TR-IT-0236

German in Eight Weeks - A Crash Course for CHATR

Caren Brinckmann

September 1997

With all its different modules, CHATR is an extremely big system. If the output doesn't sound quite right, there are always several ways of improvement. As an example of what can be achieved within eight weeks, this technical report describes how to improve the German voice of CHATR, focussing on the database, the lexicon and the prosody prediction.

Contents

Chapter 1

Introduction

In the summer of 1997, I stayed at ATR Interpreting Telecommunications Research Laboratories for eight weeks. This internship was the main part of the 1997 REES Programme (Research Experience for European Students), which was organized (and sponsored) by JISTEC on the Japanese side and the "Studienstiftung des deutschen Volkes" on the German side.

Eight weeks is a tremendously short time to do research, and it's even shorter when you have to get know the ropes within such a big system as CHATR. So, together with Kristina Striegnitz, with whom I collaborated the whole time, I could only take first steps to make the German voice(s) of CHATR more intelligible and natural. Implementation of different intonation models and thorough comparative testing (with naive subjects) is definitely necessary, but lay beyond what could be achieved during eight weeks.

Therefore, this technical report is aimed at future researchers to help them work with CHATR (not necessarily only the German part), so that they hopefully can continue our work and further improve CHATR's (German) prosody, which is still quite rudimentary.

While still in Germany, I received my research plan for my stay at ITL, which was entitled "German Prosody Prediction for Speech Synthesis". As I am involved in the Speech Synthesis project at the Institute of Phonetics, University of the Saarland, where we are currently also working on the prosody prediction and realization, I was very excited about the opportunity to work in the same field, but with a different speech synthesis system.

Of course, I expected that I would need some time to get into CHATR, but when I played around with CHATR during my first week at ITL, I noticed that half of the German utterances were unintelligible. Of course, I could have proceeded as planned, working on the prosody only. But who will notice a nice F0 contour, when you cannot understand what is being said? That's why I decided to improve the German voice of CHATR on the segmental level first, in order to make it speak intelligible German. After that, I worked on the prosody prediction.

There are as many ways to improve CHATR as there are modules. For a general introduction to CHATR as well as an overview of the system architecture, please refer to the manual [Weeks 97], especially the part concerning the processing sequence.

In this report, I will describe in detail my work concerning

- the German database,
- the German lexicon, and
- the prosody prediction for German.

During the whole time of my stay at ITL, I worked closely together with Kristina Striegnitz. Since Kristina was the one who struggled with the actual CHATR code, while I worked with the CHATR Lisp Interface only, please refer to [Striegnitz 97] for implementation details concerning the C code of CHATR.

As I had mainly worked with PCs before coming to ATR, it was during this summer that I wrote my first awk-script, and even this report is only the second LaTeX document (after the slides for my Final Talk) written by myself. That's why I included many Unix commands in this report that were necessary for my work with CHATR, just in case any future reader has about the same knowledge about Unix machines and programs as I had before. For those of you who think that this is kindergarten stuff: please don't be annoyed, but kindly skip those paragraphs which are aimed at the more computer-unminded readers.

Chapter 2

Database

To get a first impression of what the German voice of CHATR sounded like when I arrived at ITL, please listen to the examples on

http://www.itl.atr.co.jp/ ~xcaren/before_after.html.

Of some utterances the rhythm only is somewhat unnatural (but native speakers can still understand what is being said), while other sentences are completely unintelligible.

In this chapter I will describe

- my diagnosis of CHATR's German output, and
- my treatment concerning the database.

2.1 Diagnosing the Output

2.1.1 Symptoms

Unit Selection

When you listen to what CHATR (version 0.92, speaker_KKO) makes out of your command
`(SayText "Mein juengster Sohn moechte Innenarchitekt werden.")`, you might wonder why the word "Innenarchitekt" (interior designer) gets so distorted. To have a quick look at which units CHATR chose from the database to synthesize the utterance, you don't have to start the whole `(Inspect)` environment. Simply type `(Save UnitLabels+ '-)` and something like the following will appear on your screen:

...

```
("/dept2/work16/pi/data/chatr_dbs/KKO/wav/KKO182.wav" 2053 37 1
   (  I   37   67)       ;;   6IC        ; a:b6ICm9
   )
```

```
("/dept2/work16/pi/data/chatr_dbs/KK0/wav/KK0118.wav" 6995 60 1
   ( n   60   81)       ;;  InaU        ; nu:6InaUks
   )
("/dept2/work16/pi/data/chatr_dbs/KK0/wav/KK0153.wav" 5244 138 2
   ( 6   84   38)       ;;   d6n        ; ve:d6na:x
   ( n   54   26)       ;;  6na:        ; e:d6na:xm
   )
("/dept2/work16/pi/data/chatr_dbs/KK0/wav/KK0177.wav" 2146 105 1
   ( a6  105   59)      ;;  ka6t        ; fa:6ka6t@n
   )
("/dept2/work16/pi/data/chatr_dbs/KK0/wav/KK0107.wav" 2554 86 1
   ( C   86   47)       ;;  U6CU        ; ndU6CUnt
   )
("/dept2/work16/pi/data/chatr_dbs/KK0/wav/KK0167.wav" 2006 69 1
   ( i:   69   56)      ;;  zi:m        ; nnzi:mi:6b
   )
("/dept2/work16/pi/data/chatr_dbs/KK0/wav/KK0150.wav" 1611 62 1
   ( t   62   22)       ;;  i:ts        ; t@i:tse:n
   )
("/dept2/work16/pi/data/chatr_dbs/KK0/wav/KK0012.wav" 968 43 1
   ( E   43   37)       ;;   tEs        ; ge:tEsnI
   )
("/dept2/work16/pi/data/chatr_dbs/KK0/wav/KK0187.wav" 2364 79 1
   ( k   79   40)       ;;   Ekf        ; i:6vEkfa:r
   )
("/dept2/work16/pi/data/chatr_dbs/KK0/wav/KK0117.wav" 1301 45 1
   ( t   45   22)       ;;   ktf        ; rEktf0n
   )
```

...

The column in the middle tells you the original segmental context of the selected units. For example, unit "a6" has "k" as left and "t" as right context, because it was realized in the word "Fahrkarten". What we really need is an "n" (or at least something alveolar) as left context and a "C" (or something palatal) as right context. Because CHATR cannot find ideal units, you can still hear the "k" in the "a6"-unit of this particular example.

You might also wonder why the schwa in "Innen-" is a "6" instead of a regular schwa ("@"). This is in fact a problem of the lexicon, which will be addressed in Chapter 3.

Unit Labels: stressed vs. unstressed

CHATR's output of `(SayText "die Biene")` is also quite unintelligible. Looking at the selected units, we get:

```
("/dept2/work16/pi/data/chatr_dbs/KK0/wav/KK0039.wav" 548 130 1
```

```
   (  d  130   58)       ;;  ##da        ; ##daNk
   )
("/dept2/work16/pi/data/chatr_dbs/KKO/wav/KKO083.wav" 932 138 2
   ( i:   64  130)       ;;  di:b        ; Isdi:bIl
   (  b   74   76)       ;;  i:bI        ; sdi:bIlI
   )
("/dept2/work16/pi/data/chatr_dbs/KKO/wav/KKO127.wav" 1863 137 2
   ( i:  107  135)       ;;  ti:n        ; sIti:na:x
   (  n   30   34)       ;; i:na:        ; Iti:na:xh
   )
("/dept2/work16/pi/data/chatr_dbs/KKO/wav/KKO061.wav" 1663 62 1
   (  6   62   38)       ;;   n6r        ; Oan6r@m
   )
("/dept2/work16/pi/data/chatr_dbs/KKO/wav/KKO162.wav" 0 613 1
   ( ##  613  600)       ;;   ##f        ; ##fa:r
   )
```

Again, the context is not always right. But this time I want to focus on the difference between the two "i:" units.

In German, as in many other languages, an unstressed realization of any vowel is generally shortened and more centralized compared to the stressed variant. On the phonemic level, both realizations would be transcribed the same, of course. However, if two units have the same label, CHATR cannot distinguish between them.

In this particular example, the "i:" for the German article "die" comes from an actual realization of "die", so there's no problem. On the other hand, the "i:" in "Biene" needs to be a long, stressed realization of that vowel. But the selected unit comes from "Intercity", where the last vowel is unstressed and therefore shortened and centralized.

2.1.2 Root Causes

Unit Selection

Why does CHATR eventually choose those "unfit" units? Looking at the database, I found out that there were only about 9,000 units for speaker KKO (as opposed to approximately 30,000 units for most of the japanese speakers). As CHATR weighs continuity distance against unit distance, it happens quite often with small databases, that there's no unit in the database that fulfills both the segmental and the prosodic requirements that have been predicted.

In theory, with an infinite database, the actual unit selection is no problem, because there will be enough units to choose from in any given situation. In order to come close to this theoretical paradise, the database should be as big as possible.

Unit Labels

Theoretically, the unstressed and the stressed realizations of a certain vowel should be distinguishable by their duration. Unfortunately, CHATR is not very good at duration, yet. So in fact, you could attribute the problem of the unit labels to the "bad" unit selection, too.

From a more practical point of view, the problem is caused by a too phonemic labelling. Therefore, in order to be able to distinguish between stressed and unstressed realizations of a vowel, the labels should be different.

2.2 Treatment: Building a new Database

2.2.1 Gathering Speech Material

The units for speaker KKO (Prof. Klaus Kohler) came from the "Kiel Corpus of Read Speech Vol. I" (also known as PhonDat 1 and 2). Apart from the actual speech files, the following information is available:

- orthographic representation of the text,
- canonical transcription,
- segmental labels, and
- canonical and variants lexica.

For further information, please refer to

http://www.phonetik.uni-muenchen.de/Bas/BasPD1eng.html, and
http://www.phonetik.uni-muenchen.de/Bas/BasPD2eng.html.

Unfortunately, the same speaker is abbreviated as "k61" and "kko" for PhonDat 1 and PhonDat 2 respectively. That's why for the KKO database only PhonDat 2 material (approx. 17 min) was used. The improved database (called kko) is based on PhonDat 1 and 2 (together approx. 45 min), which gives CHATR three times more units to choose from.

In addition to the male speaker, a female speaker (called rtd) also read the whole corpus. So we now have two German voices within CHATR.

For all the necessary commands to build a new database, please refer to Appendix A, and the section "Making a Speech Synthesizer Database" of the CHATR manual ([Weeks 97]).

2.2.2 Fixing the Labels

The "Kiel Corpus of Read Speech" is already segmented and labelled, as mentioned in section 2.2.1 above. For the labelling conventions of the corpus, please refer to

`http://www.phonetik.uni-muenchen.de/Bas/BasFormatseng.html#S1`.

Of course, we don't want to label all the speech data again, so we have to take a close look at the already existing label files and convert them into a format which is useful to CHATR.

Problems

1. "mixed" transcription:
   - mostly phonemic transcription
   - phonetic transcription of aspiration and glottalization
   - punctuation labels
   - transcription of elisions and replacements
2. phonemic transcription: As mentioned in section 2.1.1, using the same label for stressed and unstressed (= shortened and centralized) realizations of the same vowel can cause some problems.

Solutions

1. (careful) substitution/deletion of unnecessary labels
2. new phonemeset "sampaG", which consists of all standard SAMPA labels for German plus extra labels for (canonically) stressed phonemes.

See Appendix C.1 for the awk-scripts written for these tasks and their detailed explanation.

In Figure 2.1 you can see an example of the original transcription compared to the new sampaG transcription.

2.2.3 Training

By now we have

- sufficient speech material,
- corrected label files, and
- a new phonemeset (sampaG_def.ch).

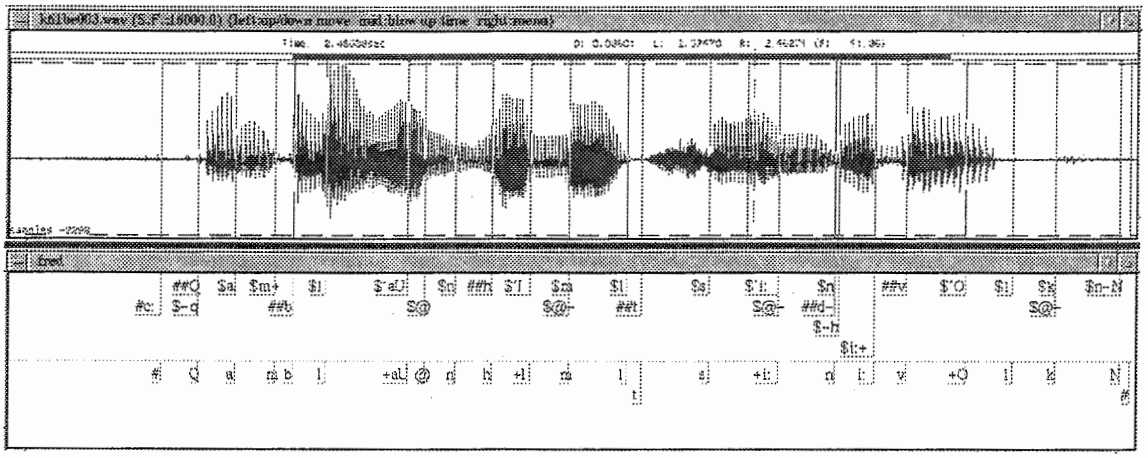

Figure 2.1: upper row: original transcription, lower row: sampaG

What you still have to specify, is

- the database description file,
- the database parameters file (with information about the phoneme-set, clustering of phones in the database, parameter weights, lexicon, intonation, duration parameters), and
- the training parameters file.

Please refer to Appendix A.3 and [Weeks 97] for further instructions.

Finally, after having run the whole training process (which might actually take a few days, because nothing ever runs smoothly), you get

- a full database index,
- unit descriptions for every phoneme (unit) in the database,
- pitch marks,
- acoustic parameter files,

and a lot more...

Specify the directory of the new database in your .chatrrc:

```
(defspeaker "<database_name>" "<database_directory>")
```

After that, you should be able to call the new speaker within CHATR by simply typing `(speaker_<database_name>)`. Now enjoy the new voice (and all its flaws)!

Chapter 3

Lexicon

Since CHATR has no module to do morphological decomposition so far, we simply look up every word of the Input (input format `(SayText "...")` or `(Say (Synth (Utterance Text "..."))))` in a lexicon, which therefore has to contain not only lemmas but all possible wordforms. That lexicon has to be in a specific CHATR format, of course (see Figure 3.1).

CompLexicon
sampaG
("Mordversuchen" (((m +O6 t) (1)) ((f E6) (0)) ((z u:) (0)) ((x @ n) (0)))
((CAT N)))

Figure 3.1: minimal CHATR lexicon (one entry)

3.1 Source

The only German lexicon that was electronically available at ITL, is the German part of the "CELEX Lexical Database" [CELEX 95] (currently in `/DB/LTDB4/celex_rel_2/german/`).

The CELEX databases used for the current version of the sampaG lexicon of CHATR are:

- gpw.cd (German Phonology, Wordforms), and
- gsl.cd (German Syntax, Lemmas).

Example entries:

gpw.cd: `312069\Mordversuchen\0\25432\'mOrt-fEr-zu-x@n\...`

gsl.cd: `25432\Mordversuch\12\1\1\\N\N\\`

3.2 Problems & Solutions

In the following sections, I will describe some minor problems concerning the lexicon and their solution. For a detailed description of how to actually build a German CHATR lexicon, please refer to Appendix B.

3.2.1 Phonemic Transcription

Are x and C allophonic?

In the German CELEX database, the fricatives "x" and "C" (SAMPA transcription) are treated as allophonic, which is probably true for the words in the database. But since x and C are only allophones on a morphemic level (famous example: "Kuchen" vs. "Kuhchen"), we certainly want to have two different units in the CHATR database. That's why the original CELEX "x" gets replaced by a "C", unless it follows a back vowel.

@r

Since CELEX transcribes the words phonemically, e.g. "Mutter" has "mUt@r" as transcription. However, if "@r" is followed by a consonant or a syllable boundary, it is realized as "6" in normal speech. That's why we replace it according to this rule.

3.2.2 Orthography

The previous German CHATR lexicon (which was also based on CELEX), only had entries starting with lower-case letters. But since capitalization can also distinguish between entries with different part-of-speech, we want to keep that extra information in the lexicon.

3.2.3 Part-of-speech Information

The prosody prediction rules (see Chapter 4 and [Striegnitz 97]) for simple Text Input need some basic information about the part-of-speech of each word. Since this is not available from `gpw.cd` but from `gsl.cd` (via the LemmaId), we have to combine both sources.

3.2.4 Double Entries

This is actually a problem not yet solved. Entries that are *exactly* the same can obviously be deleted from the lexicon very easily. But what is to be done with entries that differ only in their

- pronunciation, or
- part-or-speech

(or possibly both)?

Since CHATR only looks up word by word in the dictionary, it takes the item that comes first in the lexicon and doesn't search any further. As a first step, only the (heuristically) most probable entry should remain in the lexicon, but this hasn't been implemented yet.

3.3 One Speaker, Many Languages

I order to have several different speakers for one particular language, you simply (?!) have to make new databases. Since CHATR is a multi-lingual system, the other way round is also possible: Let one speaker speak several different languages!

For example, there's already an English lexicon available within CHATR, so why not use it for your new speaker?

If you try to use the Englisch "beep" lexicon, just by typing:

```
(setup_beep_lex)
```

you won't be very successful, because CHATR will complain if you try to make it speak English with:

```
(SayText "My English is very good.")
```

What is missing, is the phoneme mapping between "beep" and "sampaG". If you

```
(load '~xcaren/CHATR/all_map.ch)
```

then the (German) speaker will be able to speak English.

In Appendix B.4 you can see, that the English diphtong "ow" (which doesn't exist in German) will be mapped to the German monophthong "o:" for example. By using the German phonemeset, the speaker will have a German accent in any foreign language, of course.

Chapter 4

Prosody Prediction

After improving the database and the lexicon, the two new German CHATR speakers kko and rtd were now almost perfect on the segmental level. However, the intonation was still a bit strange. Actually, a japanese (JToBI) Linear Regression model (`lib/data/mht_lrf0.ch`) was used for the German speakers, too.

Obviously, Japanese and German intonation don't have very much in common. Therefore, the next thing we tried, was a LR model for English (`lib/data/f2b_lrf0.ch`). Unfortunately, that model was trained on a corpus of read news, where rising intonation was very rare. That's why it was almost impossible to get a natural intonation for questions, even if a high boundary tone was predicted.

Finally we resolved not to use a trained LR model, but the so-called APL model based on [Anderson 84]. This model realizes previously predicted ToBI tones as F0 target points, taking into account speaker range and parameters. Then it converts the set of target points to a sampled F0 at desired rate by smoothing the target points.

What we had to do now, is predict those ToBI tones and prosodic phrases

- directly from the Text Input, and/or
- from the output of the TDMT translation system.

In the following, I will give a very short overview of the theories that can be used for the prediction of prosodic phrases and intones (for a description of the German ToBI labels, see [Reyelt 96]). The syntax based approach of [Isačenko 64] will be described in more detail, as well as the phrasing and intonation rules for the two different input formats. For the actual implementation, please refer to [Striegnitz 97].

4.1 Theories

4.1.1 Focus-based Approaches

If a human being doesn't know the meaning of the text that he or she has to read aloud, often the prosodic realization is the first thing that suffers. The wrong words become accented, or the global intonation contour is not correct. So, how should a Text-to-Speech system, that doesn't know anything about the semantic content of a particular sentence, produce an output with a suitable intonation?

One possibility is, that the user gives the system information about the topic-focus structure of the utterance in question. For example, [Féry 92] and [Uhmann 91] both analyze the topic-focus-structure of German sentences and the resulting sequence of ToBI tones and breaks.

4.1.2 Syntax-based Approaches

In [Bierwisch 66] information about the syntactic tree structure of a given sentence is used to predict accents and intonation phrases. [Stock 82] use part-of-speech information to form so-called "accent groups" within a sentence, and predict the placement of accents within those groups.

[Isačenko 64] carried out experiments concerning the relation between the pitch accents (called "tone switches") of an utterance and the perceived sentence type. If you "translate" their tone switches to ToBI labels, the following rules can be derived:

rules concerning nuclear accent

1. L*+H as last pitch accent and a high boundary tone → question
2. L+H* as last pitch accent and a high boundary tone (= continuation rise) → "unfinished" statement
3. L* or H+L* as last pitch accent and a low boundary tone → complete statement
4. H* as last pitch accent and a low boundary tone → "contrastive" statement (focus on the nuclear accented word)

It doesn't matter which pitch accents are used before the last one.

rules concerning boundary tones

1. two words are seperated by two different tones (namely the boundary tones H- and L- or L% called "interrupter") → the two words belong to different phrases

2. no boundary tones between two words → words belong to the same phrase

A sentence consists of one or more phrases. A phrase contains at least one pitch accent and is separated from other phrases by boundary tones.

rules concerning focus

1. the word with the nuclear accent (last pitch accent) is the one that is focussed
2. "main stress" = last pitch accent of the whole sentence,
   "secondary stress" = last pitch accent of a phrase or (if the sentence consists of only one phrase) the penultimate pitch accent of the sentence
3. if a word carries two tones (namely L+H* and a low tone immediately after the accented syllable), it is contrastively focussed
4. the neutral position for "main stress" is the right-most accentable word, if shifted to the left, the last pitch accented word gets contrastive focus

rules concerning the intonation of lists

1. each list item can either have the intones (L*+H H-) or (L+H* H-) (but the same for each item of one particular list)
2. emotional connotations: (H* L-) on the list items
3. in natural speech it is possible (?? usual!!) to gradually lower the pitch of each listed item

4.1.3 Syllabification and Lexical Stress

Instead of looking up every word in a lexicon, it might be desirable to predict the pronunciation with

- letter-to-sound rules,
- syllabification rules, and
- rules concerning lexical stress.

[Féry 95] applies Optimality Theory to German stress patterns and can thus predict syllable boundaries and lexical stress. It would be at least interesting to implement her theoretic framework and apply it to those words, that are not in the lexicon. But then we would need reliable letter-to-sound rules, as well.

4.2 Possibilities

4.2.1 Available Information

If CHATR has to synthesize an utterance from Text Input, following information is available from the lexicon:

- phonemic transcription,
- syllable boundaries,
- lexical stress, and
- part-of-speech.

In addition, we know about the punctuation within each sentence.

If the input is in the TDMT format (output of the translation system), then we have extra information about

- topological fields, and
- sentence type.

Therefore, we don't have to predict syllable boundaries and lexical stress, but get the information via lexicon look-up instead.

So far, we don't have any knowledge about the topic-focus structure of any given sentence, nor its syntactic tree structure. That's why we based our prosody prediction on

- [Isačenko 64],
- [Stock 82], and
- common knowledge about phrasing.

In the following, the rules used for the prosody prediction of TDMT Input and Text Input respectively are specified.

4.2.2 TDMT Input

We have four different levels of phrasing: Discourse (D), Sentence (S), Clause (C), and Phrase (P), all of which are associated with specified pauses at the end of the respective phrase. E.g. in the <database_name>_synth.ch you can put in a definition like:

```
(Stats Pause
(
(discourse  500)
(sentence   500)
(clause     200)
(phrase     10)
))
```

The phrases are used like intermediate phrases, and the clauses like intonation phrases (of the ToBI system). That means that a phrase has to contain at least one pitch accent and ends with a H- or a L- as boundary tone. A clause consists of at least one phrase and ends with a X-Y% as boundary tone.

First, we have to group the words of a sentence into phrases, the phrases into clauses, the clauses into sentences, which constitute a discourse. (We could take every line of the tdmt-input as a new phrase. But then we would generate too many intones which would make CHATR sound very unstable.)

Following rules are applied:

1. put a sentence boundary whenever you encounter one of the following four: . ? ! ;

2. put a clause boundary between a MAINCLAUSE and a SUBORDCLAUSE (see key to abbreviations below) - or the other way round

3. put a clause boundary between two different MAINCLAUSes

4. if there isn't one yet, put a clause boundary whenever you encounter: :

5. if there isn't one yet, put a phrase boundary whenever you encounter: , (for example between two SUBORDCLAUSes)

6. every tdmt-line that contains a NOMEN (noun) or an ADJEKTIV (adjective) builds a phrase together with all its preceding tdmt-lines (that don't contain NOMEN or ADJEKTIV)

7. The lines after the last NOMEN/ADJEKTIVE line don't build their own phrase but belong to the previous phrase.

Having piled up a series of phrases, we have to decide which intones to take...

1. last phrase before a sentence-boundary:

   - SATZ, IMP, DES: H+L* as pitch accent and L-L% on last syllable
   - FRAGE: L*+H as pitch accent and H-H% on last syllable

2. last phrase before a clause-boundary: L+H* as pitch accent and H-L% on last syllable

3. all other phrases: H* as pitch accent and L- on last syllable

...and where to put the pitch accent:

1. go through the phrase from right to left

2. check the POS of every word

3. if it is NOMEN or ADJEKTIV, put the pitch accent on that word and stop searching

4. if you cannot find any NOMEN or ADJEKTIV in the whole phrase, look for an ADVERB, VERBZUSATZ, KARDINALZAHL, VERB, FIX-EXP, or PRAEPOSITION (in that order! always going from right to left through the whole phrase), put the pitch accent on the respective word and stop searching

5. if 3) and 4) fail, put the pitch accent on the last word in the phrase.

Either before or after building up phrases and deciding on intones, we have to put a stress mark on the syllable that carries lexical stress. But only for words with certain part-of-speech: VERB, VERBZUSATZ, NOMEN, ADJEKTIV, FRAGEADVERB, INTERROGATIVPRONOMEN, KARDINALZAHL, ORDINALZAHL, ADVERB, FIX-CAP, FIX-UP, STOP-WORD, FIX-INTRO, FIX-END

Abbreviations:

- SATZ is SATZ, SATZ+, SATZ_VERB-FIRST
- IMP is IMP-S-I, IMP-S-II
- DES is DES-S-I, DES-S-II
- FRAGE is WH-Q, YN-Q, FRAGE_VERB-FIRST
- MAINCLAUSE is SATZ, IMP, DES, FRAGE
- SUBORDCLAUSE is REL-S, KAU-S, KONS-S, KONZ-S, TEM-S, KOND-S, FIN-S, MOD-S, IND-R, INH-S-xxx

4.2.3 Text Input

From the lexicon we have information for each word in the following format:

```
("blinkte" (((b l I N k)(1))((t @)(0))) ((CAT V)))
orthography   phones stress syl.boundary  part-of-speech
```

Following rules are applied:

1. put a sentence boundary whenever you encounter one of the following four: . ? ! ;
2. put a clause boundary whenever you encounter: : ,
3. put a phrase boundary after a CONTENTWORD, but only if it is followed by a FUNCTIONWORD (see below for a list of abbreviations)
4. delete the last phrase boundary, if there is no CONTENTWORD in the following phrase.

Having piled up a series of phrases, we have to decide which intones to take...

1. last phrase before a sentence-boundary:
   - sentence finishes with a . ! or ; $\rightarrow$ H+L* as pitch accent and L-L% on last syllable
   - sentence finishes with a ? $\rightarrow$ L*+H as pitch accent and H-H% on last syllable
2. last phrase before a clause-boundary: L+H* as pitch accent and H-L% on last syllable
3. all other phrases: H* as pitch accent and L- on last syllable

...and where to put the pitch accent:

1. go through the phrase from right to left
2. check the POS of every word
3. if it is N or A put the pitch accent on that word and stop searching
4. if you cannot find any N or A in the whole phrase, look for an ADV, NUM, PRON, PREP (in that order! always going from right to left through the whole phrase), put the pitch accent on the respective word and stop searching
5. if 3) and 4) fail, put the pitch accent on the last word in the phrase.

Either before or after building up phrases and deciding on intones, we have to put a stress mark on the syllable that carries lexical stress. But only for words with certain part-of-speech: N, A, NUM, V, ADV, I.

Abbreviations:

- CONTENTWORD is N, A
- FUNCTIONWORD is NUM, V, ART, PRON, ADV, PREP, C, I

4.3 Evaluation

Unfortunately, the time was too short, to evaluate the "new" German voices of CHATR experimentally. Therefore, evaluation has to remain on a somewhat impressionistic level.

Compared to the previous speaker, German CHATR has been improved substantially. Even with an input in the Utterance Phoneme format (with no syntactic information), impressively intelligible and natural utterances can be produced. Please have a look at

http://www.itl.atr.co.jp/chatr/german.html

and judge yourself.

For examples that were produced with Text Input, go to

http://www.itl.atr.co.jp/ xcaren/examples.html

Still, the prosody prediction rules have been put up in a heuristic manner, so they sometimes make CHATR chop its words quite unnaturally. And even if the predicted prosody suits the utterance, you can never be sure that the right units will be selected from the database.

In Figure 4.1 you can see that the falling contour at the end of the synthesized utterance is exactly the opposite of the rising F0 contour that was predicted for that question[1].

So, as always, there is more than one possible reason for a bad CHATR output...

[1] the example is taken from the TDMT output, so it is not totally correct German

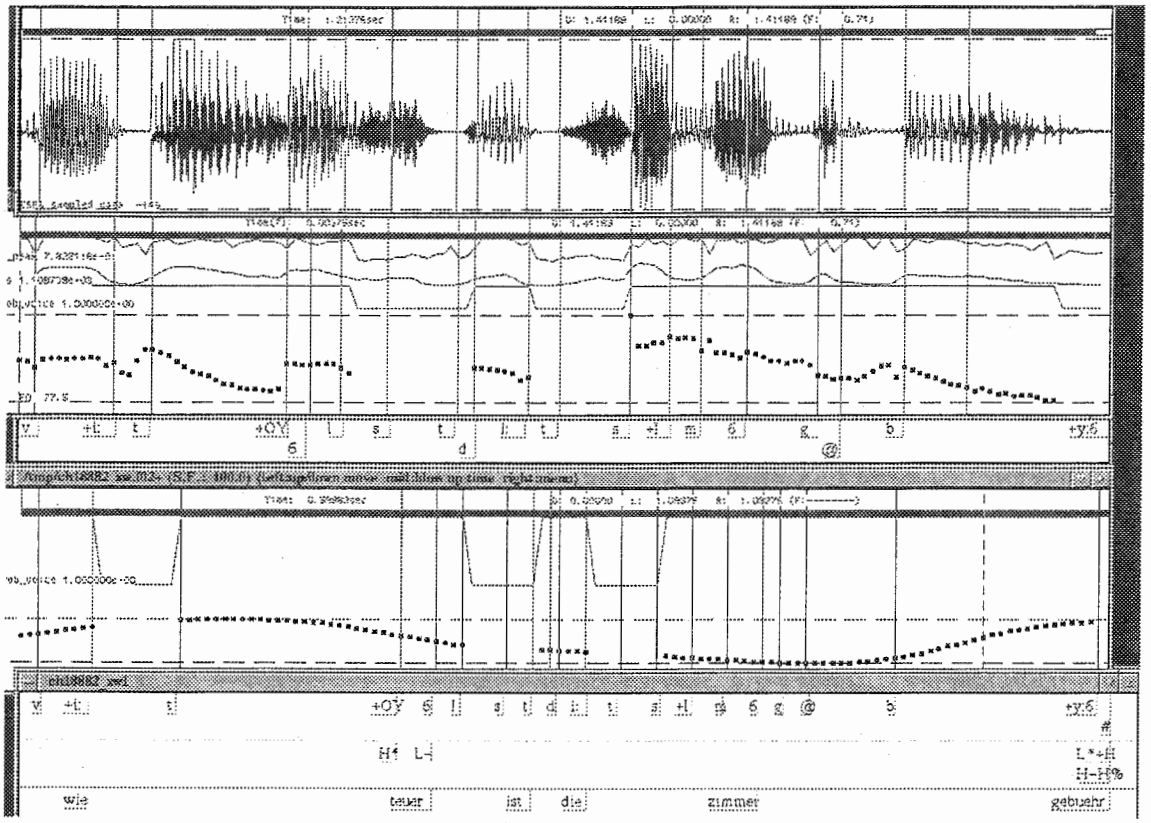

Figure 4.1: second row: synthesized F0 contour, forth row: predicted F0 contour

Chapter 5

Future Work

The following points should be regarded as suggestions to those who want to further improve the German voice of CHATR. This collection is by no means complete but might be somewhat helpful.

5.1 Database

- The sampaG phonemeset was set up because of practical reasons, as explained in section 2.1.2. In theory, there should be no difference between the phonemic transcription of stressed and unstressed variants of the same vowel. They should be predictable from different lexical stress and the durational difference that follows it.
- The same argument holds also for the "Q"-label which marks glottalization or glottal stops. Glottalization is predictable from the context, and therefore should not be labelled separately.

5.2 Lexicon

- As mentioned in section 3.2.4, the problem of double entries in the lexicon is not solved yet.
- In section 4.1.3 you can find some suggestions regarding pronunciation of unknown words (LTS rules) and the prediction of lexical stress and syllable boundaries.

5.3 Prosody Prediction

- Thorough (experimental) testing of the so far implemented prosody prediction rules is certainly necessary.

- As German ToBI-labeled corpora become available, a (linear regression) model should be trained on natural speech material, so that the German prosody of CHATR becomes more natural.
- Implement another theory of prosody prediction (see section 4.1 to get some ideas) and compare it to the one that has been implemented so far!

Chapter 6

Conclusion

At first, CHATR seemed to be an impenetrable jungle of modules, variables and functions. I doubted that anything substantial could be achieved during those eight weeks of my stay at ITL. But looking back, quite a bit has been improved regarding the German voice of CHATR. At least on the segmental level everything is stable by now.

What certainly became clear, is that without any knowledge about the semantic (and pragmatic) content of a sentence, prosody prediction cannot be perfect. The so far implemented syntax-based prediction gives acceptable results for simple "everyday" utterances, but it must fail inevitably when trying to synthesize literature. Fairy-tales, for example, were noticeably worse (regarding prosody) than hotel reservation dialogues. A reason for this might be, that our expectations are much higher when listening to a story that has been read to us many times.

We could show, that by simply expanding the database the output becomes much more intelligible and natural. But in order to avoid the problem, that a unit with the wrong F0 gets selected, we need a database that is

- even bigger, and
- contains a wider variety of prosodic patterns.

Bibliography

[Anderson 84] M.D. Anderson, J.B. Pierrehumbert, and M.Y. Liberman. Synthesis by rule of english intonation patterns. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, IEEE, New York, 1984.

[Bierwisch 66] M. Bierwisch. Regeln für die Intonation deutscher Sätze. In *Untersuchungen über Akzent und Intonation im Deutschen*, pp. 99–201. Akademie-Verlag, Berlin, 1966.

[CELEX 95] R. H. Baayen, R. Piepenbrock & L. Gulikers. *The CELEX Lexical Database (CD-ROM).* Linguistic Data Consortium, University of Pennsylvania, Philadelphia, PA, 1995.

[Féry 92] Caroline Féry. *Focus, Topic and Intonation in German.* SFB 340: Sprachtheoretische Grundlagen für die CL / Bericht Nr. 20. IBM Deutschland, Heidelberg, 1992.

[Féry 95] Caroline Féry. *Alignment, syllable and metrical Structure in German.* SfS-Report-02-95. Eberhard Karls Universität, Tübingen, 1995.

[Isačenko 64] A.V. Isačenko and H.J. Schädlich. *Untersuchungen über die deutsche Satzintonation.* Akademie-Verlag, Berlin, 1964.

[Reyelt 96] Matthias Reyelt et al. *Prosodische Etikettierung des Deutschen mit ToBI.* Verbmobil-Report 154, DfKI, Saarbrücken, 1996.

[Stock 82] E. Stock and C. Zacharias. *Deutsche Satzintonation.* VEB Verlag Enzyklopädie, Leipzig, 1982.

[Striegnitz 97] Kristina Striegnitz. *Teaching CHATR German Intonation - Lesson One.* ATR Technical Report, TR-IT-0237 (1997-09).

[Uhmann 91] S. Uhmann. *Fokusphonologie. Eine Analyse deutscher Intonationskonturen im Rahmen der nicht-linearen Phonologie.* Niemeyer, Tübingen, 1991.

[Weeks 97] Martyn Weeks. *The CHATR User Guide*. http://www.itl.atr.co.jp/~mweeks/. ATR Interpreting Telecommunications Research Laboratories, 1997.

Appendix A

Building a German Database

A.1 Speech Files

List of files read by Klaus Kohler (male speaker): (.16 for speech wave, .s1h for label file)

```
~nick/german/kiel/kiel_cors/ph90/berlin/k61/*.16 or *.s1h
~nick/german/kiel/kiel_cors/ph90/marburg/k61/*.16 or *.s1h
~nick/german/kiel/kiel_cors/ph90/restkorp/k61/*.16 or *.s1h
~nick/german/kiel/kiel_cors/ph90/butter/k61*.16 or k61*.s1h
~nick/german/kiel/kiel_cors/ph90/nordwind/k61*.16 or k61*.s1h
~nick/german/kiel/kiel_cors/ph92/erlangen/kko/*.16 or *.s1h
~nick/german/kiel/kiel_cors/ph92/siemens/kko/*.16 or *.s1h
```

List of files read by "rtd" (female speaker):

```
~nick/german/kiel/kiel_cors/ph90/berlin/k62/*.16 or *.s1h
~nick/german/kiel/kiel_cors/ph90/marburg/k62/*.16 or *.s1h
~nick/german/kiel/kiel_cors/ph90/restkorp/k62/*.16 or *.s1h
~nick/german/kiel/kiel_cors/ph90/butter/k62*.16 or k62*.s1h
~nick/german/kiel/kiel_cors/ph90/nordwind/k62*.16 or k62*.s1h
~nick/german/kiel/kiel_cors/ph92/erlangen/rtd/*.16 or *.s1h
~nick/german/kiel/kiel_cors/ph92/siemens/rtd/*.16 or *.s1h
```

In the directory where you want to build the database, type:

```
mkdir wav
```

Using the Kiel Corpus of Read Speech, you have to change the byte order of the .16 files. While in the directory where you want to build the database, type (you have to repeat this for every pathname in the list above):

```
for i in ~nick/german/kiel/kiel_cors/ph90/berlin/k61/*.16
> do n=`basename $i .16`
> echo $n
> dd conv=swab if=$i of=wav/$n.wav
> done
```

A.2 Label Files

In the directory where you want to build the database, type:

```
mkdir lab
```

The label files have to be changed to the XWAVES format. While doing that, we want to get rid of some labels and change some of the characters used in the original transcription:[1]

```
for i in ~nick/german/kiel/kiel_cors/ph90/berlin/k61/*.s1h
> do n=`basename $i .s1h`
> echo $n
> gawk -f ~xcaren/Phondat/kiel2xlb_new.awk $i | gawk -f
~xcaren/Phondat/postkiel.awk > lab/$n.lab
> done
```

For a detailed documentation of the awk-scripts used in the loop above, see Appendix C.1 below.

A.3 Training

In the directory where you want to build the database, type:

```
mkdir index
ln -s /dept2/work11/src/chatr-0.92/SunOS/chatr-0.92/db_utils
```

Make a list of all speech files:

```
ls wav/*.wav > files
```

Copy the following files and change them accordingly[2]:

```
cp ~xcaren/kkx/db_description db_description
cp ~xcaren/kkx/remake_db remake_db
cp ~xcaren/kkx/index/kko_synth.ch index/<your_database_name>_synth.ch
cp ~xcaren/kkx/index/kko_train.ch index/<your_database_name>_train.ch
```

Checking the database:

```
db_utils/check_labs
db_utils/check_phoneset
db_utils/check_align
```

[1] Attention: After having left ATR, the contents of my home directory will probably be stored somewhere else, and therefore they won't be accessable via xcaren anymore. In that case, you will have to change the paths for the awk-scripts.

[2] Check [Weeks 97] for further details.

If the checks find any segments that are too short (or whatever) change the label files by hand (using XWAVES) or take the file out of the "files" list.

When everything is ready:

```
remake_db > make.log 2>&1
```

If a message like "Serious network problems" pops up, it might be because you are not one of the VIPs who can check out HTK. Let the system manager fix this for you, so that you can use HTK to build the MFCC etc.

Appendix B

Building a German Lexicon

B.1 Available Awk Scripts

In order to create a CHATR lexicon from the data available on the CELEX CD-ROM, you can use the following awk-scripts (see Appendix C.2 for a detailed documentation of the more complex ones), which are currently in ~xcaren/CHATR/AWK/Lexicon/[1]:

1. celex2g_dic_new.awk gives you an uncompiled CHATR lexicon with orthographic and phonetic information (SAMPA transcription) only

2. celex2g_dic_+pos.awk gives you an uncompiled CHATR lexicon with orthographic, phonetic (SAMPA transcription) and part-of-speech information

3. nospace.awk takes a CHATR lexicon (compiled or not) and gets rid of the entries with a space in the orthographical string (detached prefixes)

4. onlyspace.awk keeps only those entries with a space in the orthographical string, in case you want to use them later on

5. delete_doubleentries.awk deletes those entries that are *exactly* the same as the preceeding entry

6. double_entry.awk gives you a list of all entries in the lexicon with the same orthographic representation

7. sampa2sampaG.awk converts the SAMPA transcription into sampaG, as defined in sampaG_def.ch

[1] Attention: After having left ATR, the contents of my home directory will probably be stored somewhere else, and therefore they won't be accessable via xcaren anymore. In that case, you will have to change the paths for the awk-scripts.

B.2 Lexicon Recipe

Suppose you would like to create a compiled CHATR lexicon for German, using the sampaG transcription, with information about part-of-speech, without "spacy" entries, and without entries that are exactly the same. Then you would have to type the following commands (or something similar, if you don't like the filenames or use a different directory):

```
gawk -f /DB/LTDB4/celex_rel_2/german/gsl/awk/synlabel.awk
/DB/LTDB4/celex_rel_2/german/gsl/gsl.cd 4 CL > ~xcaren/CHATR/lemma_pos.cd

cp /DB/LTDB4/celex_rel_2/german/gpw/gpw.cd ~xcaren/CELEX/gpw.cd

gawk -f ~xcaren/CHATR/AWK/Lexicon/celex2g_dic_+pos.awk
~xcaren/CELEX/gpw.cd > ~xcaren/CHATR/celex+pos.ch

gawk -f ~xcaren/CHATR/AWK/Lexicon/nospace.awk
~xcaren/CHATR/celex+pos.ch > ~xcaren/CHATR/celex+pos_nosp.ch
```

Go to your CHATR window, and type:

```
(Lexicon Compile /home/as70/xcaren/CHATR/celex+pos_nosp.ch
/home/as70/xcaren/CHATR/celex+pos_nosp_comp.ch)
```

If this doesn't work because of memory problems, you should delete the first two lines and the last line of celex+pos_nosp.ch by hand, then sort the file with the UNIX sort command, and insert

```
CompLexicon
sampaG
```

as the first two lines into the sorted file, which you should call celex+pos_nosp_comp.ch.

Then continue with:

```
gawk -f ~xcaren/CHATR/AWK/Lexicon/delete_doubleentries.awk

gawk -f ~xcaren/CHATR/AWK/Lexicon/sampa2sampaG.awk
~xcaren/CHATR/celex+pos_nosp_comp.ch >
~xcaren/CHATR/celex+pos_nosp_G_comp.ch
```

B.3 Tell CHATR about it

In order to make CHATR use the newly created lexicon, you have to change the `lib/data/lexicons.ch` file within CHATR, or (which is recommended while testing the new lexicon) load a file into CHATR, which is similar to the following ~xcaren/CHATR/sampa_lexicons.ch:

```
;;;  Functions to specify the different lexicons
;;;
;;; temporary locations while testing !!! :
(defvar sampaG_compiled_lex
  "/data/as70/xcaren/celex+pos_nosp_G_comp.ch"
  "Celex based lexicon for German in sampaG")
(defvar sampa_compiled_lex
  "/data/as70/xcaren/celex+pos_nosp_comp.ch"
  "Celex based lexicon for German in SAMPA")
;;
;;; Lexicon for German in sampaG:
;;
(define setup_sampaG_lex ()
(require 'sampaG_def)
; (Lexicon Select sampaG_compiled_lex)
(Lexicon Phone_Set sampaG)
(Lexicon Use sampaG_compiled_lex)
(Lexicon Fail Error)
(Lexicon Add sampaG
 ("Caren" (((k +a:)(1))((r @ n)(0))) ((CAT N)))
 ("Kristina" (((k r I s)(0))((t +i:)(1))((n a:)(0))) ((CAT N)))
 )
)
;;
;;; Lexicon for German in sampa:
;;
(define setup_sampa_lex ()
(require 'sampaG_def)
; (Lexicon Select sampaG_compiled_lex)
(Lexicon Phone_Set sampaG)
(Lexicon Use sampa_compiled_lex)
(Lexicon Fail GLTS)
)
;;
(provide 'sampa_lexicons)
'ok
```

To load this file into CHATR, you simply type:

```
(load '~xcaren/CHATR/sampa_lexicons.ch)
```

Then, you have to tell CHATR to actually use this lexicon for the currently selected speaker by typing:

```
(setup_sampaG_lex)
```

(or whatever the name of the function is, that you defined in the file you just loaded).

B.4 Mapping between "beep" and "sampaG"

```
(Phoneme Map beep sampaG
(
(ax 6)
(axr 6)
(aa +a:)
(ao +o:6)
(ah +a)
(ay +aI)
(aw +aU)
(ae +E)
(ea +E:6)
(ia +i:6)
(ua +u:6)
(el l)
(en n)
(er +96)
(eh +E)
(ey +e:) ;; should put +EI here (two separate phonemes)
(iy +i:)
(ih +I)
(uh +U)
(uw +u:)
(em m)
(oh +O)
(ow +o:)
(oy +OY)
(y j)
(r r)
(l l)
(m m)
(n n)
(ng N)
(nx n)
(jh Z) ;; should put dZ here (two separate phonemes)
(ch S) ;; should put tS here (two separate phonemes)
(zh Z)
(sh S)
(th s)
(dh z)
(p p)
(b b)
(d d)
(dx d)
```

```
(t t)
(k k)
(g g)
(f f)
(v v)
(z z)
(s s)
(hh h)
(hv v)
(w v)
(sil #)
(brth #)
))
```

Appendix C

AWK Scripts

C.1 Changing Labels

C.1.1 kiel2xlb_new.awk

```
## name: kiel2xlb_new.awk
## written by: caren & nick
## last change: 1997/8/11 by caren and nick
##
## input: *.s1h phoneme label files from "The Kiel Corpus of Read Speech"
##        (currently in ~nick/german/kiel/kiel_cors/)
##        [for a detailed description of the label format see
##         http://www.phonetik.uni-muenchen.de/Bas/BasFormatsdeu.html#S1]
##
## output: label files in the XWAVES format
##
## attention: use the output of kiel2xlb_new as
##            input of ~xcaren/Phondat/postkiel.awk
##       to get rid of unwanted labels (which cannot be removed at once)!
##
## explanatory remarks:
##   The *.s1h-label files have the label at the beginning of the segment,
## but we want it at the end of the segment.
##   So, at the current time we want to stick in the previous label.
##   To make it even more complicated, we want to get rid of the '-h' label,
## which marks aspiration after plosives before vowels. We don't want
## to throw it away completely, but shift the label of the plosive to the
## time of the '-h' label (because the aspiration really belongs to the
## plosive).
##   That's why we have to store not only the previous label, but also the
## previous previous label (in case a '-h' follows).
##
```

```
BEGIN{print "#";clr=123;lbl="#";while ($1!~/hend/) getline;}
{
  if (lbl2 ~/\-h/) {
    time=$1/16000; # previous label, current time
    lbl2=$2;
    getline;
  }
  if (lbl!="" && time!=0) {
      printf("%9.4f%4s%7s\n", time,clr,lbl);
  }
  gsub(/\r/,"",lbl2);     # kill ^M

  gsub(/\#\#/,"\$",lbl2);# transform special boundaries to normal ones ($ marks
  gsub(/\#/,"\$",lbl2);  # the beginning of the phone-string)

  gsub(/\$.*-/,"",lbl2); # get rid of transcribed elisions, replacements or
                         # insertions -> just use the transcription of the
                         # phone that is actually produced

  gsub(/\./,"",lbl2);     # information about punctuation not needed
  gsub(/\?/,"",lbl2);     # (only the comma will be transformed into a
  gsub(/\!/,"",lbl2);     # pause-character, which might get deleted in the
  gsub(/\,/,"#",lbl2);    # post-processing "postkiel.awk")

  gsub(/p:/,"#",lbl2);  # p: = pause
  gsub(/c:/,"#",lbl2);  # c: = beginning of first word = pause
  gsub(/v:/,"",lbl2);   # information about mispronunciation not needed
                        # (as long as transcribed correctly)

  if (lbl2 ~/\~./)      # get rid of fine transcription of nasalization
    gsub(/\~/,"",lbl2); # (e.g. ~E), but leave in the (originally French)
                        # nasal vowels (e.g. E~)

  gsub(/\%/,"",lbl2);     # kill information about uncertain boundary
  gsub(/\=/,"",lbl2);     # no fine transcription of diphthongs

  gsub(/\+/,"",lbl2);     # kill information about function words

  gsub(/\'/,"+",lbl2);    # new phonemeset sampaG:
  gsub(/\"/,"+",lbl2);    # mark stressed vowels with a + in front

  gsub(/q/,"Q",lbl2);     # no difference between q and Q for post-processing

  gsub(/\$/,"",lbl2);     # finally kill $ (= marked beginning of phone-string)
```

```
  time=$1/16000; # previous time
  lbl=lbl2; # previous previous label
  lbl2=$2;
}
END{if (lbl!="")          # print the last label
      {printf("%9.4f%4s%7s\n",$1/16000,clr,lbl)}}
```

C.1.2 postkiel.awk

```
## name: postkiel.awk
## written by: kristina & caren
## last change: 1997/8/19 by caren
##
## input: label files in XWAVES format produced by kiel2xlb_new.awk
##
## output: cleaned label files in XWAVES format (cleaned = no double labels
##         at one time, and without unnecessary information about glottal
##         stops or glottalization)
##
## explanatory remarks:
##   The 'Q' labels (glottal stop or glottalization) will be deleted or
## not, depending on their neighbouring labels: between two consonants or
## two vowels the 'Q' is redundant (predictable), but between a consonant
## and a vowel (or the other way round, if that ever happens) we have to
## mark it somehow. That's why we insert the 'Q' label just 10 ms after
## the first of the two sounds.
##   Also, we remove (or shift the time of) any label, that has the same time
## as the preceeding label. So we have to look ahead, in case there are more
## than two labels with the same time.
##

function rules(tpprev,lblpprev,tprev,lblprev,lblcurr) {

   v["a"]; v["a:"]; v["E"]; v["E:"]; v["u:"]; v["U"]; v["o:"]; v["O"]; v["i:"];
v["I"]; v["e:"]; v["y:"]; v["Y"]; v["2:"]; v["9"]; v["@"]; v["6"]; v["u"];
v["o"]; v["i"]; v["e"]; v["y"]; v["2"]; v["aI"]; v["aU"]; v["OY"]; v["a~"];
v["E~"]; v["9~"]; v["O~"]; v["a6"]; v["a:6"]; v["E6"]; v["E:6"]; v["u:6"];
v["U6"]; v["o:6"]; v["O6"]; v["i:6"]; v["I6"]; v["e:6"]; v["y:6"]; v["Y6"];
v["2:6"]; v["96"]; v["u6"]; v["o6"]; v["i6"]; v["e6"]; v["y6"]; v["26"];
v["aI6"]; v["aU6"]; v["OY6"]; v["+a"]; v["+a:"]; v["+E"]; v["+E:"]; v["+u:"];
v["+U"]; v["+o:"]; v["+O"]; v["+i:"]; v["+I"]; v["+e:"]; v["+y:"]; v["+Y"];
v["+2:"]; v["+9"]; v["+u"]; v["+o"]; v["+i"]; v["+e"]; v["+y"]; v["+2"];
v["+aI"]; v["+aU"]; v["+OY"]; v["+a~"]; v["+E~"]; v["+9~"]; v["+O~"]; v["+a6"];
v["+a:6"]; v["+E6"]; v["+E:6"]; v["+u:6"]; v["+U6"]; v["+o:6"]; v["+O6"];
```

```
v["+i:6"]; v["+I6"]; v["+e:6"]; v["+y:6"]; v["+Y6"]; v["+2:6"]; v["+96"];
v["+u6"]; v["+o6"]; v["+i6"]; v["+e6"]; v["+y6"]; v["+26"]; v["+aI6"];
v["+aU6"]; v["+0Y6"];
   c["b"]; c["d"]; c["g"]; c["p"]; c["t"]; c["k"]; c["f"]; c["v"]; c["s"];
c["z"]; c["S"]; c["Z"]; c["C"]; c["x"]; c["h"]; c["l"]; c["r"]; c["m"]; c["n"];
c["N"]; c["j"];

   clr = 123;

   if (lblprev == "Q")
     {
      if (lblpprev in v && lblcurr in v)
        {
         printf("%9.4f%4s%7s\n",tpprev,clr,lblpprev);
        }
      if (lblpprev in c && lblcurr in c)
        {
         printf("%9.4f%4s%7s\n",tpprev,clr,lblpprev);
        }
      if (lblpprev in v && lblcurr in c)
        {
         tprev += 0.01;
         printf("%9.4f%4s%7s\n",tpprev,clr,lblpprev);
         printf("%9.4f%4s%7s\n",tprev,clr,lblprev);
        }
      if (lblpprev in c && lblcurr in v)
        {
         tprev += 0.01;
         printf("%9.4f%4s%7s\n",tpprev,clr,lblpprev);
         printf("%9.4f%4s%7s\n",tprev,clr,lblprev);
        }
      if (lblpprev == "#" || lblpprev == "Q")
        {
         printf("%9.4f%4s%7s\n",tpprev,clr,lblpprev);
        }
     }
   else
     {
      printf("%9.4f%4s%7s\n",tpprev,clr,lblpprev);
     }
}
BEGIN{tprev = -1; clr = 123; tpprev = -1;
      print "#";}
$1 != "#" && $1 != 0 {
    if (tprev != -1)   # if a label has been read already
```

```
      {
       if (tprev == $1)  # if current time equals previous time,
         {                 # and if current time also equals
          if (tpprev == $1) # preprevious time,
            {                  # save only the current label,
             lblprev = $3;     # but throw away the previous label.
            }
          else              # and if current time does not equal
            {               # preprevious time,
             tpprev = tprev;    # keep on reading (don't throw away
             lblpprev = lblprev;# any labels)
             lblprev = $3;
            }
         }
       else            # if current time does not equal previous time,
         {                 # and if preprevious time does not exist or
          if (tpprev == -1) # has been dealt with already,
            {               # then print previous time with its label
             printf("%9.4f%4s%7s\n",tprev,clr,lblprev);
             tprev = $1; lblprev = $3;
            }
          else              # and if preprevious time equals previous time
            {                    # then apply the rules (which also print)
             rules(tpprev,lblpprev,tprev,lblprev,$3);
             tpprev = -1;        # and reset the times and labels
             tprev = $1; lblprev = $3;
            }
         }
      }
    else  # if no label has been read yet
      {
       tprev = $1; lblprev = $3; # read it!
      }
}
END {
     printf("%9.4f%4s%7s\n",$1,clr,$3);
     printf("%9.4f%4s%7s\n",$1+=0.02,clr,"#"); # add a pause label at the end
}                                              # of the label file
```

C.2 Lexicon

C.2.1 celex2g_dic_+pos.awk

```
## name: celex2g_dic_+pos.awk
## written by: caren
```

```
## last change: 1997/9/8 by caren
##
## input: CELEX's gpw.cd (German Phonology Wordforms)
##        [currently in /DB/LTDB4/celex_rel_2/german/gpw/]
##
## output: German dictionary in the uncompiled CHATR format
##         (orthography, phonemic transcription _and_ part-of-speech)
##
## attention: if there isn't one yet, please create the file "lemma_pos.cd"
##            with the following command:
##    gawk -f /DB/LTDB4/celex_rel_2/german/gsl/awk/synlabel.awk
/DB/LTDB4/celex_rel_2/german/gsl/gsl.cd 4 CL > ~xcaren/CHATR/lemma_pos.cd
##
## advice: This script takes a looong time to run, and over the network
##         it is even much slooower. So, you'd better copy gpw.cd onto
##         your machine and possibly split it into smaller pieces (I don't
##         know, if the splitting contributes to the speed, but try it
##         anyway).
##
## explanatory remarks:
##   Since there is no information about the part-of-speech of each word
## in gpw.cd, we have to look it up in gsl.cd (German Syntax Lemmas). To make
## processing easier, we first convert gsl.cd into a file of three fields:
## LemmaNumber\Lemma\POS.
##   The POS information of every lemma is then read into an array at the
## beginning of the script.

BEGIN{

   FS="\\"   # backslash as field separator

   while (getline <"/home/as70/xcaren/CHATR/lemma_pos.cd" > 0) {
       pos[$1]=$3   # create array with POS information
   }

   cons["C"]; cons["N"]; cons["S"]; cons["Z"]; cons["b"]; cons["d"];
   cons["f"]; cons["g"]; cons["h"]; cons["j"]; cons["k"]; cons["l"];
   cons["m"]; cons["n"]; cons["p"]; cons["r"]; cons["s"]; cons["t"];
   cons["v"]; cons["x"]; cons["z"];
                  # all consonants in SAMPA for German

   bvow["u:"]; bvow["U"]; bvow["o:"]; bvow["O"]; bvow["a:"]; bvow["a"];
   bvow["aU"];    # field of backvowels in SAMPA for German
   bvow["u"]; bvow["o"]; bvow["a"]; bvow["&"]; bvow["B"];
                  # additional backvowels in DISC for German
```

```
   print "(Lexicon sampaG";
   print "";
}{
   printf("(\"%s\" (", $2); # lexical word (orthography)
   str=0;
   lastphon=0;
   txt="("; # beginning of first syllable

   for(i=1;i<=length($5);i++)
   {
      if (substr($5,i,1)=="-" || substr($5,i,1)==" "){
         txt=phfix(txt"-", lastphon, cons, bvow); # clean up syll
 printf("(%s)(%d))", txt, str); # print syll
         lastphon=substr($5,i-1,1); # store last phone of the syll
 str=0;
 txt="("; # beginning of next syllable
      }
      else # check for stress
 if (substr($5,i,1)=="\'"){
    str=1;
 }
 else # get next phone
    txt = txt " " substr($5,i,1);
  }

   txt=phfix(txt"-", lastphon, cons, bvow); # last syll
   printf("(%s)(%d))) ((CAT %s)))\n", txt, str, pos[$4]);

}


function phfix(syl, Lastphon, Cons, Backvowels){

# fix the phones (DISC transcription to SAMPA for German)
    gsub(/\( /,"\(",syl);
    gsub(/a/,"a:",syl);
    gsub(/e/,"e:",syl);
    gsub(/1/,"e:",syl);
    gsub(/2/,"aI",syl);
    gsub(/4/,"OY",syl);
    gsub(/6/,"aU",syl);
    gsub(/w/,"v",syl);
    gsub(/\+/,"p f",syl);
    gsub(/\=/,"t s",syl);
```

```
    gsub(/J/,"t S",syl);
    gsub(/\_/,"d Z",syl);
    gsub(/i/,"i:",syl);
    gsub(/\#/,"a:",syl);
    gsub(/\$/,"06",syl);
    gsub(/u/,"u:",syl);
    gsub(/\3/,"96",syl);
    gsub(/y/,"y:",syl);
    gsub(/\)/,"E:",syl);
    gsub(/\|/,"2:",syl);
    gsub(/o/,"o:",syl);
    gsub(/W/,"aI",syl);
    gsub(/B/,"aU",syl);
    gsub(/X/,"OY",syl);
    gsub(/\//,"9",syl);
    gsub(/\{/,"E",syl);
    gsub(/&/,"a",syl);
    gsub(/A/,"a",syl);
    gsub(/V/,"a",syl);
    gsub(/\^/,"9 N",syl); # simulation of French nasals
    gsub(/c/,"E m",syl);
    gsub(/q/,"a N",syl);
    gsub(/0/,"E N",syl);
    gsub(/\~/,"O N",syl);

## rules for distinguishing between x and C
    gsub(/x/,"C",syl);
    for (q in Backvowels) {
      gsub(q" C",q" x",syl);
    }
    if (Lastphon in Backvowels) {   # x is used after backvowels
      gsub(/\(C/,"\(x",syl);        # also after syllable boundaries
    }

## replace '<vowel> r' in front of a consonant, syllable boundary, or word
## boundary with '<vowel>6'
    for (x in Cons) {
      gsub("E: r "x,"E:6 "x,syl);
      gsub("E r "x,"E6 "x,syl);
      gsub("9 r "x,"96 "x,syl);
      gsub("I r "x,"I6 "x,syl);
      gsub("O r "x,"O6 "x,syl);
      gsub("U r "x,"U6 "x,syl);
      gsub("Y r "x,"Y6 "x,syl);
      gsub("a: r "x,"a:6 "x,syl);
```

```
      gsub("a r "x,"a6 "x,syl);
      gsub("e: r "x,"e:6 "x,syl);
      gsub("2: r "x,"2:6 "x,syl);
      gsub("i: r "x,"i:6 "x,syl);
      gsub("o: r "x,"o:6 "x,syl);
      gsub("u: r "x,"u:6 "x,syl);
      gsub("y: r "x,"y:6 "x,syl);
      gsub("@ r "x,"6 "x,syl);
    }
    gsub("E: r-","E:6",syl);
    gsub("E r-","E6",syl);
    gsub("9 r-","96",syl);
    gsub("I r-","I6",syl);
    gsub("O r-","O6",syl);
    gsub("U r-","U6",syl);
    gsub("Y r-","Y6",syl);
    gsub("a: r-","a:6",syl);
    gsub("a r-","a6",syl);
    gsub("e: r-","e:6",syl);
    gsub("2: r-","2:6",syl);
    gsub("i: r-","i:6",syl);
    gsub("o: r-","o:6",syl);
    gsub("u: r-","u:6",syl);
    gsub("y: r-","y:6",syl);
    gsub("@ r-","6",syl);


   gsub(/\-/,"",syl);  # remove remaining syllable markers

   return syl;
}

END { print ")" }


##  FYI: DISC phones
##
##   "p",  "b",  "t",  "d",   "k",  "g",   "N",   "m",  "n",  "l",
##   "r",  "f",  "v",  "s",   "z",  "S",   "Z",   "j",  "x",  "h",
##   "+",  "=",  "J",  "_",   "i",  "#",   "a",   "$",  "u",  "3",
##   "y",  ")",  "e",  "|",   "o",  "1",   "2",   "4",  "6",  "W",
##   "B",  "X",  "I",  "Y",   "E",  "/",   "{",   "&",  "A",  "V",
##   "0",  "U",  "@",  "^",   "c",  "q",   "O",   "~",  "w"
```

C.2.2 sampa2sampaG.awk

```
## name: sampa2sampaG.awk
## written by: caren
## last change: 1997/9/8 by caren
##
## input: CHATR dictionary for German with SAMPA transcriptions and
##        part-of-speech information,
##        e.g. ~xcaren/CHATR/celex+pos_comp.ch
##
## output: CHATR dictionary with sampaG transcriptions, where stressed
##         vowels are marked with '+'
##
## explanatory remarks:
##    The basic CHATR dictionary for German should use the standard SAMPA
## transcription, so that it can be used in various situations. But for
## the current German speakers kko and rtd, we use an extended SAMPA,
## defined in sampaG_def.ch.
##    sampaG came to life when the transcription on the PhonDat CD-ROM
## proved to be too phonemic, so that CHATR would sometimes choose very
## centralized and short realizations of phonemes, that should have been
## stressed (and therefore not centralized). Therefore, the stressed variant
## of a phoneme is now marked by a '+'.
##    So, for kko and rtd we need a different lexicon, which can be derived
## from the basic one using this script. For the time being, this is the
## easiest way to do it...
##


BEGIN{

FS="[(][(]";
OFS="((";

v["a"]; v["E"]; v["u"]; v["U"]; v["o"]; v["O"]; v["i"]; v["I"];
v["e"]; v["y"]; v["Y"]; v["2"]; v["9"];

}

$0 !~ /ART|PRON/ {
  for (i=1; i<=NF; i++) {
    if ($i ~ /1/) {
      for (j=1; j<=length($i); j++){
k=substr($i,j,1);
if(k in v){
```

```
  gsub(k,"+"k,$i);
  j=length($i)+1;
}
      }
    }
  }
  gsub(/\(\+/, "\(Q \+"); # this adds a glottal stop before every
                          # vowel at the beginning of a stressed
                          # syllable
  print;
}
```