blank line to OrthPretty.txt, so that the
# number of lines coincide when using 'paste'.
#
# gawk -f "lineAdjust.awk" "inputfile.txt" > "outputfile.txt"
#
##########################################################################

BEGIN { printf "\n"; }

{ print $0; }
```

E.3 lineSplitt.awk

```
#########################################################################
#
# lineSplit.awk
#
# 19.05.99
# Ana Paula Quirino Simoes
#
# Project - X-Vario VERBMOBIL
#
# This script generates one blank line between speech variants
# originally contained in the LexDB, where they are separated by
# a semicolon.
#
# gawk -f "lineSplit.awk" "inputfile.txt" > "outputfile.txt"
#
# Important!!! This script shall be run after twoLines.awk.
#
##########################################################################

{ gsub(/;/,"\n \n ");
  print $0;   }
```

E.4 noNumbers.awk

```
#########################################################################
```

```
#
# noNumbers.awk
#
# 29.06.99
# Ana Paula Quirino Simoes
#
# Project - X-Vario VERBMOBIL
#
# This script deletes the first column of the output of the sort operation.
#
# gawk -f "noNumbers.awk" "inputfile.txt" > "outputfile.txt"
#
##########################################################################


{ #gsub(/\n/,"")
  print  $2; }
```

E.5 noParenth.awk

```
#########################################################################
#
# noParenth.awk
#
# 09.08..99
# Ana Paula Quirino Simoes
#
# Project - X-Vario VERBMOBIL
#
# This script deletes empty brackets.
#
# gawk -f "noParenth.awk" "inputfile.txt" > "outputfile.txt"
#
##########################################################################

{ gsub(/Opening/,""); }

(!/<  >/&&!/<   >/)  { print $0; }
```

E.6 noVariantMarking.awk

```
########################################################################
#
# noVariantMarking.awk
#
# 29.06.99
# Ana Paula Quirino Simoes
#
# Project - X-Vario VERBMOBIL
#
# This scripts deletes the marking of the variants originally contained
# in the LexDB.
#
# gawk -f "noVariantMarking.awk" "inputfile.txt" > "outputfile.txt"
#
#########################################################################

{ gsub(/%%Sprachvariante%%/,"  ");
  print $0;
}
```

E.7 noVariantsOneLine.awk

```
########################################################################
#
# noVariantsOneLine.awk
#
# 29.06.99
# Ana Paula Quirino Simoes
#
# Project - X-Vario VERBMOBIL
#
# This scripts deletes the marking of the speech variants
# and prints all variants in one line.
#
# gawk -f "noVariantsOneLine.awk" "inputfile.txt" > "outputfile.txt"
#
#########################################################################



#{ gsub(/Opening/,"");
#  gsub(/<  Closing  >/,"");
#  gsub(/<   >/,"");
#}
```

```
#/\%\%Sprachvariante\%\%/ { gsub(/%%Sprachvariante%%/,"  ");
#                            printf ("%s", $0);
#                          }

!/\%\%Sprachvariante\%\%/ { gsub(/Opening/,"");
                            gsub(/<  Closing  >/,"");
                            gsub(/<   >/,"");
                            printf ("%s %s", "\n", $0); }

/\%\%Sprachvariante\%\%/ { gsub(/%%Sprachvariante%%/,"  ");
                           printf ("%s", $0);
                         }
```

E.8 oneLine.awk

```
##########################################################################
#
# oneLine.awk
#
# 28.07.99
# Ana Paula Quirino Simoes
#
# Project - X-Vario VERBMOBIL
#
# This script prints all columns of one block in one line.
#
# Aufruf: gawk -f "oneLine.awk" "inputfile.txt" > "outputfile.txt"
#
###########################################################################

BEGIN { block[NR]; }


/^.+/ {  gsub(/ Opening /,"");
         gsub(/<  Closing  >/,"");
 #       gsub(/<   >/,"");
         block[NR] = $0;  }


#/^.+/ {  block[NR] = NR;  }


!/^.+/ {  block[NR] = " >\n< "; }
```

```
END   {   printf ("<  ");
           for(i=1;block[i];i=i+1)
             {  printf ("%s ", block[i]);  }
          printf (" > \n \n");
       }
```

E.9 prettyPhonseg.awk

```
#########################################################################
#
# prettyPhonseg.awk
#
# 29.07.99
# Ana Paula Quirino Simoes
#
# Project - X-Vario VERBMOBIL
#
# This script prints  the entries of the Phonseg.txt in brackets.
#
# gawk -f "prettyPhonseg.awk" "inputfile.txt" > "outputfile.txt"
#
##########################################################################


!/^\n$/ { print "< " $0 " >" }

/^ \n$/ { }

/^ +\n$/ { }
```

E.10 twoLines.awk

```
#########################################################################
#
# twoLines.awk
#
# 19.05.99
# Ana Paula Quirino Simoes
#
# Project - X-Vario VERBMOBIL
#
# This script generates two blank lines between the entries
```

```
# of the input file for X-Vario (in the LexDB-format), so that the blocks
# of generated speech variants can be separated from each other
#
# The strings which do not belong to the Phonseg.txt
# are deleted by this script.
#
# gawk -f "twolines.awk" "inputfile.txt" > "outputfile.txt"
#
# Importatnt!!! This script shall be run before lineSplit.awk.
#
##########################################################################


{ gsub(/A /,"")
  gsub(/Opening/,"")
  gsub(/BIphonseg/,"")
  gsub(/Harald/,"")
  gsub(/Orth.Pointers.db/,"")

  print $0;
  print  "\n \n";   }
```

E.11 variantsMarking.awk

```
#########################################################################
#
# variantsMarking.awk
#
# 29.06.99
# Ana Paula Quirino Simoes
#
# Project - X-Vario VERBMOBIL
#
# Script to mark the speech variants present in the LexDB
#
# gawk -f "variantsMarking.awk" "inputfile.txt" > "outputfile.txt"
#
##########################################################################

/^ .+/ { gsub(/ /,"%%Sprachvariante%%");}
  {print $0;}
```

F The *XFST*-scripts

F.1 test01.script

```
# Script for the automatic generation of speech variants
# Input had to be in the MPARA-format

load /homes/vmobil/VM/SOFTWARE/XFST/Rules/testregel21to23+27+29to30.fst

apply down <  /homes/vmobil/VM/DATEN/XFST/Tmp/lineSplitted.txt

quit
```

F.2 test02.script

```
# Script for the automatic generation of speech variants
# Input had to be in the MPARA-format

load /homes/vmobil/VM/SOFTWARE/XFST/Rules/testregel01to11.fst

apply down <  /homes/vmobil/VM/DATEN/XFST/Tmp/lineSplitted.txt

quit
```

F.3 test03.script

```
# Script for the automatic generation of speech variants
# Input had to be in the MPARA-format

load /homes/vmobil/VM/SOFTWARE/XFST/Rules/testregel12to20.fst

apply down <  /homes/vmobil/VM/DATEN/XFST/Tmp/lineSplitted.txt

quit
```

F.4 test04.script

```
# Script for the automatic generation of speech variants
# Input had to be in the MPARA-format

load /homes/vmobil/VM/SOFTWARE/XFST/Rules/testregel31+33to34+36+24.fst

apply down <   /homes/vmobil/VM/DATEN/XFST/Tmp/lineSplitted.txt

quit
```