TR-IT-0237

Teaching CHATR German Intonation

Lesson One

Kristina I. Striegnitz

Sep. 1997

ABSTRACT

I will describe the work on prosody prediction for German, which I did during my two month stay at ITL. I will briefly discuss the heuristics used to predict phrasing, placement of pitch accent and intones and will then explain more detailed how I implemented these things to be used by CHATR.

Contents

Chapter 1

Introduction

I spent eight weeks at ITL as a participant in the REES program (research experience for european students) of the Japanese science and technology exchange center (JISTEC). Another German participant of the same program was at ITL as well and we were working together very closely.

During this time I worked on the German synthesis within CHATR. I was mainly concerned with predicting prosody (in the form on ToBI labels) taking simply text as well as the output of the TDMT system as input for CHATR. TDMT is an example based machine translation system, which is being developed at ITL. Since during the translation process some syntactic analysis is done, we could base our prosody prediction rules on a little bit more than just the words.

I will describe the algorithms we used to predict phrasing and intonation from text input and from the input we received from the TDMT system. I will also explain the changes which had to be made to the CHATR source code in order to achieve a better result in the final F0 prediction.

In chapter 2, I will describe the heuristics we used for prosody prediction. Chapter 3 briefly introduces the relevant modules of CHATR, showing which parts of the information necessary to synthesise a wave each of them adds to a given utterance. Chapter 4 describes the prosody prediction for TDMT input in more detail and chapter 5 does the same for text input. Finally, in chapter 6, I will explain, how the actual F0 values are predicted from the specified intones.

Chapter 2

A Touch of Theory

I will describe the rules we used for prosody prediction from a very technical point of view. Anybody interested in more theoretical details should refer to Caren Brinckmann's report ([2]).

2.1 From Part-of-Speech and Topological Fields to Prosody

The TDMT system translates, among other language pairs, from Japanese to English. In the course of the translation process the utterance is syntactically analysed, which yields information very helpful to the correct prediction of phrasing. The structure which is passed on from the TDMT system to CHATR contains the following information:

- words
- phonemic transcription, syllable boundaries, lexical stress
- part-of-speech
- sentence type, punctuation
- topological fields

The information about sentence type in combination with punctuation enables us to distinguish between main clauses and subordinate clauses as well as between statements and questions.

For the generation of German TDMT uses a theory of topological fields. Therefore, we, unfortunately, can not get a phrase structure tree, but we do have some kind of information about grouping of words.

Stock and Zacharias (in [8]) use part-of-speech and sentence structure to predict phrasing and the placement of pitch accent within these phrases.

We adapted their approach to our environment. For predicting the actual intones, we oriented ourselves on the rules given by Isačenko and Schädlich in [6].

2.2 The Prediction Rules

Here comes an overview of the rules we used for prediction of phrasing, placement of pitch accent and type of intones for pitch accents and boundary tones.

We used ToBI (or rather GToBI, which is a language specific variant of ToBI [7]) to annotate prosody. ToBI devides sentences into intonation phrases, which again are devided into intermediate phrases.

2.2.1 Phrasing

- put a sentence boundary at every '.', '!', '?' and ':'
- put an intonation phrase boundary
  - between two main clauses
  - between a main clause and a subordinate clause
  - at every ':'
- put an intermediate phrase boundary
  - at every ','
  - at the end of every 'syntactic phrase', which contains an adjective or noun

'syntactic phrase' is here used in the sense of the TDMT algorithm. It just denotes a grouping of words, which usually is identical to a topological field.

2.2.2 Placement of Pitch Accent

Every intermediate phrase has to have one pitch accent specified. Usually the last noun or adjective respectively get accented, but there might be phrases without either one.

- on the last noun or adjective in every intermediate phrase
- if there are no nouns or adjectives, on the last accentable word with the highest priority; accentable words are (ordered by priority): adverbs, verb prefixes, idiomatic expressions and prepositions

- if there is no accentable word, on the last word in the phrase

2.2.3 Intones

Now, that the utterance is devided into phrases and every phrase contains one word marked as accented, we have to decide, what intones to put on the different phrase boundaries and pitch accents. As mentioned above we based this on a paper by Isačenko and Schädlich.

- last intermediate phrase in a sentence
  question: L*+H on pitch accent and H-H% on last syllable
  others: H+L* on pitch accent and L-L% on last syllable

- last intermediate phrase in an intonation phrase
  L+H* on pitch accent and H-L% on last syllable

- all other phrases
  H* on pitch accent and L- on last syllable

Chapter 3

Welcome to CHATR

CHATR ist the speech synthesis system developed by the ATR Interpreting Telecommunications Research Laboratories. Anybody not familiar with the basic idea behind CHATR, should refer to [3] or [4].

I will only describe those parts of CHATR relevant for our work. For a more thorough description of the system, please refer to Tony Hébert's technical report ([5]), which includes a very nice illustration of the changes an utterance will undergo from its input until its synthesis, or to the CHATR manual ([9]).

CHATR has various input formats, which differ in the amount of information they can hold. The format called phonoform is the most informative one. In cases, where all the slots are filled, it perfectly describes a wave, i.e. the utterance can be synthesised right away with no further processing needed. We used the phonoform input format for the TDMT interface. Text input is maybe the least informative of the input formats. We used a dictionary to obtain some additional information. I will now explain more in more detail how CHATR handles input in these two formats and how this input is processed to acquire the information necessary for synthesis.

3.1 Text and HLP Input

When CHATR receives the command

```
(SayText ''Als der Frosch ueber die Wiese sprang,
        wurde er vom Storch gefressen. Ist das nicht raurig?'')
```

it adds to the utterance a stream containig a tree structure. The stream is called Sphrase stream, where "S" is standing for "syntactic". But, since we are talking about text input and no syntactic analysis is done, CHATR does not have any information about the syntax of the utterance, so that the Sphrase tree will actually be quite a flat structure in this case.

```
(((CAT D))
  (((CAT S) (IFT Statement))
    (((LEX als)))
    (((LEX der)))
    (((LEX Frosch)))
    (((LEX ueber)))
    (((LEX die)))
    (((LEX Wiese)))
    (((PUNC ,) (LEX sprang)))
    (((LEX wurde)))
    (((LEX er)))
    (((LEX vom)))
    (((LEX Storch)))
    (((PUNC .) (LEX gefressen))))
  (((CAT S) (IFT Question))
    (((LEX ist)))
    (((LEX das)))
    (((LEX nicht)))
    (((PUNC ?) (LEX traurig)))))
```

Figure 3.1: The initial Sphrase tree built from text input

As can be seen in figure 3.1, the only structuring is done by the nodes labeled "(CAT D)" and "(CAT S)", which can be thought of as meaning discourse or sentence boundary respectively. Of course, the Sphrase tree structure is also capable of representing syntactic constituent trees for example. And, in fact, from this point on our input will take the exact same route through CHATR's modules as an Sphrase tree built from an input in the HLP format (a high level linguistic structure) would; i.e. it will pass through the following modules: hlp_module, word_module, phonology_module, intone_module, duration_module, int_target_module, power_module. I will now give a very rough description of what they do.

hlp_module the features which can be added to nodes in hlp input are used to predict phrasing and prosody (I will later come back to this module in greater detail)

word_module lexicon lookup of the words to get information about the phonemic transcription, syllable boundaries and lexical stress

phonology_module pause prediction

intone_module prosody prediction

duration_module prediction of durations for the segments

int_target_module prediction of F0 contour from the prosodic labels (intone_module)

power_module prediction of power

At this point then, the segments are specified in a way, that CHATR has enough information to chose appropriate units from the database.

3.2 Phonoform Input

The phonoform input is the most informative way of giving an input to CHATR. It was originally intended to serve as a way of describing speech waves in a database. These could then be removed from the database and be resynthesised for testing. The following things can be explicitly specified in phonoform input as can also be seen in figure 3.2.

- segments
- duration
- power
- pitch
- syllables
- lexical stress
- ToBI labels
- break indices
- words

Everything that makes up a speech wave is specified. CHATR thus only calls two functions upon phonoform input. The phonoform input function, which constructs a structure of type 'Utterance' from the given input and the synthesis function.

```
(Utterance
  PhonoForm
  (:D nil
   (:S ((PauseLength 65))
    (Word Attorney nil
     (Syl ax () (Phoneme ax 70 8.5100 ((187.0000 35))))
     (Syl t.er ((Stress 1) (Intones HiF0 H*))
      (Phoneme t 110 7.1200 ((242.0000 55)))
      (Phoneme er 80 8.7500 ((255.0000 40))))
     (Syl n.iy nil
      (Phoneme n 50 8.6700 ((233.0000 25)))
      (Phoneme iy 60 8.3400 ((193.0000 30)))))
    (Word General ((Break 1))
     (Syl d.jh.eh.n ((Stress 1) (Intones !H*))
      (Phoneme d 60 7.7300 ((173.0000 30)))
      (Phoneme jh 40 7.4100 ((226.0000 20)))
      (Phoneme eh 110 8.4300 ((205.0000 55)))
      (Phoneme n 30 8.2700 ((196.0000 15))))
     (Syl axr () (Phoneme axr 130 8.2600 ((158.0000 65))))
     (Syl el ((Intones L-H%))
      (Phoneme el 110 7.9000 ((180.0000 55))))))
   (:S nil
    (Word James nil
     (Syl d.jh.ey.m.z ((Stress 1) (Intones H*))
      (Phoneme d 70 7.0900 ((182.0000 35)))
      (Phoneme jh 50 7.0800 ((184.0000 25)))
      (Phoneme ey 150 8.2200 ((154.0000 75)))
      (Phoneme m 100 7.7600 ((143.0000 50)))
      (Phoneme z 30 6.7700 ((200.0000 15)))))
    (Word Shannon ((Break 1))
     (Syl sh.ae.n ((Stress 1) (Intones HiF0 H*))
      (Phoneme sh 90 6.9900 ((200.0000 45)))
      (Phoneme ae 150 8.3700 ((172.0000 75)))
      (Phoneme n 80 8.1000 ((144.0000 40))))
     (Syl ax.n ((Intones L-L%))
      (Phoneme ax 30 7.4400 ((104.0000 15)))
      (Phoneme n 50 7.1100 ((145.0000 25)))))))))
```

Figure 3.2: An example for the phonoform input format. The numbers specify duration, power and pitch.

Chapter 4

The TDMT Interface

4.1 From TDMT to Phonoform

The input we receive from the TDMT system looks like the structure in figure 4.1. The first field of each line contains the words. The second field contains syntactical information and is further devided into three subfields containing sentence type, information about the syntactic function of that line, and part-of-speech. The third field contains the phonemic transcription of the words including syllable boundaries. And the fourth field holds lexical stress information. Each line is thought to represent one topological field. That is not in all cases consistantly realised though. The topological fields might be split up further according to syntactic functionality (e.g subjects and objects may be in a line of their own).

The phonoform input format was chosen for the TDMT interface, because it allowed us to use all the information we received from TDMT. Words, segments, syllables and lexical stress can be directly taken over from one format into the other. Using the rules from chapter 2 we will predict phrasing, which gives us boundaries, such as 'S', 'C', 'P' and the associated 'PauseLength' features (cf. figure 3.2). We used the 'C' (clause) boundary as intonation phrase boundary and the 'P' (phrase) boundary as intermediate phrase boundary. We can then use the rules from chapter 2 to assign intones to words. Lexical stress then determines, to which syllable they actually have to be added to as a feature. This will produce a phonoform, which looks like the one in figure 4.2.

This leaves a couple slots in the phonoform format still open. We dont't have any information about duration, the actual F0 values or power and want CHATR to use its own prediction modules to fill these slots. Just like it does for e.g. text input. Since so far phonoform input was ony intended to be used fully specified, these extra module calls had to be added to the phonoform input function. In our case the duration, int_target and power

modules have to be called.

4.2 m2b - the Interface Program

The conversion from the TDMT format to the phonoform input format, and thus the prediction of placement and type of ToBI labels, is done as a front end to CHATR.

The TDMT output is read and parsed into a three dimensional array. The first dimension represents the lines of the TDMT format and the second dimension the four fields containing words, syntactical information, phonemes and syllables, and lexical stress. Each field of information is thus a string, which is stored in the slot of the array determined by the linenumber and fieldnumber.

The entry "BOUNDARY" in the second field introduces a C boundary at that point[1]. I will then go over all the lines between two C boundaries to decide on the placement of accents and P boundaries. After having collected this information and stored it in chained structures, I will go over these lines again to actually print the phonoform line by line, while adding the appropriate intones at the predicted places.

```
#include <stdio.h>
#include <stdlib.h>

#define BUFFSIZE        4096 /* bad limitation !! */

int put_wrd(char *wrd, int n);
int put_syl(char *syl, int n, int pitch_accent, char acc_type,
    char bound, int put_stress_mark);
int put_ph(char *ph, int n, int put_stress_mark);
int stressable(char *pos);
int vowel(char phone);

int make_phonoform(char txt[1024][4][256], int len) /* ugly */
{
  struct boundary {   /* to collect boundaries */
    int line;
    int type;
    struct boundary *next;
  };
```

[1]Theoretically "BOUNDARY" should describe a boundary between two clauses. Unfortunately, until now kommas between clauses and other kommas, like in enumerations, are not yet distinguishable by this, so that in effect every komma introduces a C boundary

```
struct accent {     /* to collect accents */
  int priority;
  int line;
  int wrd;
  char type;
  struct accent *next;
};
struct stress_mark {    /* to collect '+'able words - since the plus */
  int line;             /* is specific to sampaG, is has to be added */
  int wrd;
  struct stress_mark *next;
};

struct boundary *boundary, *current_bound, *last_bound;
struct accent *accent, *acc, *last_acc;
struct stress_mark *stress_mark, *current_stress;

int x,i,j,l,posc;
char acc_type, bound;
int n_wrd;
int wrd_x,gr_x,syl_x,ph_x,str_x;
int put_stress_mark, pitch_accent = 0;
char pos[64];

fprintf(stdout, "\n(Say (Synth (set utt\n");
fprintf(stdout, " (Utterance PhonoForm (:D nil (:S nil (:C nil (:P nil");

i=0;
while(i<len) {    /* for all lines */

  /* first we have to find, WHERE to put pitch accents and */
  /* boundary tones                                        */
  last_bound = NULL;
  current_bound = boundary =
    (struct boundary *)malloc(sizeof(struct boundary));
  last_acc = NULL;
  acc = accent =
    (struct accent *)malloc(sizeof(struct accent));
  acc->priority = 0;
  current_stress = stress_mark =
    (struct stress_mark *)malloc(sizeof(struct stress_mark));

  for(j=i;(j<len && !strstr(txt[j][1],"BOUNDARY"));j++) {
    /* for all lines until next clause boundary */
```

```
      n_wrd = 0;
      gr_x = 0;
      x=0;
      while(x<2){   /* get to POS information */
if(txt[j][1][gr_x++]==':')
  x++;
      }
      while(txt[j][1][gr_x]!='\0'){   /* until all words in that */
                                      /* line have been checked  */
while(txt[j][1][gr_x]==' ')   /* skip spaces between words */
  gr_x++;
posc = 0;
while(txt[j][1][gr_x]!=':' && /* read POS for one word */
      txt[j][1][gr_x]!=' ' &&
      txt[j][1][gr_x]!='\0')
  pos[posc++] = txt[j][1][gr_x++];
pos[posc] = '\0';

/* check, whether pos is a noun or adjective      */
/* if so, save line in boundary structure and add */
/* information to accent structure                */
if(strcmp(pos,"NOMEN")==0)
  {
    current_bound->type = 'p';
    current_bound->line = j;
    acc->priority = 15;
    acc->line = j;
    acc->wrd = n_wrd;
    acc->type = 'p';
  }
else if(strcmp(pos,"ADJEKTIV")==0)
  {
    current_bound->type = 'p';
    current_bound->line = j;
    acc->priority = 15;
    acc->line = j;
    acc->wrd = n_wrd;
    acc->type = 'p';
  }
/* if no boundary has been found yet, check what accent */
/* priority pos would get                               */
/* if it is the highest, yet, add info to accent struct */
else if(last_bound==NULL)
  {
      if(strcmp(pos,"ADVERB")==0 && acc->priority<=13)
```

```
{
  acc->priority = 13;
  acc->line = j;
  acc->wrd = n_wrd;
}
      if(strcmp(pos,"VERBZUSATZ")==0 && acc->priority<=11)
{
  acc->priority = 11;
  acc->line = j;
  acc->wrd = n_wrd;
}
      else if(strcmp(pos,"KARDINALZAHL")==0 && acc->priority<=9)
{
  acc->priority = 9;
  acc->line = j;
  acc->wrd = n_wrd;
}
      else if(strcmp(pos,"VERB")==0 && acc->priority<=7)
{
  acc->priority = 7;
  acc->line = j;
  acc->wrd = n_wrd;
}
      else if(strcmp(pos,"FIX-EXP")==0 && acc->priority<=5)
{
  acc->priority = 5;
  acc->line = j;
  acc->wrd = n_wrd;
}
      else if(strcmp(pos,"PRAEPOSITION")==0 && acc->priority<=3)
{
  acc->priority = 3;
  acc->line = j;
  acc->wrd = n_wrd;
}
      else if(acc->priority==0) /*accent on last word */
{
  acc->line = j;
  acc->wrd = n_wrd;
}
  }
if(stressable(pos)) /* if word is '+'able, memorize */
  {
    current_stress->line = j;
    current_stress->wrd = n_wrd;
```

```
    current_stress = current_stress->next = (struct stress_mark *)malloc(sizeof(struct
  }
n_wrd++;
      }
      /* if a boundary has been added in this line, start looking */
      /* for the next one and also the next accent                 */
      if(current_bound->type=='p')
{
  last_acc = acc;
  acc = acc->next = (struct accent *)malloc(sizeof(struct accent));
  acc->priority = 0;
  last_bound = current_bound;
  current_bound = current_bound->next =
                        (struct boundary *)malloc(sizeof(struct boundary));
}
    }


    free(current_stress);
    /* adjust last boundary and accent, in case no boundary was found */
    /* at all in this clause, or words are left after the last phrase */
    /* boundary                                                       */
    /* also mark type of last boundary: clause, statement, question   */
    if(last_bound!=NULL){
      free(acc);
      free(current_bound);
      acc = last_acc;
      current_bound = last_bound;
    }
    else {
      acc->next = NULL;
      current_bound->next = NULL;
    }
    current_bound->line = j-1;
    switch(txt[j][0][0]) {
    case ',':
      current_bound->type = 'c';
      acc->type = 'c';
      break;
    case '?':
      current_bound->type = 'q';
      acc->type = 'q';
      break;
    default:
      current_bound->type = 's';
```

```
      acc->type = 's';
      break;
    }

    /* write phonoform line by line, adding intones, whenever a pitch */
    /* accent is specified in the accent structure or a boundary is   */
    /* specified in the boundary structure                            */
    current_bound = boundary;
    acc = accent;
    current_stress = stress_mark;
    for(l=i;l<j;l++)
      {
/* open phrase if required by boundary structure */
if(current_bound->line==l-2) {  /* '-2' to skip line with */
  if(current_bound->type=='p')  /* only punctuation       */
    fprintf(stdout,"\n (:P nil");
  else if(current_bound->type=='c')
    fprintf(stdout,"\n (:C ((PauseLength 150)) (:P nil");
  else if(current_bound->type=='s' || current_bound->type=='q')
    fprintf(stdout,"\n (:S ((PauseLength 300)) (:C nil (:P nil");
}
n_wrd = 0;
wrd_x = syl_x = ph_x = str_x = 0;
while(txt[l][0][wrd_x]!='\0') /* until end of line */
  {
    /* put word */
    wrd_x = put_wrd(txt[l][0],wrd_x);

    while(txt[l][3][str_x]==' ') /* skip spaces between words */
      str_x++;
    while(txt[l][2][syl_x]==' ') /* skip spaces between words */
      syl_x++;
    while(txt[l][2][syl_x]!=' ' && txt[l][2][syl_x]!='\0')
      /* until end of word */
      {

/* check for a pitch accent in the accent  */
/* structure                               */
if(acc->line==l &&
   acc->wrd==n_wrd &&
   txt[l][3][str_x]=='1')
  {
    pitch_accent = 1;
    acc_type = acc->type;
    if(acc->next!=NULL)
```

```
      {
acc = acc->next;
free(accent);
accent = acc;
      }
  }
else
  pitch_accent = 0;

/* check whether '+' has to be added to first */
/* vowel in syllable                          */
if(current_stress->line==1 &&
   current_stress->wrd==n_wrd &&
   txt[1][3][str_x]=='1')
  {
    put_stress_mark = 1;
    if(current_stress->next!=NULL)
      {
current_stress = current_stress->next;
free(stress_mark);
stress_mark = current_stress;
      }
  }
else
  put_stress_mark = 0;

/* check whether boundary tones have to be added to */
/* syllable                                         */
if(txt[1][3][str_x+1]=='\0' && current_bound->line==1)
  bound = current_bound->type;
else
  bound = '\0';

/* put syllables of word */
syl_x = put_syl(txt[1][2],syl_x,pitch_accent,
acc_type,bound,put_stress_mark);


while(txt[1][2][ph_x]==' ' || txt[1][2][ph_x]=='-')
  /* skip spaces and dashes between words and syllables */
  ph_x++;
while(txt[1][2][ph_x]!='-' &&
      txt[1][2][ph_x]!=' ' &&
      txt[1][2][ph_x]!='\0') /* until end of syllable */
  {
```

```
    /* put phonemes of syllable */
    ph_x = put_ph(txt[1][2],ph_x,put_stress_mark);

  }
/* add pause at the end of sentence, sounds better */
if(bound=='s' || bound=='q')
  fprintf(stdout, "\n       (Phoneme # 0 0 ((0 0)))");
fprintf(stdout, ")");    /*close syl */
str_x++;
      }
    fprintf(stdout, ")");        /* close word */
    n_wrd++;
  }
/* close opened phrase */
if(current_bound->line==1) {
  if(current_bound->type=='p')
    fprintf(stdout,")");
  else if(current_bound->type=='c')
    fprintf(stdout,"))");
  else if(current_bound->type=='s' || current_bound->type=='q')
    fprintf(stdout,")))");
  current_bound = current_bound->next;
  free(boundary);
  boundary = current_bound;
}
      }
    i = j+1;
  }
  fprintf(stdout, "))))))\n");     /* close everything*/

  return 0;
}



/* prints one word */
int put_wrd(char *txt, int n)
{
  int i=0;
  char wrd[64];

  while(txt[n]==' ')
    n++;
```

```
  while(txt[n]!=' ' && txt[n]!='\0')
    wrd[i++]=txt[n++];
  wrd[i]='\0';

  fprintf(stdout, "\n   (Word %s nil",wrd);

  return(n);
}

/* prints one syllable */
int put_syl(char *txt, int n, int pitch_accent,
    char acc_type, char bound, int put_stress_mark)
{
    int put_glot=0, i=0;
    char syl[64];

    while (txt[n]=='-' ||
   txt[n]==' ')
n++;

    /* add glottalization in the beginning of syllable, if it starts */
    /* with a vowel                                                   */
    if((vowel(txt[n]) || txt[n]=='@' || txt[n]=='6') && put_stress_mark)
      {
put_glot = 1;
syl[i++] = 'Q';
syl[i++] = '.';
      }
    while(txt[n]!='-' && txt[n]!=' ' && txt[n]!='\0')
syl[i++]=txt[n++];
    syl[i]='\0';

    /* write syllables and intones, appropriate vor the accents and */
    /* boundaries                                                   */
    if(pitch_accent)
      {
switch(acc_type) {
case 'p':
  fprintf(stdout, "\n     (Syl %s ((Stress 1) (Intones H*",syl);
  break;
case 'c':
  fprintf(stdout, "\n     (Syl %s ((Stress 1) (Intones L+H*",syl);
  break;
case 'q':
  fprintf(stdout, "\n     (Syl %s ((Stress 1) (Intones L*+H",syl);
```

```
  break;
case 's':
  fprintf(stdout, "\n     (Syl %s ((Stress 1) (Intones H+L*",syl);
  break;
default:
  fprintf(stdout, "\n     (Syl %s ((Stress 1) (Intones H*",syl);
  break;
}
switch(bound) {
case 'p':
  fprintf(stdout, " L-))");
  break;
case 'c':
  fprintf(stdout, " H-L%%))");
  break;
case 's':
  fprintf(stdout, " L-L%%))");
  break;
case 'q':
  fprintf(stdout, " H-H%%))");
  break;
default:
  fprintf(stdout, "))");
  break;
}
      }
    else
      {
switch (bound) {
case 'p':
  fprintf(stdout, "\n     (Syl %s ((Intones L-))",syl);
  break;
case 'c':
  fprintf(stdout, "\n     (Syl %s ((Intones H-L%%))",syl);
  break;
case 's':
  fprintf(stdout, "\n     (Syl %s ((Intones L-L%%))",syl);
  break;
case 'q':
  fprintf(stdout, "\n     (Syl %s ((Intones H-H%%))",syl);
  break;
default:
  fprintf(stdout, "\n     (Syl %s ()",syl);
  break;
}
```

```
      }
    if(put_glot)
      fprintf(stdout, "\n       (Phoneme Q 0 0 ((0 0)))");
    return(n);
}

/* prints one phoneme */
int put_ph(char *txt, int n, int put_stress_mark)
{
    int i=0;
    char ph[64];

    while (txt[n]=='-' || txt[n]==' ')
n++;

    /* add stress sign '+', if word is '+'able and phoneme is a vowel*/
    if(put_stress_mark && vowel(txt[n])){
      ph[i++] = '+';
      put_stress_mark = 0;
    }

    while(txt[n]!='.')
      {
ph[i++]=txt[n++];
      }
    ph[i]='\0';

    n++; /* get past the '.' */

    fprintf(stdout, "\n       (Phoneme %s 0 0 ((0 0)))", ph);

    return(n);
}


int stressable(char *pos)
{
  if(strcmp(pos,"NOMEN")==0 ||
     strcmp(pos,"VERB")==0 ||
     strcmp(pos,"VERBZUSATZ")==0 ||
     strcmp(pos,"ADVERB")==0 ||
     strcmp(pos,"FRAGEADVERB")==0 ||
     strcmp(pos,"ADJEKTIV")==0 ||
     strcmp(pos,"INTERROGATIVPRONOMEN")==0 ||
     strcmp(pos,"KARDINALZAHL")==0 ||
```

```
      strcmp(pos,"ORDINALZAHL")==0 ||
      strcmp(pos,"FIX-CAP")==0 ||
      strcmp(pos,"FIX-UP")==0 ||
      strcmp(pos,"STOP-WORD")==0 ||
      strcmp(pos,"FIX-END")==0 ||
      strcmp(pos,"FIX-INTRO")==0)
       return(1);
  else
       return(0);
}

int vowel(char phone)
{
  if(phone=='a' ||
     phone=='o' ||
     phone=='0' ||
     phone=='u' ||
     phone=='U' ||
     phone=='y' ||
     phone=='Y' ||
     phone=='i' ||
     phone=='I' ||
     phone=='e' ||
     phone=='E' ||
     phone=='2' ||
     phone=='9')
    return(1);
  else
    return(0);
}

/* Reads in the TDMT format and saves it into a 3-dimensional */
/* array. The first dimension being the lines, the second the */
/* four fields of the TDMT format (words, syntax, phonemes,   */
/* lexical stress) and the third the characters, i.e. each    */
/* TDMT field is stored as a string                           */
int main()
{
    char  buff[BUFFSIZE];
    int i,j, k, n, len, ch_n;
    char for_chatr[2048]; /* ugly */
    char txt[1024][4][256]; /* ugly */

    while ((len=read(0, buff, sizeof buff)) > 0) { /* ugly */
i=j=k=n=ch_n=0;
```

```
while( i++ < len ){

    while( buff[i]=='\"') i++;

    txt[n][k][j++]=buff[i];

    /* parse the structure into four fields */
    if( buff[i]=='|' || buff[i]=='\n'){
      txt[n][k][j-1]='\0';
      j=0;
      k++;
      if(k==4){
  while(txt[n][2][j]!='\0')
      for_chatr[ch_n++]=txt[n][2][j++];
  for_chatr[ch_n++]=' ';
  n++;
  j=0;
  k=0;
      }
  }
}
    }
    for_chatr[ch_n++]='\0';

    /* this is where the actual work gets done */
    make_phonoform(txt,n-1);

    exit(0);
}
```

```
  "bitte|SATZ:INTRO:FIX-INTRO|b.I.-t.@.|10
?|SATZ:BOUNDARY-END:INTERPUNKTION|4|0
es scheint|SATZ:INTRO:FIX-INTRO FIX-INTRO|@.s. S.aI.n.t.|0 1
,|SATZ:BOUNDARY:INTERPUNKTION|2|0
die gebuehr|SATZ:SUBJECT:DETERMINATIV NOMEN|d.i:. g.@.-b.y:6.|1 01
,|REL-S:BOUNDARY-START:INTERPUNKTION|2|0
die|REL-S:INTRO:RELATIVPRONOMEN|d.i:.|1
ich|REL-S:SUBJECT:PERSONALPRONOMEN|I.C.|1
im reisefuehrer|REL-S:F-MIDDLE:PRAEPOSITION NOMEN|I.m. r.aI.-z.@.-f.y:.-r.6.|0 1000
gesehen|REL-S:V-INF:VERB|g.@.-z.e:.-@.n.|010
habe|REL-S:V-FIN:HILFSVERB|h.a:.-b.@.|10
,|SATZ:BOUNDARY:INTERPUNKTION|2|0
ist|SATZ:V-FIN:HILFSVERB|I.s.t.|1
anders|SATZ:VP-ADVERB:ADVERB|a.n.-d.@.r.s.|10
.|SATZ:BOUNDARY-END:INTERPUNKTION|4|0
"
```

Figure 4.1: The TDMT output

```
(Utterance PhonoForm (:D nil
 (:S nil (:C nil (:P nil
   (Word drei nil
     (Syl d.r.aI. ()
       (Phoneme d 0 0 ((0 0)))
       (Phoneme r 0 0 ((0 0)))
       (Phoneme +aI 0 0 ((0 0)))))
   (Word stuecke nil
     (Syl S.t.Y. ((Stress 1) (Intones H+L*))
       (Phoneme S 0 0 ((0 0)))
       (Phoneme t 0 0 ((0 0)))
       (Phoneme +Y 0 0 ((0 0))))
     (Syl k.@. ((Intones L-L%))
       (Phoneme k 0 0 ((0 0)))
       (Phoneme @ 0 0 ((0 0)))
       (Phoneme # 0 0 ((0 0)))))))
 (:S ((PauseLength 300)) (:C nil (:P nil
   (Word oder nil
     (Syl Q.o:. ((Intones L*+H))
       (Phoneme Q 0 0 ((0 0)))
       (Phoneme +o: 0 0 ((0 0))))
     (Syl d.6. ((Intones H-H%))
       (Phoneme d 0 0 ((0 0)))
       (Phoneme 6 0 0 ((0 0)))
       (Phoneme # 0 0 ((0 0)))))))))
```

Figure 4.2: The input to CHATR derived from a TDMT output

Chapter 5

Text Input

5.1 Adding Information to the Sphrase Tree

Now, in the case of text input, we have even slightly less information than we receive from the TDMT system. Luckily, there is a lexicon for German, which contains part-of-speech. This means, we have to rely on punctuation and part-of-speech to do our phrasing. We don't have any information about sentence type as we did with the input from the TDMT system, except for what is conveyed by punctuation ('.' vs '?'). Thus we cannot distinguish between different uses of kommas anymore, so that we just have to assume, that every komma means the end of an intonation phrase, which is not always the case as e.g. in enumerations. Also, we have to slightly change our rules for the grouping of words into intermediate phrases, because don't have any information about topological fields. So we will put an intermediate phrase boundary after every noun and adjective, which is not followed by a noun or adjective. The rules on placement of pitch accent are actually pretty much the same. We just adapted them to the categories that are distinguished by the lexicon.

As I mentioned in chapter 3, text input gets transformed into a very flat Sphrase tree (cf. figure 3.1). But as I also mentioned before, the Sphrase stream structure is capable of holding a lot more information than the one in figure 3.1 does. We will use this fact by adding information about phase boundaries and accents to the Sphrase stream, which later get translated into ToBI labels. The function text_input.c in file hlp/hlp.c will now look like this, so that phrasing and placement of pitch accents will be predicted by our rules, if the variable 'text_prosody_strategy' is set to "POS", otherwise text input will be treated as before.

```
void text_input(Utterance utt)
{
  /* A text input mode, basically to fill in the gap between   */
  /* TTS and HLP                                               */
```

```
  /* simply converts the input to an HLP tree and continues in */
  /* the same way
                                                        */
  List input, prosody_strategy;

  /* P_Message("*** hlp_input.c/text_input ***"); */
  if (stringp(UTTERANCE(utt)) == FALSE)
  {
    P_Error("Utterance contents not a string in Text utterance type");
    list_error(On_Error_Tag);
  }

  input = text_to_hlp(STRVAL(UTTERANCE(utt)));

  utt_set_stream("Sphrase",hlp_build_sphrase(input,utt),utt);

  /******NEW*NEW*NEW*NEW*NEW*NEW*NEW*NEW*NEW*NEW*NEW*NEW********/

  prosody_strategy = list_str_eval("text_prosody_strategy",NULL);

  if(list_sequal("POS",prosody_strategy))
      g_predict_phracc(utt);

  /******WEN*WEN*WEN*WEN*WEN*WEN*WEN*WEN*WEN*WEN*WEN*WEN*******/

  get_and_apply_options(utt); /* just above; tony */

  list_free_tree(input);
}
```

After the function g_predict_phracc enriched the Sphrase tree with additional information, it will look like the one in figure 5.1. We again want to think of the features '(PhraseLevel :P)' as the end of an intermediate phrase and of '(PhraseLevel :C)' as the end of an intonation phrase. I used the attributes 'PAccent', 'CAccent' and 'Accent' to distinguish between the last pitch accents in an intonation phrase, the last pitch accent in an intermediate phrase and all other pitch accents. As we saw earlier the intones assigned to them may differ.

This structure now takes the same course an Sphrase tree built from HLP input would. (The stages an HLP input will go through are very nicely decribed in [5], Tony Hébert's report.) It first passes through the hlp_module, where the information contained in the Sphrase tree is reorganised and further processed. If the input was a phrase structure tree e.g. the user might choose to do prosody prediction on the basis of the syntactic structure, which would result in an Sphrase tree very similar to ours from figure

```
(((CAT D))
  (((CAT S) (IFT Statement))
    (((LEX als)))
    (((LEX der)))
    (((PAccent +) (LEX Frosch)))
    (((PhraseLevel :P) (LEX ueber)))
    (((LEX die)))
    (((CAccent +) (LEX Wiese)))
    (((PUNC ,) (LEX sprang)))
    (((PhraseLevel :C) (LEX wurde)))
    (((LEX er)))
    (((LEX vom)))
    (((Accent +) (LEX Storch)))
    (((PUNC .) (LEX gefressen)))
  (((CAT S) (IFT Question))
    (((LEX ist)))
    (((LEX das)))
    (((LEX nicht)))
    (((PAccent +) (PUNC ?) (LEX traurig)))))
```

Figure 5.1: The Enriched Sphrase Tree

5.1. That means of course that we want to skip some of the processing in the hlp_module, because we already have that information. Setting the variables "HLP_phrase_strategy" and "HLP_prosodic_strategy" to "None" does it.

How do we now get from phrase boundaries and accents to the appropriate intones? We have to define a set of rules and assign it to a variable called "HLP_Patterns", which are then used by the function hlp_realise_accents, the last function in the hlp_module, to translate boundary and accent features into ToBI labels. In our case, they might look like the ones in figure 5.2.

Intones will now be assigned to words like this:

```
als der Frosch ueber die Wiese sprang # wurde er vom Storch
         H* L-           L+H*    H-L%                 H+L*
gefressen # ist das nicht traurig
     L-L%                L*+H H-H%
```

```
(set  HLP_Patterns
 '((Statement (START )
       (Accent (+ (H+L*)))
       (PAccent (+ (H*)))
       (CAccent (+ (L+H*)))
       (PPHRASE (L-))
       (CPHRASE (H-L%))
       (TAIL (L-L%)))
   (Question (START )
       (Accent (+ (L*+H)))
       (PAccent (+ (H*)))
       (CAccent (+ (L+H*)))
       (PPHRASE (L-))
       (CPHRASE (H-L%))
       (TAIL (H-H%)))))
```

Figure 5.2: An Example for HLP Patterns

5.2 g_predict_phracc - Predicting Prosody from Text Input

Here is the function that predicts the accents and phrase boundaries and adds them to the Sphrase tree.

It runs over the word stream and puts a C boundary, whenever it encounters any punctuation. The S boundaries are already added by CHATR (cf. 3.1). For every word, the part-of-speech is looked up in the lexicon. A phrase boundary is added after every noun or adjective, which is not followed by another noun or adjective. For the placement of accents, we used, similar as before, a method of assigning to them a certain priority. If no noun or adjective is found, the word with the highest priority gets accented.

```
static void g_predict_phracc(Utterance utt)
{
  Stream w,v;
  List punct, entry, cat;
  int pbound=0, cbound=0;
  int i, acc_pos=0, acc_grade=0, last_pbound=0, last_cbound=0, wordx;
  char *word, *acc_type="PAccent";

  /* go over all words in the word stream */
  for(w=utt_stream("Word",utt),wordx=1;w!=SNIL;w=SC_next(w),wordx++){

    word = SC(w,Word)->text; /* get word */
```

```
entry = lex_lookup(word,NIL); /* lookup word in lexicon */
cat = list_assoc_str("CAT",list_last(entry)); /* get POS */

if(cbound)
  /* put C boundary if specified */
  {
    SC(w,Word)->features =
      cons(make_fpair_from_str("PhraseLevel",":C"),SC(w,Word)->features);
    cbound = 0;
    pbound = 0;
  }

if(!list_sequal("N",car(cdr(cat))) &&
   !list_sequal("A",car(cdr(cat))) &&
   pbound)
  /* put P boundary if specified and word is no content word */
  /* put accent on previous word - that is the one that      */
  /* triggered the P boundary                                */
  {
    SC(w,Word)->features =
      cons(make_fpair_from_str("PhraseLevel",":P"),SC(w,Word)->features);
    SC(SC_previous(w),Word)->features =
      cons(make_fpair_from_str(acc_type,"+"),
           SC(SC_previous(w),Word)->features);

    pbound = 0;
  }

if((list_sequal("N",car(cdr(cat))) ||
    list_sequal("A",car(cdr(cat)))) &&
   SC_next(w)==SNIL)
    /* if word is a content word and also last word in stream, */
    /* put an accent                                           */
  {
    SC(w,Word)->features =
      cons(make_fpair_from_str(acc_type,"+"),SC(w,Word)->features);
    pbound = 1;
    acc_grade = 1;
    acc_pos = wordx;
    last_pbound = wordx;
  }
else if(list_sequal("N",car(cdr(cat))) ||
        list_sequal("A",car(cdr(cat))))
  /* if word is a content word, it possibly needs a pitch accent */
  /* and introduces a boundary - store that info                 */
```

```
  {
    pbound = 1;
    acc_grade = 1;
    acc_pos = wordx;
    last_pbound=wordx;
  }
/* for other POS, grade is higher than what you have so far */
/* the word might become pitch accent - store info          */
else if(list_sequal("ADV",car(cdr(cat))) && acc_grade>=3)
  {
    acc_grade = 3;
    acc_pos = wordx;
  }
else if(list_sequal("NUM",car(cdr(cat))) && acc_grade>=5)
  {
    acc_grade = 5;
    acc_pos = wordx;
  }
else if(list_sequal("PREP",car(cdr(cat))) && acc_grade>=7)
  {
    acc_grade = 7;
    acc_pos = wordx;
  }
else if(list_sequal("V",car(cdr(cat))) && acc_grade>=9)
  {
    acc_grade = 9;
    acc_pos = wordx;
  }

/* words with a punctuation feature are special -> C boundary */
punct = list_assoc_str("PUNC",SC(w,Word)->features);
if(punct != NIL)
  {
    cbound = 1;

    /* accent type is dirfferent, depending on whether it is the */
    /* last in a sentence or the last in a clause                */
    if(list_sequal(",",car(cdr(punct))) ||
       list_sequal(":",car(cdr(punct))))
      acc_type = "CAccent";
    else
      acc_type = "Accent";

    /* adjust last boundary and accent */
    /* have to run over stream again   */
```

```
if(last_cbound<last_pbound && last_pbound<wordx)
  /* found noun or adjective within clause */
  /* just change accent type and remove last P boundary */
  {
    for(v=utt_stream("Word",utt),i=1;i<=wordx;v=SC_next(v),i++)
      {
        if(i==acc_pos)
          {
            SC(v,Word)->features =
              list_remove(make_fpair_from_str("PAccent","+"),
                          SC(v,Word)->features);
            SC(v,Word)->features =
              cons(make_fpair_from_str(acc_type,"+"),
                   SC(v,Word)->features);
          }
        if(i==last_pbound+1)
          {
            SC(v,Word)->features =
              list_remove(make_fpair_from_str("PhraseLevel",":P"),
                          SC(v,Word)->features);
          }
      }
  }
else if(last_pbound<=last_cbound)
  /* found no noun or adjective within clause */
  if(acc_grade!=0)
    /* put pitch accent on word specified in acc_pos */
    {
      for(v=utt_stream("Word",utt),i=1;i<=wordx;v=SC_next(v),i++)
        {
          if(i==acc_pos)
            SC(v,Word)->features =
              cons(make_fpair_from_str(acc_type,"+"),
                   SC(v,Word)->features);
        }
    }
  else
    /* if nothing is specified, just put pitch accent on last word */
    SC(w,Word)->features = cons(make_fpair_from_str(acc_type,"+"),
                                SC(w,Word)->features);

/* reset counters etc. */
acc_type = "PAccent";
acc_grade = 0;
last_cbound=wordx;
```

```
    }
  }
}
```

5.3 Along the Way Bug Fixes for CHATR

While implementing our prediction rules for text input, I came across a couple of problems which could only be solved by making minor changes to CHATR's source code. I will explain these changes in this section.

5.3.1 text.c: text_sentence_head

As you might have noticed CHATR translates a '?' in the end of a sentence into the feature '(IFT Question)' as opposed to '(IFT Statement)', which is the default. The value of the IFT attributes are later used by the HLP_Patterns to select the appropriate ToBI labels. This very rough approximation of sentence type using punctuation is done in the function text_sentence_head in file text/tetx.c, if you replace

```
if (list_sequal("?",punc))
   ift = "Question";
```

by

```
if (list_sequal("?",car(cdr(punc))))
   ift = "Question";
```

The value of the variable punc will be a list, the first element of which is "PUNC" and the second element of which is the actual punctuation. So you really want to compare only the second element to "?" and not the whole list.

5.3.2 lexicon.c: lex_lookup

In order to make text input as natural as possible, the function, which does the lexicon lookup has to be able to deal with words that appear capitalized in the input text for some reason (like standing at the beginning of a phrase), but are not listed in the lexicon as capitalized. So far, this problem has been avoided by doing only lowercase lexicon lookup. For German, useful information about part-of-speech is lost that way[1]. So, what we want to do, is to search for the word in the lexicon and then, only if we cannot find it that way, try to find the lowercased version. That adds a couple of lines to the function lex_lookup in the file lex/lexicon.c.

[1]Since all nouns are capitalized in German, there are cases, in which verbs e.g can be distinguished from nouns by capitalization.

```
List lex_lookup(char *text,List features)
{
  /* looks up the word in the lexicon */
  /* at present only deal with first entry in lexicon for given word */
  /* Returns a copy of the entry */
  List entry;
  char *ltext;

  /*******OLD*OLD*OLD*OLD*OLD*OLD*OLD*OLD*OLD*OLD*OLD*OLD*OLD*OLD*OLD*OLD*********/

  /* ltext = s_to_lower(xstrdup(text)); */

  /*******LDO*LDO*LDO*LDO*LDO*LDO*LDO*LDO*LDO*LDO*LDO*LDO*LDO*LDO*LDO*LDO*********/

  if ((entry = addenda_lookup(text,features,current_lex->addenda)) != NIL)
  {
    if (current_lex->phone_set != grammar.int_ph)
       entry = lex_map_phonemes(entry);
  }
  else if ((current_lex->lfd != NULL) &&
          ((entry = find_matching_entry(text,features)) != NIL))
  {
    if (current_lex->phone_set != grammar.int_ph)
       entry = lex_map_phonemes(entry);
  }
  else {

  /*******NEW*NEW*NEW*NEW*NEW*NEW*NEW*NEW*NEW*NEW*NEW*NEW*NEW*NEW*NEW*NEW*******/

    ltext = s_to_lower(xstrdup(text));

    if ((entry = addenda_lookup(ltext,features,current_lex->addenda)) != NIL)
    {
      if (current_lex->phone_set != grammar.int_ph)
         entry = lex_map_phonemes(entry);
    }
    else if ((current_lex->lfd != NULL) &&
            ((entry = find_matching_entry(ltext,features)) != NIL))
    {
      if (current_lex->phone_set != grammar.int_ph)
         entry = lex_map_phonemes(entry);
    }

  /*******WEN*WEN*WEN*WEN*WEN*WEN*WEN*WEN*WEN*WEN*WEN*WEN*WEN*WEN*WEN*WEN*******/
```

```
    else if (streq(current_lex->onfail,"JLTS"))
      entry = mlts_lookup(text,features,"JLTS");
    else if (streq(current_lex->onfail,"GLTS"))
      entry = mlts_lookup(text,features,"GLTS");
    else if (streq(current_lex->onfail,"KLTS"))
      entry = mlts_lookup(text,features,"KLTS");
    else if (streq(current_lex->onfail,"CLTS"))
      entry = mlts_lookup(text,features,"CLTS");
    else if (streq(current_lex->onfail,"LTS"))
      entry = lts_lookup(ltext,features);
    else if (streq(current_lex->onfail,"ERROR"))
    {
      P_Error("Word %s not in lexicon",text);
      xfree(ltext);
      list_error(On_Error_Tag);
      return NIL;
    }
    xfree(ltext);
  }
  return entry;
}
```

5.3.3 hlp.c: hlp_apply_actions

The function hlp_apply_actions in file hlp/hlp.c gets called during the execution of the function hlp_realise_accents and does the actual annotation of words with ToBI labels according to the HLP patterns. As far as phrasal boundaries are concerned, it could only deal with the attribute 'PHRASE' until now, which would add boundary tones to the words before C boundaries. Since we wanted to distinguish between intonation phrases and intermediate phrases by using C boundaries as intonation phrase boundaries and P boundaries as intermediate phrase boundaries, I had to extend the function in such a way that it could handle the attributes 'PPHRASE' and 'CPHRASE'. In addition to

```
if ((act=getfvalue("PHRASE",actions)) != NIL)
{
  lword = hlp_last_word(item);
  for (word = hlp_first_word(item); word != lword; word = SC_next(word))
  {
    if (SC(word,Word)->right_boundary > P_BOUND)
      SC(word,Word)->intones =
        list_append(list_copy_tree(act),SC(word,Word)->intones);
  }
}
```

we now have

```
if ((act=getfvalue("CPHRASE",actions)) != NIL)
{
  lword = hlp_last_word(item);
  for (word = hlp_first_word(item); word != lword; word = SC_next(word))
  {
    if (SC(word,Word)->right_boundary > P_BOUND)
      SC(word,Word)->intones =
    list_append(list_copy_tree(act),SC(word,Word)->intones);
  }
}
if ((act=getfvalue("PPHRASE",actions)) != NIL)
{
  lword = hlp_last_word(item);
  for (word = hlp_first_word(item); word != lword; word = SC_next(word))
  {
    if (SC(word,Word)->right_boundary == P_BOUND)
      SC(word,Word)->intones =
    list_append(list_copy_tree(act),SC(word,Word)->intones);
  }
}
```

5.3.4 intonation.c: add_intonation

As I explained above, the function hlp_realise_accents assigns ToBI labels to words. For the prediction of F0 values the int_target_module needs an assignment from ToBI labels to syllables, though. This basically means finding the syllable which carries lexical stress within the relevant word for the pitch accent and the last syllable within that word for the boundary tones. Unfortunately, the function add_intonation in file intonation/intonation.c didn't work that way. It just took the assigned ToBI labels and put them on each syllable in that word.

Here comes my new version of add_intonation. I am assuming, that no word can have more than two ToBI labels, i.e. one pich accent and one boundary tone. I am also hoping, that no word has more than one syllable marked with lexical stress. I am aware of the fact that a word might have more than one syllable carrying lexical stress, but since I could not decide, which one is the one that gets the pitch accent anyway, I will just take the first lexically stressed syllable and stop searching.

```
void add_intonation(Utterance utt)
{
  /* build the first level of intonation stream from the word input */
  /* lots of modifications made by Tony */
```

```
  /* more modifications made by KS */

    int sylx,x;
    Stream previous = SNIL;
    Stream start = SNIL;
    List intones,i,pitch_acc,bound_tone;
    Stream newcell;
    Stream words,w,s,intone,syl;

    /* P_Message ("*** add_intonation ***"); */

    sc_delete_stream("Intone",utt);

    for(w=WORDSTREAM(utt);w!=SNIL;w=SC_next(w))
      {
pitch_acc=NIL; bound_tone=NIL;
intones = SC(w,Word)->intones;
sylx = list_length(car(cdr(SC(w,Word)->lexentry))); /* # of syls */

/*maximal 2 Intones on word - one pitch accent and one boundary tone*/
if(list_length(intones)>2)
  {
    /* P_Message("too many intones on word %s",SC(w,Word)->text); */
  }
       /* check, what kind of ToBI labels are specified for that word*/
for(i=intones;i!=NIL;i=cdr(i))
  {
    if(strstr(STRVAL(car(car(i))),"*"))
pitch_acc = car(i);
    else
bound_tone = car(i);
  }

/*go over syllables matching the word*/
for(s=Rsyl1(w),x=1;s!=SNIL;s=SC_next(s),x++)
  {
           /* if syllable carries lexical stress and there is a pitch */
           /* accent specified for that word, link the appropriate    */
           /* intones to that syllable                                */
    if(SC(s,Syl)->lex_stress==1 && pitch_acc!=NIL)
      {
newcell = make_intonation_cell(pitch_acc);
SC_previous(newcell) = previous;
if(start==SNIL)
  start = newcell;
```

```
if(previous!=SNIL)
  {
    SC_next(previous) = newcell;
  }
link_stream_cells(w,newcell);
link_stream_cells(s,newcell);
previous = newcell;

pitch_acc = NIL;
      }
            /* if syllable is last one and boundary tones are */
            /* specified, link them */
    if(x==sylx && bound_tone!=NIL)
      {
newcell = make_intonation_cell(bound_tone);
SC_previous(newcell) = previous;
if(start==SNIL)
  start = newcell;
if(previous!=SNIL)
  {
    SC_next(previous) = newcell;
  }
link_stream_cells(w,newcell);
link_stream_cells(s,newcell);
previous = newcell;
      }
  }
      }
    utt_set_stream("Intone",start,utt);
}
```

Chapter 6

F0 Prediction

CHATR offers two methods for predicting the actual F0 values. Since we were using ToBI labels, we had a choice between a linear regression technique and an implementation of Anderson, Pierrehumbert and Liebermann's technique, which is described in [1]. The linear regression model turned out be trained on data from a news corpus, so that it was almost impossible to predict questions. Therefore we started to use the technique by Anderson et. al.

This approach calculates target values for F0 by just adding or subtracting a bit from the speakers mean according to the ToBI label or the combination of ToBI labels on the relevant syllable. Later a curve is interpolated from these target points and decline is added.

To model the meaning of the ToBI labels (especially the complex ones) well, you need to place several targets on one syllable. E.g. to get a rise on the syllable labeled with H-H%. Also, there might be cases, where one ToBI label expands its influence over more than one syllable. The label L*+H for example should trigger the placement of a high target value on the following syllable. This makes the right timing of the targets a bit complicated. And in fact, in the beginning, CHATR shuffeld the intones pretty well before placing them. I did some adjustments[1] to the timing, so that now the only time a problem occurs is, when a label like L*+H is placed on the last syllable of a phrase. In that case, the last target gets placed after the boundary tone. With our rules that happens whenever the last syllable in a question carries a pitch accent. This problem could probably be fixed by compressing the span of the label a bit and then shifting it some towards th ebeginning of the utterance.

[1] changes only affected the function tobi_make_targets_apl in file intonation/ToBI.c

Chapter 7

Conclusion

Our approach to prosody prediction is based on very rough heuristics. There are far more elaborate methodes, which would be very interesting to implement and compare. Unfortunately, we didn't have time to run a lot of test, so everything we can say about the quality of our prediction is quite impressionistic. For simple "everyday" utterances like they are e.g. produced by the TDMT system, we were able to produce acceptable results. But when the sentences became more complex, the intonation started to sound strange. Especially if the utterance required some special focusing or envolved any kind of emotion. Of course, we are not able to predict these things form just part-of-speech and a little bit of syntactic information. One really needs a lot of semantic and pragmatic information for that.

One point, we had to struggle a lot with, was the unit selection. We had to put the weight on the unit cost really high (ratio 5 to 1 to the concatenation cost), in order for the predicted F0 values to have an effect on the unit selection at all. This of course, sometimes led to a loss of smoothness in the concatenation. But even with this much weigth on the unit cost, the F0 value of the selected units was sometimes exactly the opposite to what we predicted. Probably a still bigger database, preferably with a greater variety of prosodic patterns than contained in the current database, could solve this problem.

We made an interesting observation, when playing around with the beginning of a fairy tale. It sounded quite bad, although the sentences were not very complicated. After a while we noticed, that the quality actually was not that much different from the other examples we had, but our acceptance for these sentences seemed to be a lot stricter. So apparently, the standards, by which we judge speech, changes depending on the type of the utterance we are listening to and maybe also on the person who is speaking.

Appendix A

Some Useful Hints on Variable Settings

For the German speakers you should set the parameter 'Int_Method' to 'ToBI'. There you have two possibilities: one is to use the linear regression model (which was trained on English(!) news and therefore has a very limited pitch range) and the other one is the method by Anderson et. al. This is set via the 'ToBI_params'. The ToBI parameter 'target_method' has to be set to either 'apl' or 'lr'.

If you choose 'apl', there are a lot of additional ToBI parameters, where the pitch range of the speaker can be tuned. They are described in the CHATR manual.

If you use text input (or HLP input) and want the prosody to be predicted in the hlp module and later realised according to your HLP patterns, you have to prevent the intone_module from going over the intone stream again and predicting different things. The intone module is skipped if the variable 'HLP_realise_strategy' is set to 'Simple_Rules'.

If you want to use our prosody prediction for text input, you have to set the variable 'text_prosody_strategy' to 'POS'. And if you want our prediction to be the only one done on your text, you have to set the variables 'HLP_prosodic_strategy' and 'HLP_phrase_strategy' to 'None'.

Bibliography

[1] 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*, New York, 1984. IEEE.

[2] Caren Brinckmann. German in eight weeks - a crash course for chatr. Technical report, ATR, Kyoto, Japan, 1997.

[3] Nick Campbell. A step in the direction of synthesising natural sounding speech.

[4] Nick Campbell. Chatr: A high-definition speech re-sequencing system. In *Proceedings of the 3rd ASA/ASJ Joint Meeting*, pages 1223–1228, Hawaii, 1996.

[5] Tony Hébert. Prosody within CHATR. Technical report, ATR, Kyoto, Japan, 1997.

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

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

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

[9] Martyn Weeks. *CHATR - a generic speech synthesis system.* ATR-ITL, http://www.itl.atr.co.jp/~mweeks/.