TR-IT-0202

Prosody within CHATR

Tony Hebert & Nick Campbell

1997.01

ABSTRACT
Prosody makes a synthesized utterance sound more human, more intelligible. My work has consisted in improving the prosody processing within CHATR. This task has involved debugging and tracing the existing speech synthesis code, in order to determine which points had to be improved. This analysis enabled me to realize a detailed CHATR map, a document which shows step by step the whole prosody processing within CHATR. This is given in the chapter 1. Chapter 2 explains which parts of CHATR code I have modified and why. I also describe code I have written which might be integrated in future into CHATR. The third chapter of this report discusses the possible use of a parser from Dept3. This module is likely to improve the pause prediction. I also describe an algorithm to use this new module, the first results and how the new module should be integrated into CHATR.

Prosody within CHATR

by Tony HEBERT
Institut National des Télécommunications, Evry - FRANCE

ATR-ITL January 1997

Thanks

At INT I would like to thank Mr Guedj for having helped me to go and do this training period in Japan, a country I really wanted to discover. And I would like also to thank Mrs Jacquet, Mr Guedj's secretary, for having been so nice and so helpful.

At ATR my thanks go to Higuchi-san for the opportunity he gave me to work in his department and to Nick Campbell, my superviser, from whom I have learnt a lot, especially about team managing. And I am greatly thankful too towards all ATR people for their kindness; it has really been a pleasure to work in ATR.

Abstract

Prosody makes a synthesized utterance sound more human, more intelligible.
My work during the six months I have spent in ATR-ITL has consisted in improving the prosody processing within CHATR, the speech synthesizer developed by the members of Dept 2 of ITL.
This task has involved first to understand the existing speech synthesis piece of code, in order to determine which points had to be improved. This analysis enabled me to realize a detailed CHATR map, that is a document which shows step by step the whole prosody processing within CHATR. This is the chapter 1 of this report.
The chapter 2 explains which parts of CHATR code I have modified and why. In this chapter I am also dealing with some parts of code I have written and which might be integrated in future into CHATR.
And finally in the third chapter of this report, I discuss the possible use of a module realized by some members of Dept3. This module is likely to improve the pause prediction. I also describe an algorithm I have written to use this new module, the first results and how the new module should be integrated into CHATR.

Le rôle de la prosodie consiste a rendre la synthèse vocale plus humaine, plus intelligible.
Mon travail pendant les six mois que j'ai passés à ATR-ITL a consisté à améliorer le traitement de la prosodie au sein de CHATR, le synthétiseur de parole développé par les membres du département 2 d'ITL.
Cette tâche a impliqué tout d'abord de comprendre le code existant, afin de déterminer ce qui devait être amélioré. Cette analyse m'a permis de réaliser une "carte" détaillée de CHATR, c'est-à dire un document montrant étape par étape tout le traitement de la prosodie au sein de CHATR. Ceci constitue le premier chapitre de ce rapport.
Le second chapitre explique quelles parties du code de CHATR j'ai faites modifiées et pour quelles raisons. Dans ce chapitre je decris aussi des parties de code que j'ai ecrites et qui pourraient être intégrées à l'avenir dans CHATR.
Et finalement dans le troisieme chapitre de ce rapport, je discute de l'utilisation possible d'un module réalisé par des membres du département 3. Ce module pourrait améliorer la prédiction des pauses. Je décris également un algorithme que j'ai écrit afin d'exploiter ce nouveau module, les premiers résultats et la manière selon laquelle ce nouveau module devrait être intégré dans CHATR.

Contents

Chapter 1

The alignment of intones to syllables

This chapter explains how CHATR deals with prosody. To be more precise, it is about how prosodic features are aligned with syllables. At the time I wrote this document, I was working on intonation within CHATR since four months. This is not a long time; but the knowledge I have acquired will help, I hope, anybody who will have to work on prosody within CHATR. Given an utterance and its type (text, HLP, PhonoWord...), the whole path this utterance is following within CHATR is described. This is very useful, as CHATR is really a huge program, in which it is easy to get lost. But first an overview of STREAMS and UTTERANCE TYPES may be useful.

1.1 Streams

A stream is defined as a level of an utterance. A simple example should be sufficient to understand this nevertheless important notion :

"I have a car"

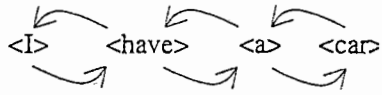
is an utterance. Several streams can be attached to it, such as :

```
- the WordStream (for words)       ( <I> <have> <a> <car> )

- the SylStream (for syllables)     ( <ai> <hav> <@> <kaa> )

- ...

- the IntoneStream (for intones)  ( <H*> <L-L%> )
```

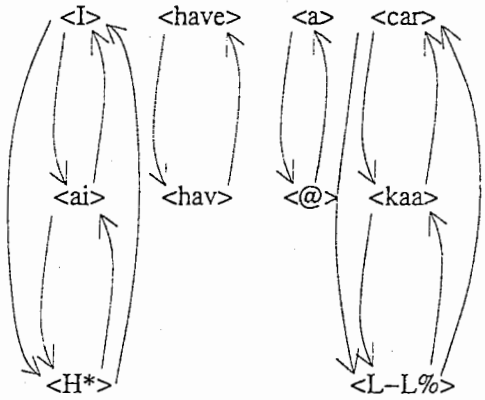
(H*, L-L% are symbols belonging to the ToBI intonation method)

Other kinds of stream exist, such as the PhoneStream (for phonemes), the SegStream (for segments), the PphraseStream (for prosodic phrases),...; but to remember only the WordStream, the SylStream and the IntoneStream is enough to understand what is going to follow.

A stream is a sequel of cells < ... > . Each cell is linked by pointers to the previous one and to the next one :



But, and that is very important, cells of different streams are also linked between each others. Thus in the example above, <ai> and <I> would be linked; the same for <hav> and <have>, <H*> and <I> (if H* was put on "I"),... :



1.2 Utterance Types

An utterance is what you want CHATR to synthesize as speech. It is the input of CHATR. Several ways of doing this input exist, giving more or less prosodic specifications to CHATR. Here are the most commonly used.

1.2.1 The Text Type

This is the simplest way of giving input to CHATR. It is just plain text, without any information concerning prosody. CHATR will have no external help to predict prosody for the text :

```
(Utterance Text
 "You can pay for the hotel with a credit card.")
```

1.2.2 The PhonoWord Type

This kind of input enables to specify prosodic phrases as well as intonation features. It is rather easy to use. There are four phrase levels (Discourse, Sentence, Clause and Phrase). Usually the Discourse and Sentence phrase levels are used only once in a given input (because you usually only give CHATR one sentence to synthesize). The Clause and the Phrase phrase levels can be used to insert breaks (a small one with the Phrase phrase level). The following example shows a PhonoWord Utterance labelled with the ToBI intonation method :

```
(Utterance
 PhonoWord
 (:D ()
     (:S ()
         (:C ()
             (you (H*))
             (can)
             (pay))
         (:C ()
             (for)
             (the)
             (hotel (H*)))
         (:C ()
             (with)
             (a)
             (credit)
             (card (H*) (L-L%))))))
```

The second C phrase level will make CHATR have a break in saying this sentence. At the time I was writing this document this was not really convincing and had to be inproved. The H* enables to emphasis the important words of a sentence (usually the content words). The L-L% is the typical accent of the end of a declarative sentence.

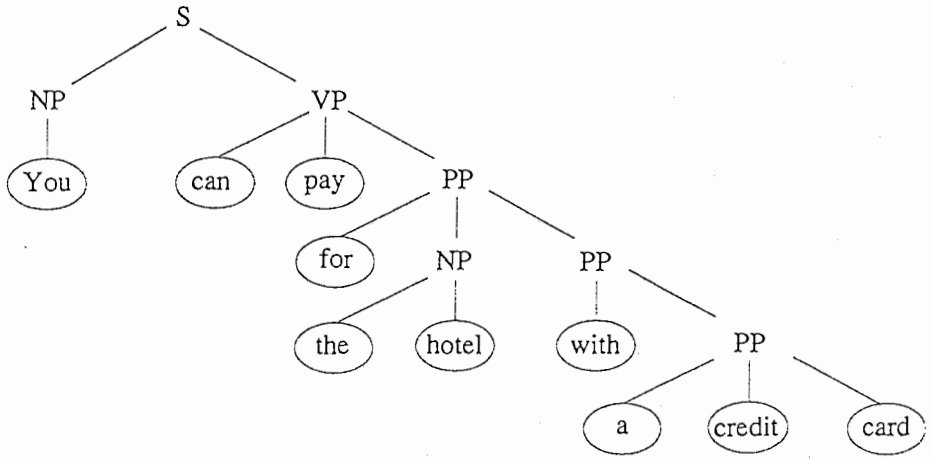
1.2.3 The HLP Type

An HLP (High Level Phrasing) input gives a lot of informations to CHATR. These informations (concerning semantics, syntax, speech act type and focus) will enable CHATR to predict pauses and accents. It is possible to indicate the category (CAT) of a word (a Noun, a Verb, an Adjective ...) and of a phrase (a Noun Phrase NP, a Verb Phrase VP or a Prepositional Phrase PP). Moreover the type of accent for a given word can be specified (a deaccent, an accent, a strong accent... by using the labels "Focus -", "Focus +", "Focus ++"...). Here is an example of an HLP input :

```
(Utterance
 HLP
 (((CAT S) (IFT Statement))
  (((CAT NP) (LEX you) (Focus ++)))
  (((CAT VP))
   (((CAT Aux) (LEX can)))
   (((CAT Verb) (LEX pay)))
   (((CAT PP))
    (((CAT Prep) (LEX for)))
    (((CAT NP))
     (((CAT Det) (LEX the)))
     (((CAT Noun) (LEX hotel) (Focus ++)))))
  (((CAT PP))
   (((CAT Prep) (LEX with)))
   (((CAT NP))
    (((CAT Det) (LEX a)))
```

```
(((CAT Adj) (LEX credit) (Focus ++)))
(((CAT Noun) (LEX card)))))))))
```

S means Sentence. IFT means Illocutionary Force Type; it identifies the nature of a sentence (Statement, Question, Interjection). An HLP utterance is a tree where nodes are phrases (NP, VP, PP) and leaves are words :

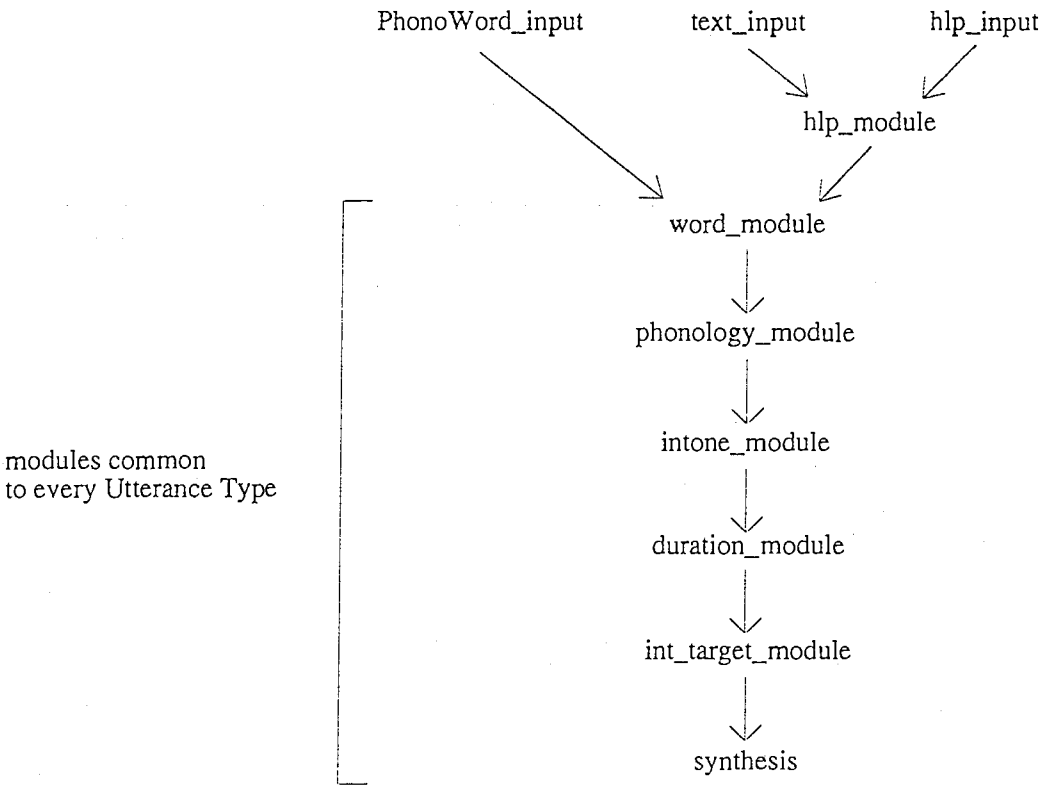


Unlike the PhonoWord Utterance type, the HLP Utterance type is a bit complex, but far more detailed too. The part of code matching with it is also very developed.

Now this refreshing about STREAMS and UTTERANCE TYPES has been done, it is time to see how CHATR is dealing with prosody through its different modules.

1.3 The Alignment of intones to syllables

Independenly of the Utterance Type which is chosen, CHATR will run a certain number of modules (6 to be precise). Before this, CHATR will run one or two modules, depending on to the Utterance Type. The figure below illustrates all this for the three Utterance Types discussed in the previous section :



(an arrow indicates the path followed by the utterance, according to its type)

As a text input will be converted in a (simple) hlp input, let's start to see first in details the PhonoWord_input module.

1.3.1 The PhonoWord_input function

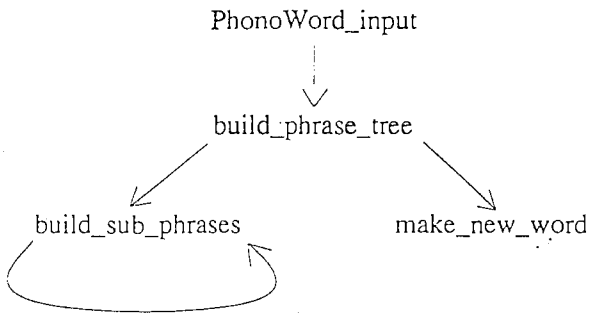
Location

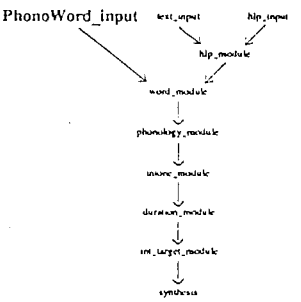
The PhonoWord_input function can be found in :

`~chatr/src/input/pw_input.c`

Overview

The arrows indicate a called function.





Analysis

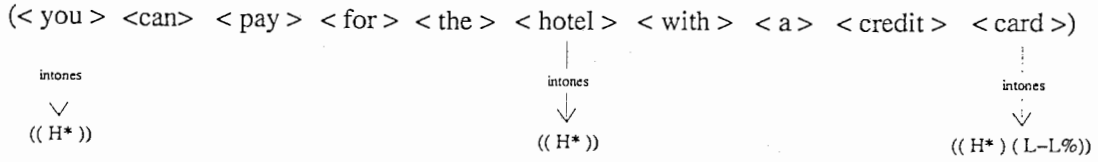
This function is going to create two streams :

- the PphraseStream (P stands for Prosodic)
- the WordStream.

This document will not discuss about the PphraseStream (not so important in a first approach).
Everytime the *build_phrase_tree* function encounters a word in the given utterance, it calls the *make_new_word* function. This function adds the new word (its intones and features too if they exist) to the WordStream.
Thus, for the PhonoWord input :

```
(Utterance
 PhonoWord
 (:D ()
     (:S ()
         (:C ()
             (you (H*))
             (can)
             (pay))
         (:C ()
             (for)
             (the)
             (hotel (H*)))
         (:C ()
             (with)
             (a)
             (credit)
             (card (H*) (L-L%))))))),
```

the execution of the *PhonoWord_input* function will create the following WordStream :



The informations contained in the intones field of each word cell will be used later to build the IntoneStream.

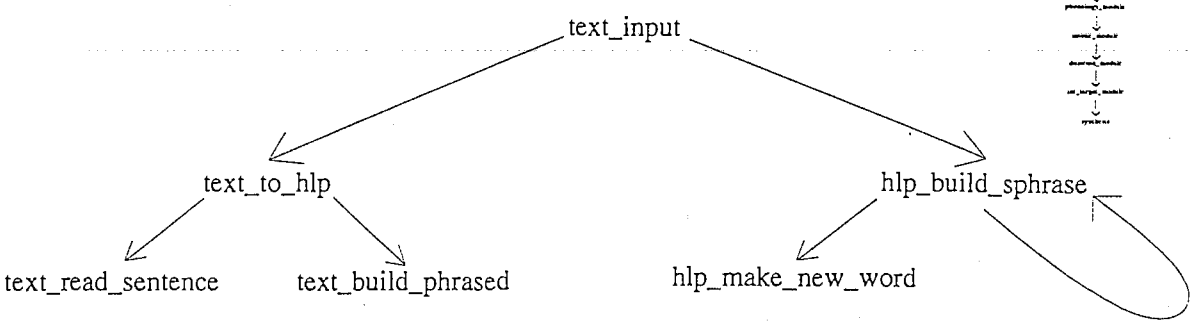
1.3.2 The text_input function

Location

The text_input function can be found in :

```
~chatr/src/input/hlp_input.c
```

Overview



(some of these functions are in :

`~chatr/src/text/text.c`)

Analysis

This function creates the SphraseStream (S stands for Syntactic). As for the PphraseStream in the previous case, this document will not discuss about this new stream.
Mainly this module does two things.
First it transforms the text input into an HLP input :

- the *text_read_sentence* function reads the text input sentence by sentence,
- the *text_build_phrased* function builds (LEX "word") cells for each sentence and adds an IFT type to the sentence.

Thus for the Text input:

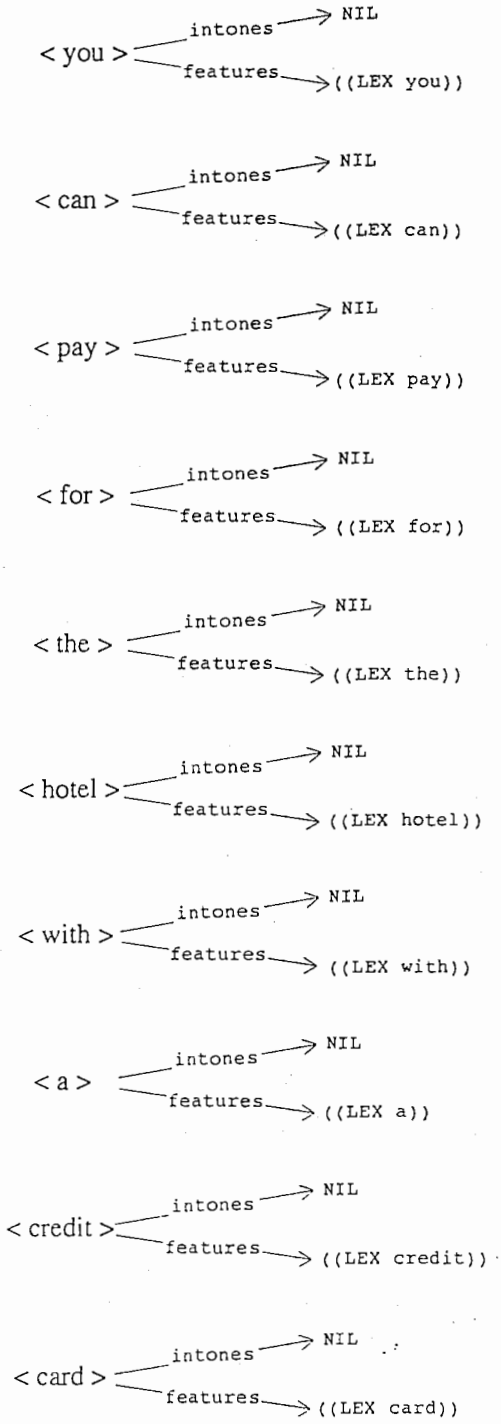
```
(Utterance Text
 "You can pay for the hotel with a credit card."),
```

the execution of the *text_input* function will create the following HLP input :

```
(Utterance HLP
  (((CAT D))
   (((CAT S) (IFT Statement))
    (((LEX You)))
    (((LEX can)))
    (((LEX pay)))
    (((LEX for)))
    (((LEX the)))
    (((LEX hotel)))
    (((LEX with)))
    (((LEX a)))
```

```
(((LEX credit)))
(((LEX card))))))
```

And secondly it creates the WordStream (by the *hlp_make_word* function; the *hlp_build_sphrase* function builds the SphraseStream). In the WordStream, the "intones" fields of each word are set to NIL. The "features" fields are filled in most cases with only a (LEX "word") cell. Thus for the same text input as above, the execution of the *text_input* function will create the following WordStream :



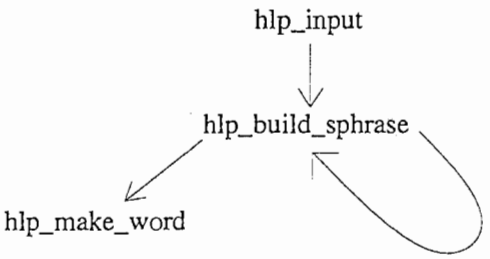
1.3.3 The hlp_input function

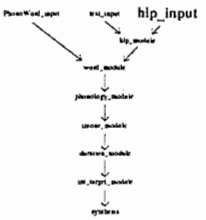
Location

The hlp_input function can be found in :

```
~chatr/src/input/hlp_input.c
```

Overview





Analysis

This function is going to create the SphraseStream (by the *hlp_build_sphrase* function) and the WordStream (by the *hlp_make_word* function). In the WordStream, the "intones" field of each word is set to NIL. The "features" field is filled with all the syntactic and prosodic informations of the current word. Thus for the HLP input:

```
(Utterance HLP
 (((CAT S) (IFT Statement))
  (((CAT NP) (LEX you) (Focus ++)))
  (((CAT VP))
   (((CAT Aux) (LEX can)))
   (((CAT Verb) (LEX pay)))
   (((CAT PP))
    (((CAT Prep) (LEX for)))
    (((CAT NP))
     (((CAT Det) (LEX the)))
     (((CAT Noun) (LEX hotel) (Focus ++)))))
  (((CAT PP))
   (((CAT Prep) (LEX with)))
   (((CAT NP))
    (((CAT Det) (LEX a)))
    (((CAT Adj) (LEX credit) (Focus ++)))
    (((CAT Noun) (LEX card)))))))),
```

the execution of the *hlp_input* function will create the following WordStream :

< you > —intones→ NIL
< you > —features→ ((CAT NP) (LEX you) (Focus ++))

< can > —intones→ NIL
< can > —features→ ((CAT Aux) (LEX can))

< pay > —intones→ NIL
< pay > —features→ ((CAT Verb) (LEX pay))

< for > —intones→ NIL
< for > —features→ ((CAT Prep) (LEX for))

< the > —intones→ NIL
< the > —features→ ((CAT Det) (LEX the))

< hotel > —intones→ NIL
< hotel > —features→ ((CAT Noun) (LEX hotel) (Focus ++))

< with > —intones→ NIL
< with > —features→ ((CAT Prep) (LEX with))

< a > —intones→ NIL
< a > —features→ ((CAT Det) (LEX a))

< credit > —intones→ NIL
< credit > —features→ ((CAT Adj) (LEX credit))

< card > —intones→ NIL
< card > —features→ ((CAT Noun) (LEX card) (Focus ++))

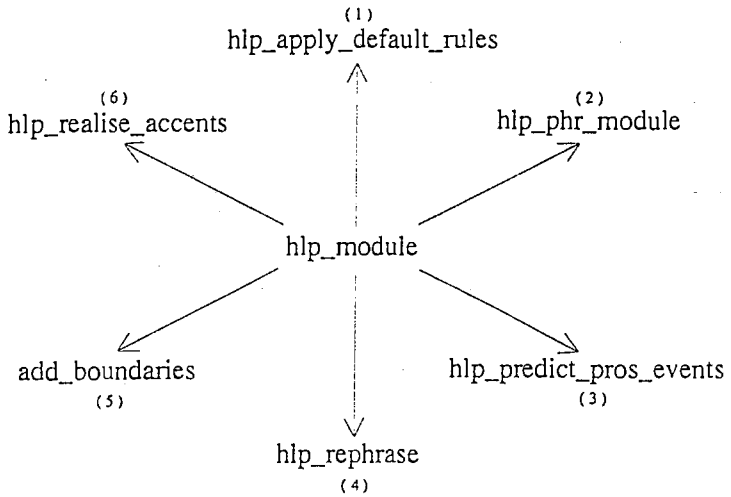
1.3.4 The hlp_module function

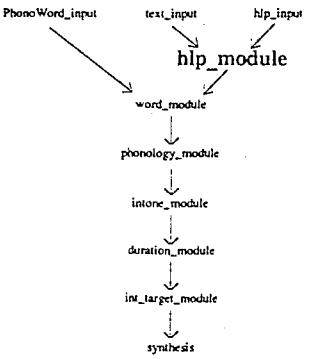
Location

The *hlp_module* function can be found in :

```
~chatr/src/input/hlp.c
```

Overview



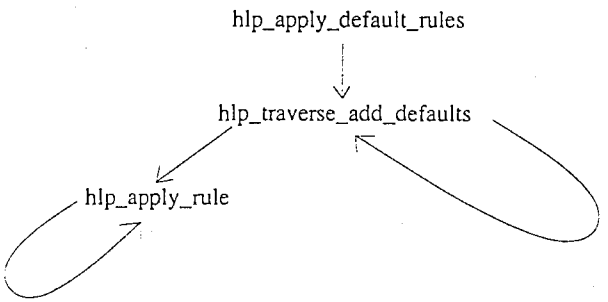


(The numbers indicate the order of execution)

Analysis

The part of code concerning the *hlp_module function* is really huge. The *hlp_module* function calls 6 major functions. Let's have a look at them one by one.

The hlp_apply_default_rules function

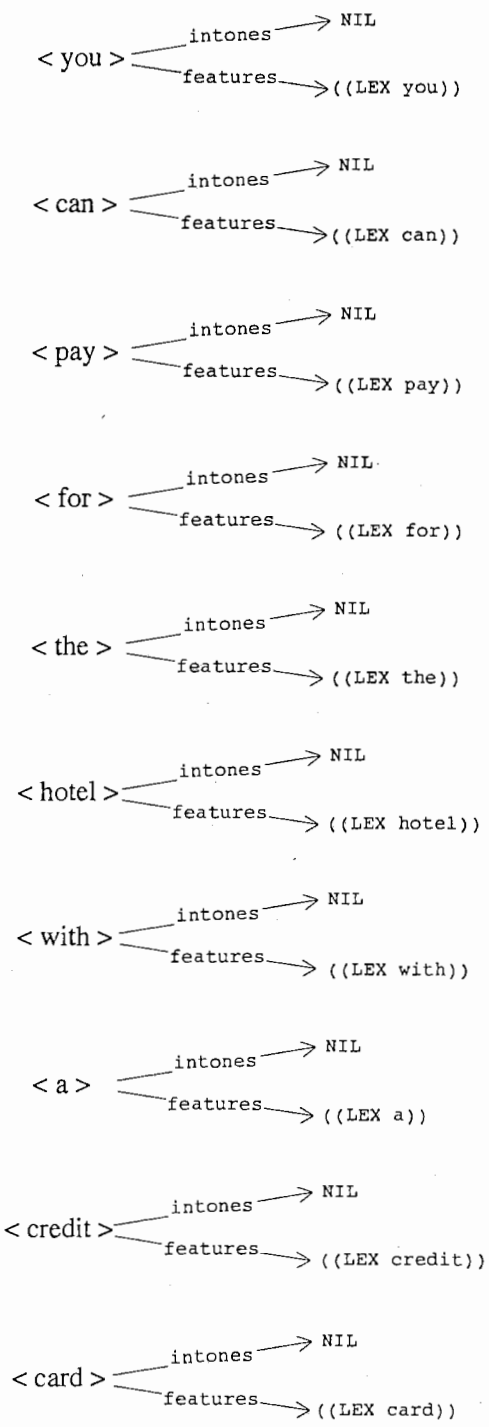


This function recurses through the given tree input (remember an HLP input can be seen as a tree) and tries to apply the user defined rules. An example of such rules (contained in the HLP_Rules variable) is :

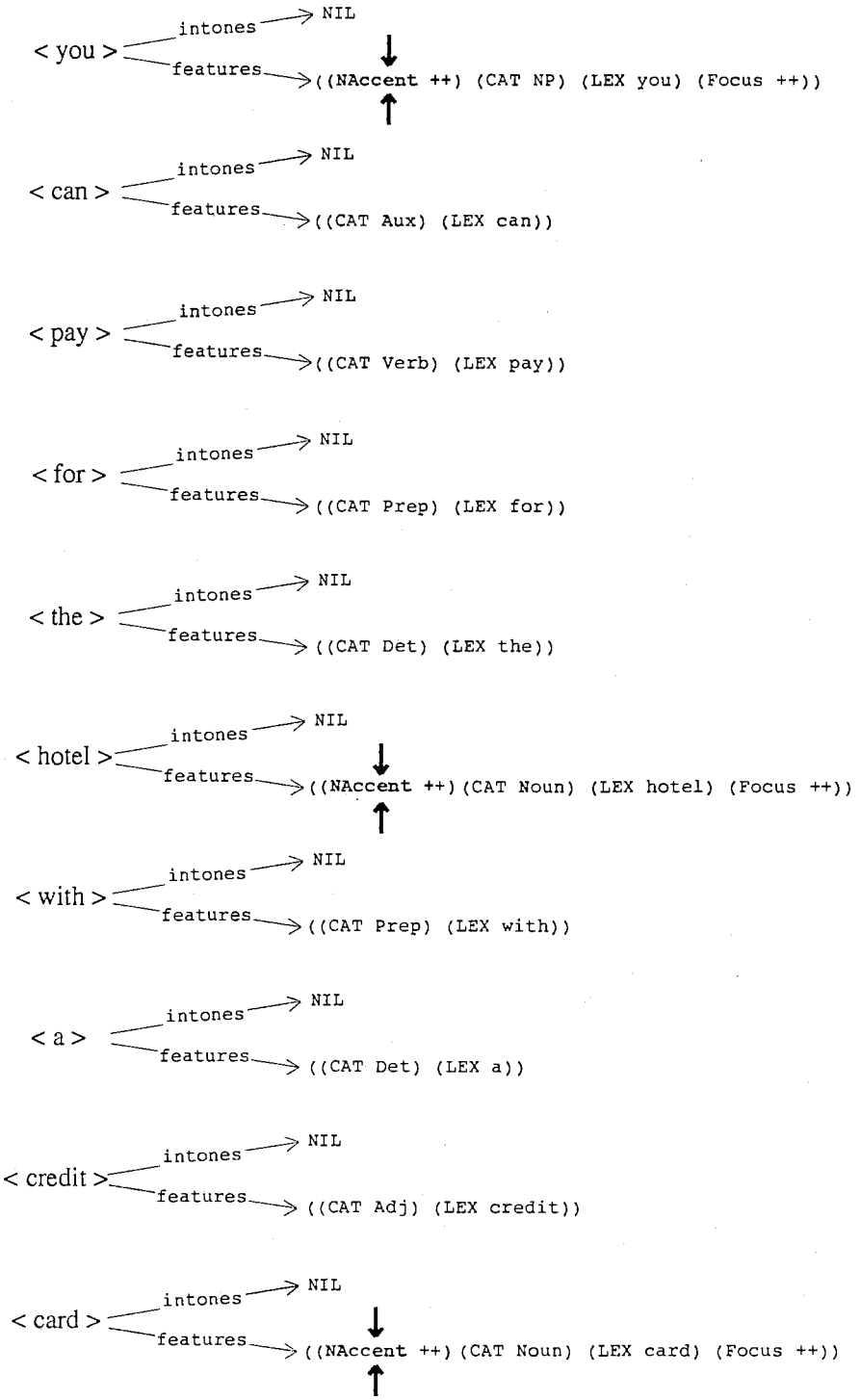
```
( ( ((Focus +)) => ((NAccent +)) )
  ( ((Focus ++)) => ((NAccent ++)) )
  ( ((Contrastive +)) => ((NAccent ++)) )
  ( ((Focus -)) => ((NAccent -)) )
  ( ((CAT S)) => ((PhraseLevel :S)) ) )
```

The *hlp_apply_default_rules* function looks at every element contained in the "features" fields. If an element matches with an expression belonging to the left side of the

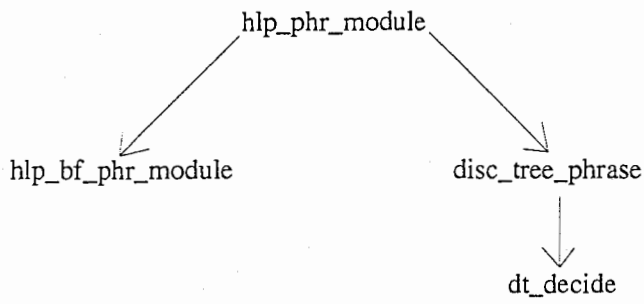
previous rules, it adds the expression of the right side to the "features" field (by the *hlp_apply_rule* function).
In the Text Input case, as it is plain text, this function does not affect the WordStream :



In the HLP Input case, after the execution of the *hlp_apply_default_rules* function the WordStream becomes :


< you >
intones
NIL
features
((NAccent ++) (CAT NP) (LEX you) (Focus ++))
< can >
intones
NIL
features
((CAT Aux) (LEX can))
< pay >
intones
NIL
features
((CAT Verb) (LEX pay))
< for >
intones
NIL
features
((CAT Prep) (LEX for))
< the >
intones
NIL
features
((CAT Det) (LEX the))
< hotel >
intones
NIL
features
((NAccent ++)(CAT Noun) (LEX hotel) (Focus ++))
< with >
intones
NIL
features
((CAT Prep) (LEX with))
< a >
intones
NIL
features
((CAT Det) (LEX a))
< credit >
intones
NIL
features
((CAT Adj) (LEX credit))
< card >
intones
NIL
features
((NAccent ++)(CAT Noun) (LEX card) (Focus ++))

The hlp_phr_module function



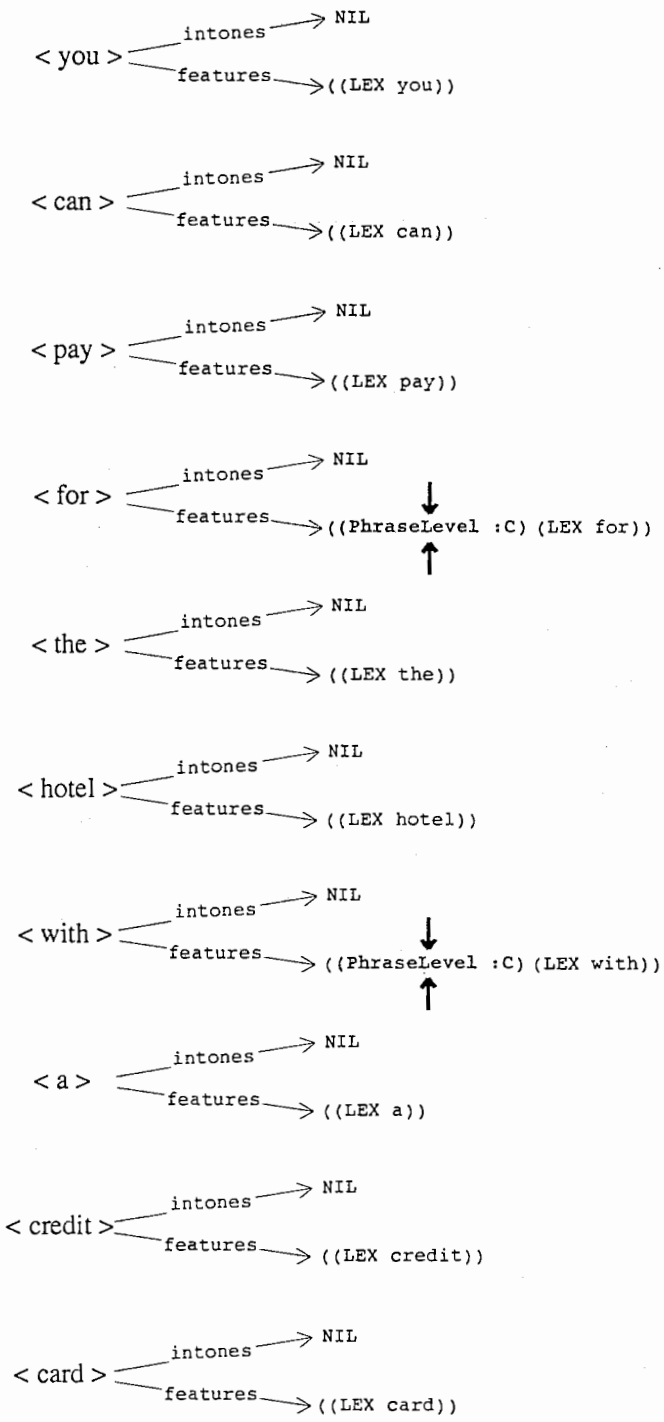
This function predicts phrasing by the method defined by the user. Two default methods exist :

- the Bachenko_Fitzpatrick method
- the DiscTree method.

Given that the DiscTree method is the one currently used by Chatr, only this method will be discussed here.

The DiscTree method takes each word one by one and for each of them, it determines its break index. The break index indicates how strongly the current word and the previous one are linked. Its values can be 1, 2, 3 or 4. A break index 1 means that the two words are closely linked (like "the" and "hotel" in the example of this document) whereas a break index 4 means that the two words are very disjoint (usually the last word of a sentence and the first one of the following sentence).
The *dt_decide* function returns for each word its break index. It uses a decison tree, which looks at the type of the previous and of the following words to decide the break index. However the returned break index has its value equal to 1 or 4 (so in that case 4 is no more for only marking the end of a sentence; it also marks possible pauses in the sentence).

If a break index 4 is returned, the *disc_tree_phrase* function adds the "(PhraseLevel :C)" feature to the word. Thus after the execution of the *hlp_phr_module* function the WordStream, in the Text Input case, becomes :



And in the HLP Input case it becomes :

< you > —intones→ NIL
< you > —features→ ((NAccent ++) (CAT NP) (LEX you) (Focus ++))

< can > —intones→ NIL
< can > —features→ ((CAT Aux) (LEX can))

< pay > —intones→ NIL
< pay > —features→ ((CAT Verb) (LEX pay))

< for > —intones→ NIL
< for > —features→ ((PhraseLevel :C) (CAT Prep) (LEX for))

< the > —intones→ NIL
< the > —features→ ((CAT Det) (LEX the))

< hotel > —intones→ NIL
< hotel > —features→ ((NAccent ++) (CAT Noun) (LEX hotel) (Focus ++))

< with > —intones→ NIL
< with > —features→ ((PhraseLevel :C) (CAT Prep) (LEX with))

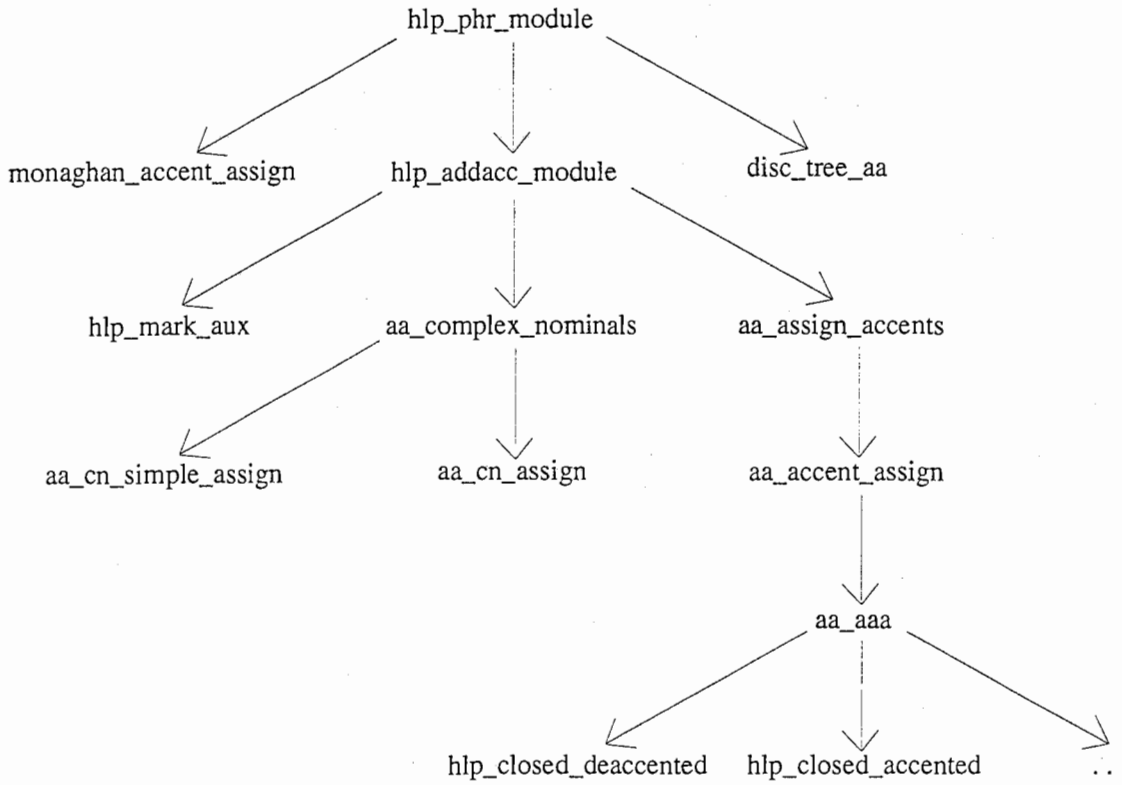
< a > —intones→ NIL
< a > —features→ ((CAT Det) (LEX a))

< credit > —intones→ NIL
< credit > —features→ ((CAT Adj) (LEX credit))

< card > —intones→ NIL
< card > —features→ ((NAccent ++) (CAT Noun) (LEX card) (Focus ++))

(the modifications are the same for both input ways)

The hlp_predict_pros_events function



(the *hlp_addacc_module* function can be found in :

`~chatr/src/input/hlp_addacc.c`)

The *hlp_predict_pros_events* function decides on the prosodic prediction strategy to use and applies it. Three strategies currently exist :

- the Hirschberg strategy,
- the Monaghan strategy and
- the DiscTree strategy.

The one used by default by Chatr is the Hirschberg strategy. The use of this strategy makes the *hlp_predict_pros_events* function calls the *hlp_addacc_module* function. This function does three things :

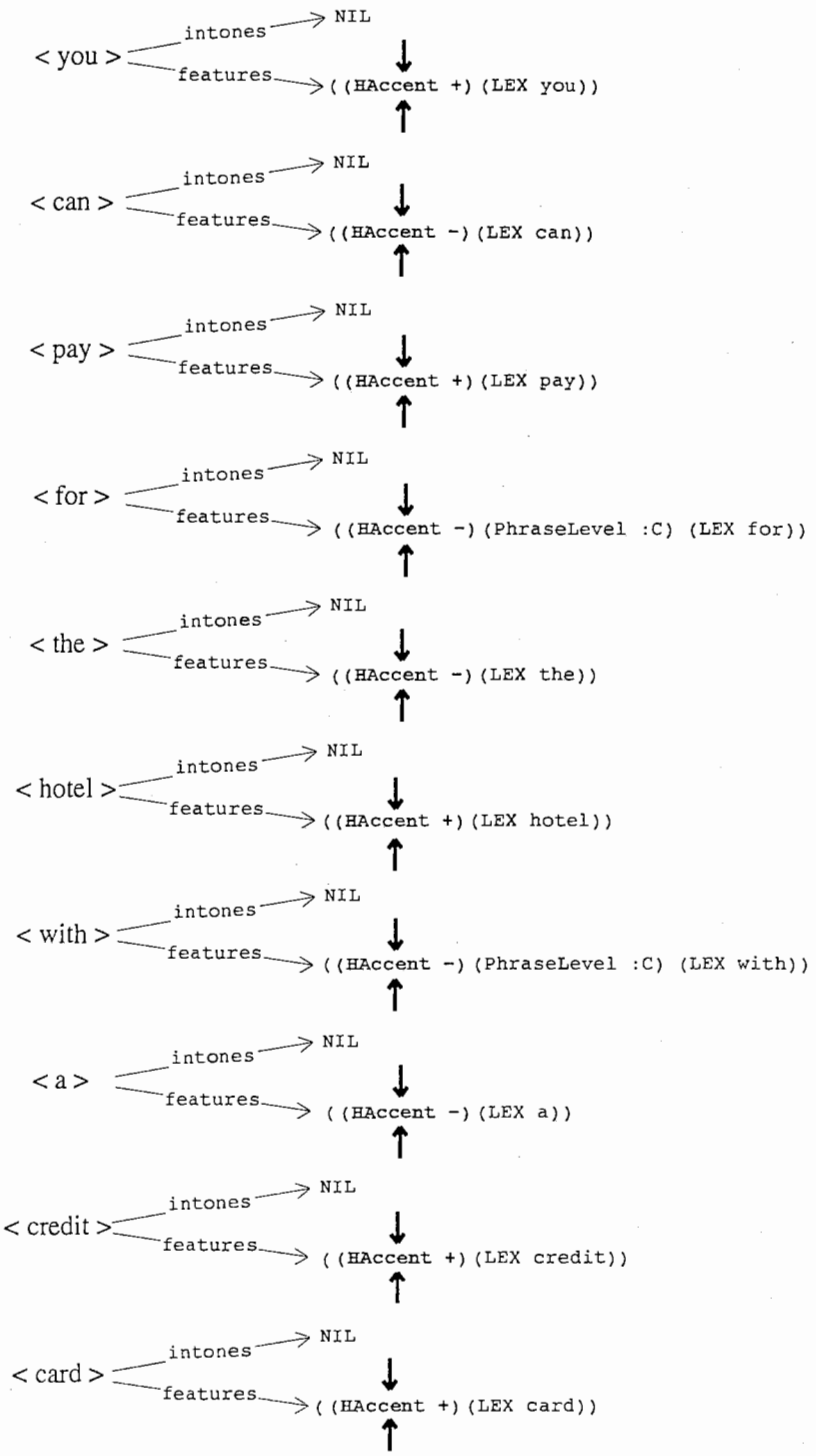
- it looks for the auxiliary verbs (by the *hlp_mark_aux* function)

Every time an auxiliary verb is detected, a "(CAT Aux)" feature is added, while the "(CAT Verb)" is deleted (if it exists). Thus in our example, if we had "(CAT Verb) (LEX can)", the *hlp_mark_aux* function would replace it by "(CAT Aux) (LEX can)".

- it assigns stress within complex nominals (*aa_complex_nominals*)
A complex nominal is a set of nouns and adjectives matching together to form a single concept, like for example "credit card". For each word of a complex nominals the *aa_complex_nominals* function (the *aa_cn_assign* function to be more precise) decides which ones have to be stressed (then a "(CN Stress)" feature is added) and which ones have to be unstressed (then a "(CN Unstress)" feature is added).

- it adds the correct type of accent to each word (*aa_assign_accents*)
According to all the features that existed and that have been added since the beginning, the *aa_assign_accents* function (the *aa_aaa* function in fact) decides to add the features "(HAccent +)", "(HAccent -)", "(HAccent ++)", "(HAccent c)" or no feature (if a HAccent feature or a NAccent feature already exists). It is from these features that the IntoneStream will be built later.

Thus after the execution of the *hlp_predict_pros_events* function the WordStream in the Text Input case becomes :

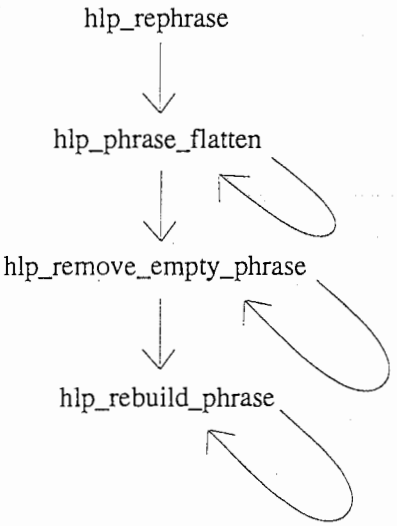


and in the HLP Input case :

```
< you >  --intones--> NIL
         --features--> ((NAccent ++) (CAT NP) (LEX you) (Focus ++))

< can >  --intones--> NIL
         --features--> ((HAccent -) (CAT Aux) (LEX can))

< pay >  --intones--> NIL
         --features--> ((HAccent +) (CAT Verb) (LEX pay))

< for >  --intones--> NIL
         --features--> ((HAccent -) (PhraseLevel :C) (CAT Prep) (LEX for))

< the >  --intones--> NIL
         --features--> ((HAccent -) (CAT Det) (LEX the))

< hotel > --intones--> NIL
          --features--> ((NAccent ++) (CAT Noun) (LEX hotel) (Focus ++))

< with > --intones--> NIL
         --features--> ((HAccent -) (PhraseLevel :C) (CAT Prep) (LEX with))

< a >    --intones--> NIL
         --features--> ((HAccent -) (CAT Det) (LEX a))

< credit > --intones--> NIL
           --features--> ((HAccent -) (CN Unstress)(CAT Adj) (LEX credit))

< card > --intones--> NIL
         --features--> ((CN Stress) (NAccent ++) (CAT Noun) (LEX card) (Focus ++))
```

Because of all the information given in the HLP Input case, the resulting WordStream gets more accurate features than in the simple Text Input case.
The *aa_assign_accents* function is not achieved yet. Moreover for some features (like "Vphr", "Proposed", "Prefixed"...) it is difficult to know their exact use. No exhaustive and explicative list of features has yet been prepared.

The hlp_rephrase function



Basically the *hlp_rephrase* function

- deletes the nodes of the HLP tree (like "(CAT NP)", "(CAT VP)" and "(CAT PP)"),
- extracts the "(PhraseLevel :...)" features,
- makes new nodes from them.

So for the following SphraseStream (the SphraseStream has not been detailed in this document; but it's not necessary to know the details, except that the 'S' refers to 'Syntax') :

```
(((PitchRange two) (Start 0.0) (PhraseLevel :S) (CAT S) (IFT Statement))
 (((NAccent ++) (CAT NP) (LEX you) (Focus ++)))
 (((CAT VP))
  (((HAccent -) (CAT Aux) (LEX can)))
  (((HAccent +) (CAT V) (LEX pay)))
  (((CAT PP))
   (((HAccent -) (PhraseLevel :C) (CAT Prep) (LEX for)))
   (((CAT NP))
    (((HAccent -) (CAT Det) (LEX the)))
    (((NAccent ++) (CAT Noun) (LEX hotel) (Focus ++)))))
  (((CAT PP))
   (((HAccent -) (PhraseLevel :C) (CAT Prep) (LEX with)))
   (((CAT NP))
    (((HAccent -) (CAT Det) (LEX a)))
    (((HAccent -) (CN Unstress) (CAT Adj) (LEX credit)))
    (((CN Stress) (NAccent ++) (CAT Noun) (LEX card) (Focus ++)))))))),
```

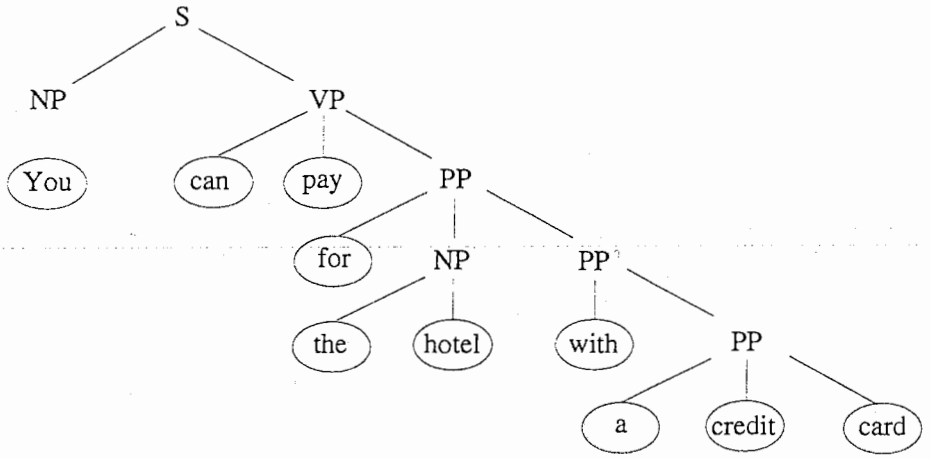
the *hlp_phrase_flatten* function puts every leaves of the tree at the same level (nodes are erased because they are not useful anymore):

```
(((PitchRange two) (Start 0.0) (PhraseLevel :S) (CAT S) (IFT Statement))
 ((NAccent ++) (CAT NP) (LEX you) (Focus ++))
 ((HAccent -) (CAT Aux) (LEX can))
 ((HAccent +) (CAT V) (LEX pay))
 ((PhraseLevel :C))
 ((HAccent -) (CAT Prep) (LEX for))
 ((HAccent -) (CAT Det) (LEX the))
 ((NAccent ++) (CAT Noun) (LEX hotel) (Focus ++))
 ((PhraseLevel :C))
 ((HAccent -) (CAT Prep) (LEX with))
 ((HAccent -) (CAT Det) (LEX a))
 ((HAccent -) (CN Unstress) (CAT Adj) (LEX credit))
 ((CN Stress) (NAccent ++) (CAT Noun) (LEX card) (Focus ++))),
```

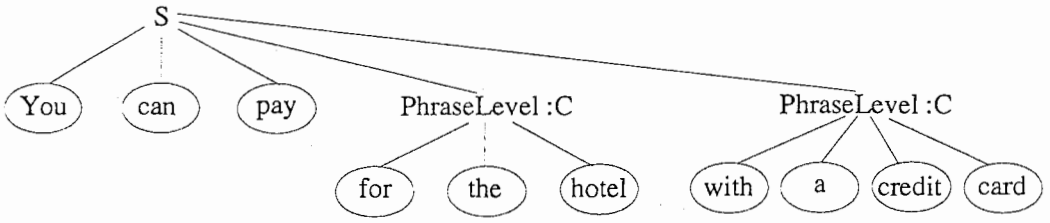
the *hlp_remove_empty_phrase* function cleans it by removing the possible empty phrases (not the case here so nothing is modified) and finally the *hlp_rebuild_phrase* function builds a new HLP tree (SPhraseStream) where the nodes are the "(PhraseLevel :...)" features (moved from where they were before) :

```
((((PitchRange two) (Start 0.0) (PhraseLevel :S) (CAT S) (IFT Statement))
  (((NAccent ++) (CAT NP) (LEX you) (Focus ++)))
  (((HAccent -) (CAT Aux) (LEX can)))
  (((HAccent +) (CAT V) (LEX pay)))
  (((PhraseLevel :C))
   (((HAccent -) (CAT Prep) (LEX for)))
   (((HAccent -) (CAT Det) (LEX the)))
   (((NAccent ++) (CAT Noun) (LEX hotel) (Focus ++))))
  (((PhraseLevel :C))
   (((HAccent -) (CAT Prep) (LEX with)))
   (((HAccent -) (CAT Det) (LEX a)))
   (((HAccent -) (CN Unstress) (CAT Adj) (LEX credit)))
   (((CN Stress) (NAccent ++)  (CAT Noun) (LEX card) (Focus ++))))))
```

The previous HLP tree



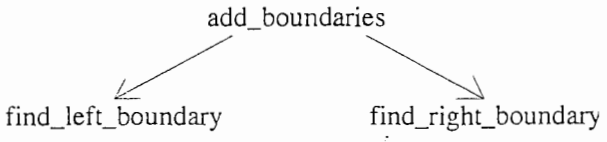
has become



Remark : in the case of the Text Input (a flat HLP tree), the new SPhraseStream is :

```
((((PitchRange two) (Start 0.0) (PhraseLevel :S) (CAT S) (IFT Statement))
  (((HAccent +) (LEX you)))
  (((HAccent -) (LEX can)))
  (((HAccent +) (LEX pay)))
  (((PhraseLevel :C))
   (((HAccent -) (LEX for)))
   (((HAccent -) (LEX the)))
   (((HAccent +) (LEX hotel))))
  (((PhraseLevel :C))
   (((HAccent -) (LEX with)))
   (((HAccent -) (LEX a)))
   (((HAccent +) (LEX credit)))
   (((HAccent +) (LEX card))))))
```

The add_boundaries function



(the *add_boundaries* function can be found in :

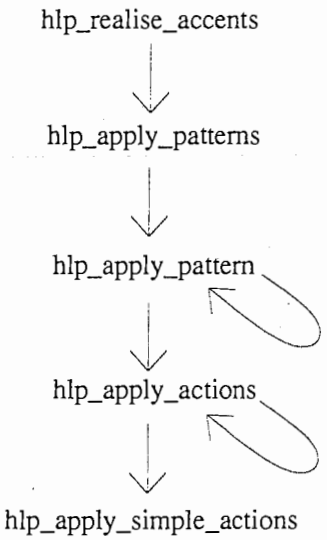
```
~chatr/src/lex/word.c )
```

The *add_boundaries* function determines the left and the right boundaries (0, 1, 2, 3 or 4) of each word. These boundaries match with the break indexes calculated by the *hlp_phr_module* function. However a 1 break index value is converted in a 0 boundary value and a 4 break index value is converted in a 2 boundary value. The left boundary of a word is equal to the right boundary of the previous word (in case of conflict the highest value is chosen). The left boundary of the first word and the right boundary of the last word are set to 4.
Here are the values of the boundaries for each word of the WordStream (they are the same in both Text and HLP Input cases) after the execution of the *add_boundaries* function :

```
4 you 0

0 can 0

0 pay  2

2 for  0

0 the  0

0 hotel  2

2 with  0

0 a  0

0 credit  0

0 card  4
```

(these values are kept in the left_boundary and the right_boundary fields of each word).

The hlp_realise_accents function



The hlp_realise_accents is the first function which is going to affect the "intones" field of the words of the WordStream.
It applies the pattern rules. A example of such rules (contained in the HLP_Patterns variable) is the following :

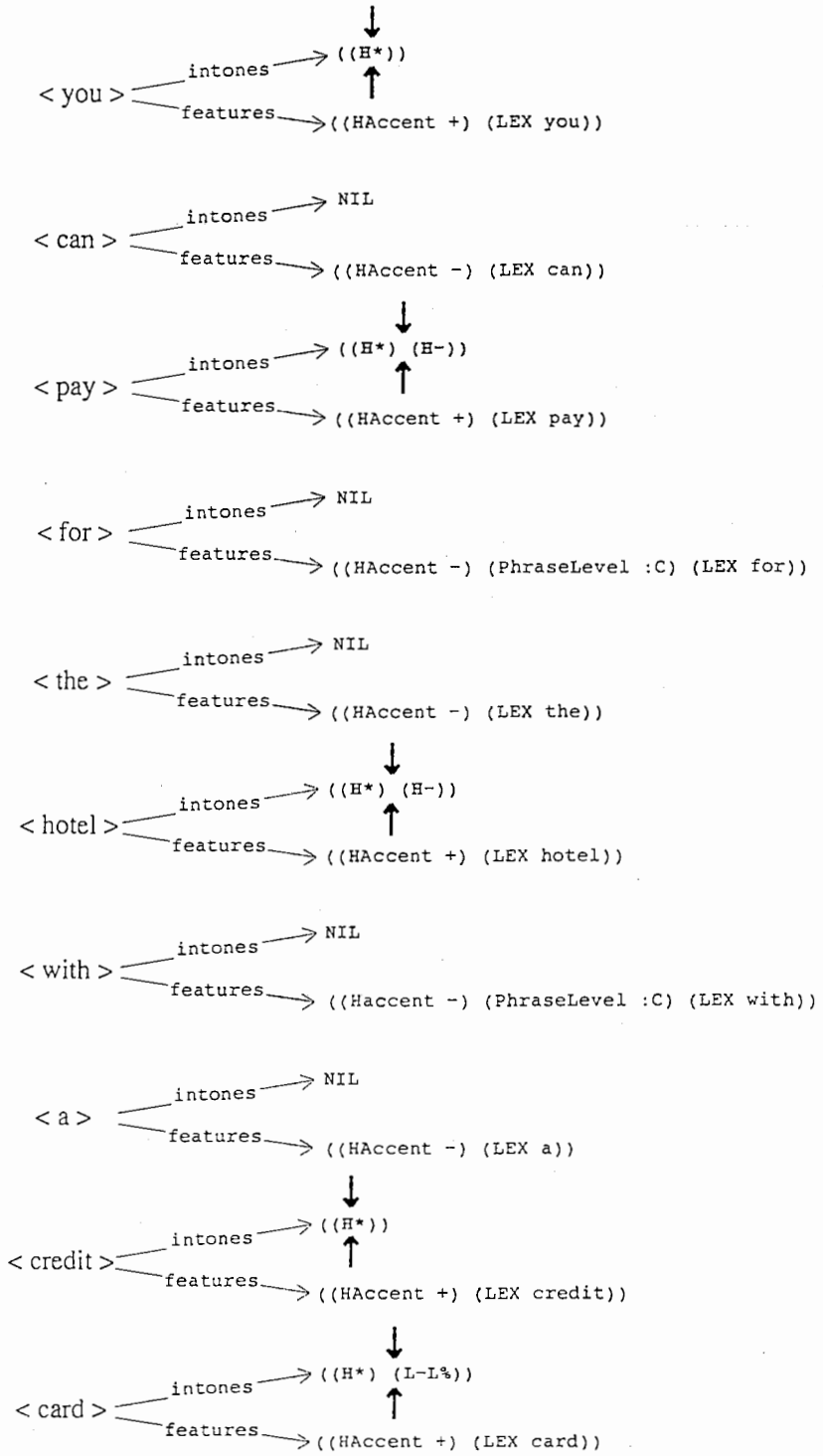
```
(Statement  (START )
            (HAccent (+ (H*))
                     (++ (L+H*)))
            (PHRASE (H-))
            (TAIL (L-L%)))
(YNQuestion (START )
            (HAccent (+ (L*)))
            (TAIL (H-H%)))
(Question   (START )
            (HAccent (+ (L*)))
            (TAIL (L-L%)))
(*          (START)
    (HAccent (+ (H*)))
            (PHRASE (H-))
            (TAIL (H-L%)))
```

Some actions are special (because they concern phrases), like START, PHRASE or TAIL; they are applied by the *hlp_apply_actions* function. Some others are said to be simple (because they concern the words), like HAccent; they are applied by the *hlp_apply_simple_actions*. For instance in the case of a Statement utterance type (as the example of this document), the part of the pattern rules which are going to be used is :

```
(Statement  (START )
            (HAccent (+ (H*))
                     (++ (L+H*)))
```

```
            (PHRASE (H-))
            (TAIL (L-L%)))
)))
```

This means that if a word has an "(HAccent +)" feature, a "(H*)" intone will be added to its "intones" field, if it is the last word of a phrase a "(H-)" intone will be added to its "intones" field, etc... Thus after the execution of the *hlp_realise_accents* function the WordStream, in the Text Input case, becomes :



and in the HLP Input case :

```
< you >  --intones--> NIL
         --features--> ((NAccent ++) (CAT NP) (LEX you) (Focus ++))

< can >  --intones--> NIL
         --features--> ((HAccent -) (CAT Aux) (LEX can))

< pay >  --intones--> ((H*) (H-))
         --features--> ((HAccent +) (CAT Verb) (LEX pay))

< for >  --intones--> NIL
         --features--> ((HAccent -) (PhraseLevel :C) (CAT Prep) (LEX for))

< the >  --intones--> NIL
         --features--> ((HAccent -) (CAT Det) (LEX the))

< hotel > --intones--> ((H-))
          --features--> ((NAccent ++) (CAT Noun) (LEX hotel) (Focus ++))

< with > --intones--> NIL
         --features--> ((Haccent -) (PhraseLevel :C) (CAT Prep) (LEX with))

< a >    --intones--> NIL
         --features--> ((HAccent -) (CAT Det) (LEX a))

< credit > --intones--> NIL
           --features--> ((HAccent -) (CN Unstress) (CAT Adj) (LEX credit))

< card > --intones--> ((L-L%))
         --features--> ((CN Stress) (NAccent ++) (CAT Noun) (LEX card) (Focus ++))
```

As the *hlp_realise_accent* function is the last one called by the *hlp_module* function, these two WordStreams are also the resulting WordStreams of the execution of the *hlp_module* function.

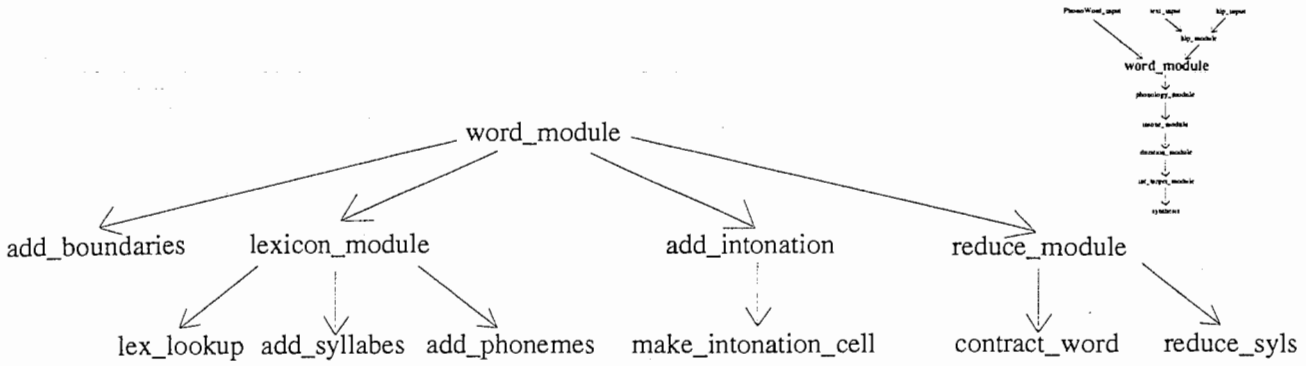
1.3.5 The word_module function

Location

The word_module function can be found in :

`~chatr/src/lex/word.c`

Overview



(The *add_intonation* function can be found in :

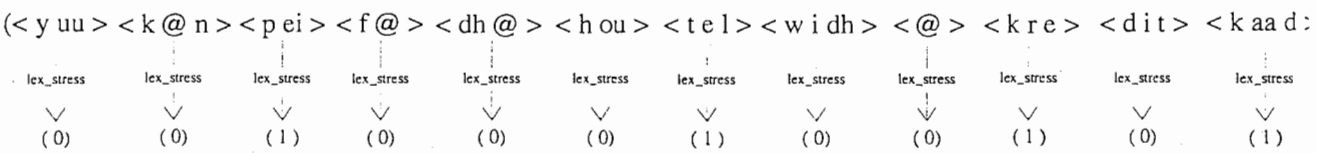
`~chatr/src/intonation/intonation.c`

The *reduce_module* function can be found in :

`~chatr/src/lex/reduce.c`)

Analysis

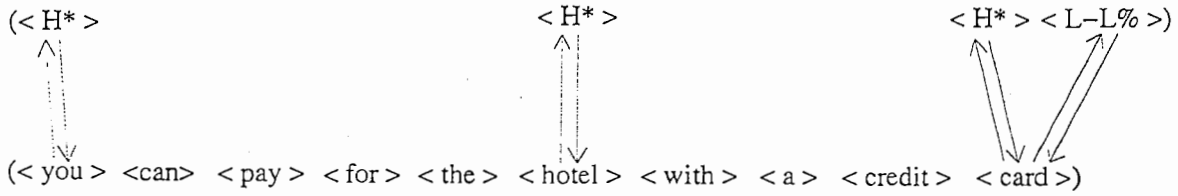
The *word_module* function creates the IntoneStream according to the "intones" field of each word of the WordStream. It also creates the SylStream (and the PhoneStream). All the newly created streams are linked to the WordStream (this is explained at the beginning of this document).
The *add_boundaries* function gives the left and right boundaries for each word of the WordStream. In the case of an HLP input this function has already been called once by the *hlp_module* function. So see the previous subsection for details.
The *lexicon_module* function looks up the entry of each word of the WordStream in the lexicon. The lexicon contains the words that CHATR can utter and their decomposition into syllables and phonemes. From this information the function builds the SylStream and the PhoneStream. For instance the SylStream associated to the example presented above :



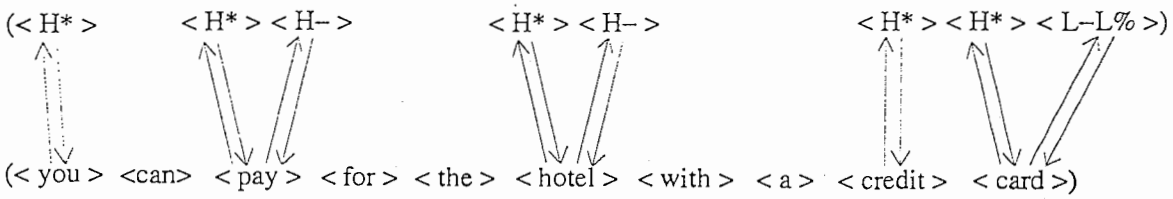
(a (1) means stressed and a (0) unstressed)

The *add_intonation* function builds the IntoneStream. It uses the information contained in the "intones" field of each word. According to the input method (Text, PhonoWord or HLP) the Intone fields have been filled from different ways. So for the

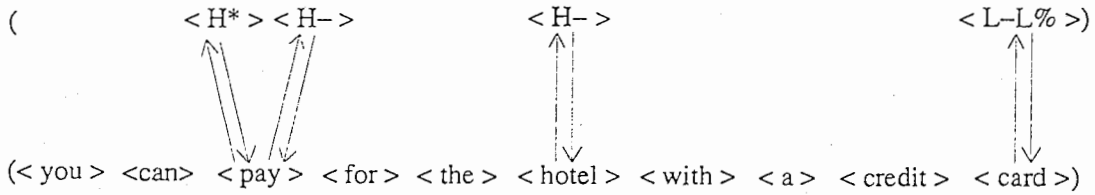
three input ways used in this document, we have three three different IntoneStreams. For the PhonoWord Input way, the IntoneStream, linked to the WordStream, is :



For the Text Input way :



And finally for the HLP Input way :

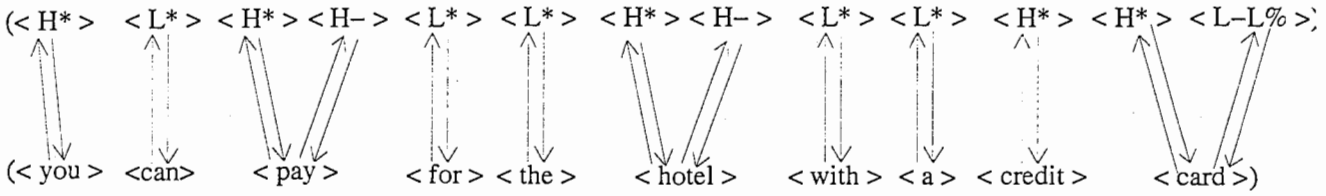


The IntoneStream relative to the HLP Input may seem poor compared to the size devoted to the *HLP_module* function. In fact this is due to the HLP_Patterns variable. In the one of this document no mapping for (NAccent ++) and (HAccent -) features exist. If the HLP_Patterns variable had been :

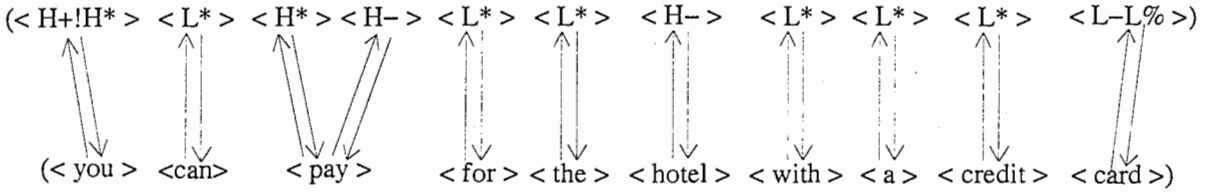
```
(Statement   (START )
             (HAccent (+ (H*))
                      (++ (L+H*))
                      (- (L*)))
             (NAccent (++ (H+!H*)))
             (PHRASE (H-))
             (TAIL (L-L%)))
(YNQuestion  (START )
             (HAccent (+ (L*)))
             (TAIL (H-H%)))
(Question    (START )
             (HAccent (+ (L*)))
             (TAIL (L-L%)))
```

```
(*          (START)
    (HAccent (+ (H*)))
            (PHRASE (H-))
            (TAIL (H-L%)))))
```

, the resulting IntoneStream would have been, for the Text Input way :



and for the HLP Input way :



In the case of the HLP Input, although "hotel" and "card" are "(Focus ++)" marked like "you", there is no (H+!H*) Intone cell matching with these words. This is because for one sentence CHATR only accepts one "(Focus ++)" marked word; the following "(Focus ++)" features are ignored. But this part of code could be easily changed (in the *hlp_realise_accent* function).
Compared to the HLP Input way, the Text Input way makes CHATR add too many (H*) intones (usually on every noun, pronoun and verb). Therefore important words are hidden among not important ones. With an HLP Input the user can mark these important words by using a (Focus ++) label. If (Focus ++) matches with a (H+!H*) accent, these words will sound differently.
Compared to the HLP Input way, the PhonoWord Input way asks the user to give all the accents. CHATR will not add new ones. If it is quick to add accents like (H*) or (H+!H*) on important words (usually not in a great number), it may become rapidly boring to add accents like (L*) on not important words (usually numerous).
As the IntoneStream is concerned, the HLP Input way is the best one. It automatically finds the accents for each word and lets the possibility to the user to indicate the words that need to be focused on for example. The only drawback is that an HLP Input is quite long to write.

The *reduce_module* function enables to reduce some words under certain conditions. If a word is in the list of the reducable words (contained in the "contract_words" variable), has no intone and has its right and left boundaries equal to 0, it will be removed and the phoneme it matches with in the "contract_words" variable will be added to the previous word. As this module is switched to off at the time I am writing this report I will not go into details.

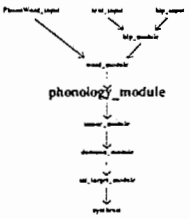
1.3.6 The phonology_module function

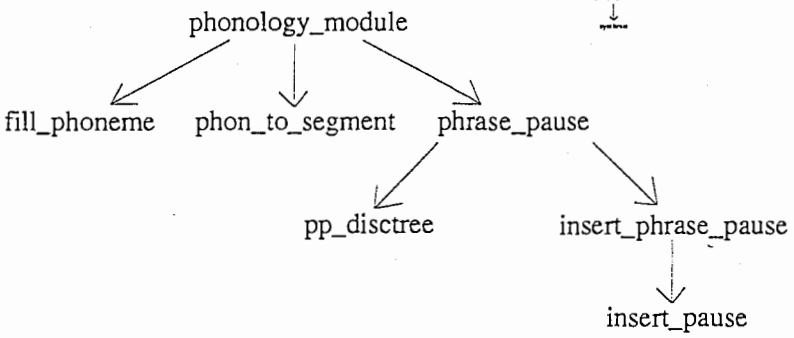
Location

The *phonology_module* function can be found in :

```
~chatr/src/phoneme/phonology.c
```

Overview





(the *insert_phrase_pause* and the *insert_pause* functions can be found in :

```
~chatr/src/intonation/phrase\_int.c )
```

Analysis

The *phonology_module* function does not affect the WordStream, nor the SylStream and nor the IntoneStream.
The *fill_phoneme* function fills the features of every phoneme of the PhoneStream and the *phon_to_segment* function builds the SegStream.
The *phrase_pause* function inserts silence segments when needed; if the chosen pause prediction method (its value is contained in the "pause_prediction_method" variable) is the disctree one, silence segments are inserted after stop, comma, question mark and colon (if the following phrase contains at least one stressed syllable). If the pause prediction method is by phrase break, a silence segment is inserted every time a phrase break (like "(PhraseLevel :C)") is encountered.
It is better to chose the "by phrase break" method, because it uses the work previously done to determine the PhraseLevels, contrary to the disctree method.

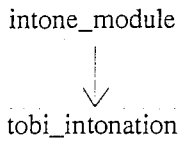
1.3.7 The intone_module function

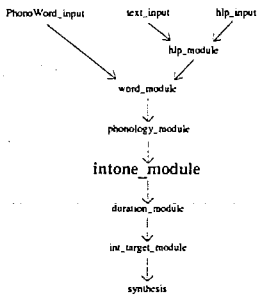
Location

The *intone_module* function can be found in :

```
~chatr/src/intonation/intonation.c
```

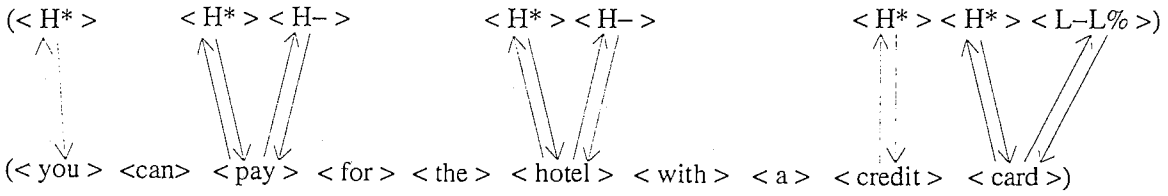
Overview



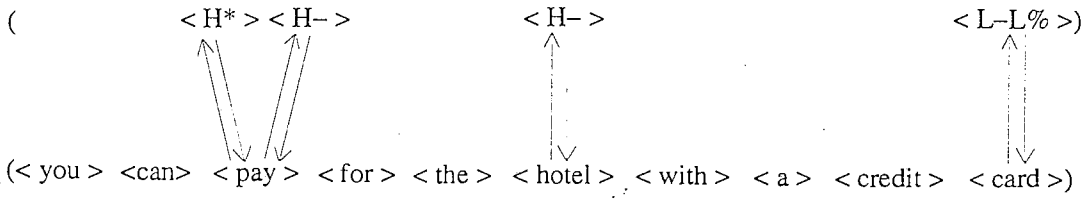


Analysis

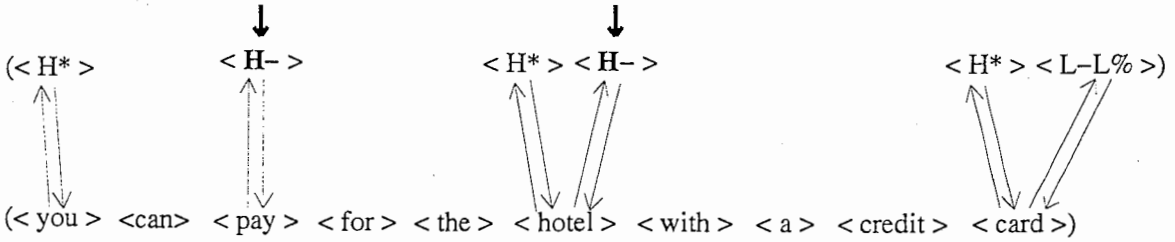
The *intone_module* function fills the IntoneStream. The method currently used at the time of this report is the ToBI intonation method (H*, L*, H-,...). For each syllable of the SylStream the *tobi_intonation* function predicts pitch accents (H*, L*,...), phrase accents (H-, L-,...) and boundaries tones (L%,...).
This CHATR module has clearly a problem. It deletes the IntoneStream, putting to garbage all the work that has been done on it by the previous modules. It is easy to prevent it from doing this (just by commenting out the guilty code) but then new Intone cells will be added to the IntoneStream. It is going to create confusion with the previous Intone cells. Moreover the prediction of pitch accents is only made on syllables that have a "prom" feature, that means that are linked to a "(Prom +)" or a "(HAccent +)" feature. The "(Prom +)" feature seems to have never been explicitly declared before this module. The "(HAccent +)" feature is only for HLP and Text Utterance. So in fact the pitch accents prediction can only be made for these two utterance types. But the *hlp_module* function gives so much information that it is really no use to add more.
In fact only the phrase accents and boundaries prediction is interesting but only for not HLP or Text utterance (because it has already been done), like a PhonoWord one.
So by switching off the pitch accents prediction and allowing phrase accents and boundaries tones prediction only for not HLP or Text utterance, the *intone_module* function will not affect the IntoneStream in the Text Input case :



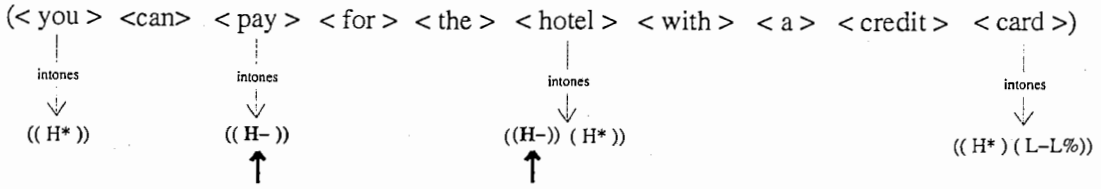
and in the HLP Input case :



But it will affect it in the PhonoWord Input case :



as well as the WordStream :



(in fact other intone cells are added, but they are similar to some previous ones, so I did not mention them. For example a "(L-L%)" intone cell is added on "card" whereas the same cell already exists)

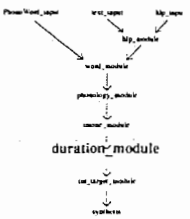
1.3.8 The duration_module function

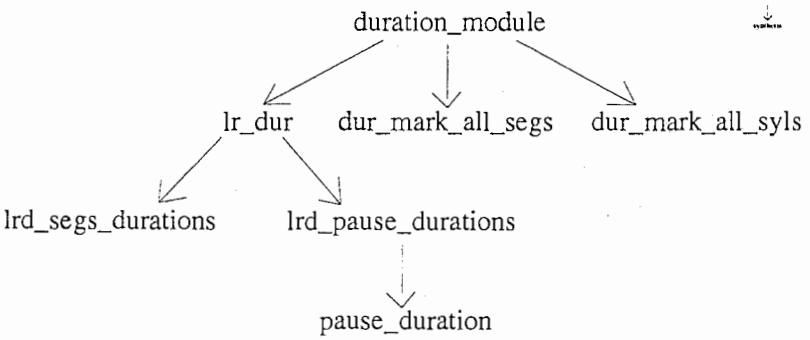
Location

The *duration_module* function can be found in :

`~chatr/src/duration/duration.c`

Overview





Analysis

The *duration_module* function determines the duration of the segments (by the *lrd_segs_durations* function) and of the pauses (by the *lrd_pause_durations* function, according to the Phrase-Level type) and marks absolute starts for segments (by the *dur_mark_all_segs* function)

and syllables (by the *dur_mark_all_syls* function).
For instance for the sentence of this document here are the segments with their absolute start (in ms) :

	Phonoword	Text	HLP
y	0	0	0
uu	66	64	61
k	118	117	89
@	237	235	208
n	276	274	247
p	317	315	288
ei	419	423	396
#	538	565	538
f	638	665	638
@	737	764	737
dh	790	817	790
@	837	864	837
h	893	920	893
ou	985	1012	979
t	1111	1138	1083
e	1205	1232	1169
l	1333	1360	1278
#	1421	1448	1361
w	1521	1548	1461

(a ``\#'' indicates a pause)

and the syllables with their absolute start (in ms) :

	PhonoWord	Text	HLP
(y uu)	0	0	0
(k @ n)	118	117	89
(p ei)	317	315	288
(f @)	638	665	638
(dh @)	790	817	790
(h ou)	893	920	893
(t e l)	1111	1138	1083
(w i dh)	1521	1548	1461
(@)	1664	1691	1604
(k r e)	1716	1743	1656
(d i t)	1926	1987	1866
(k aa d)	2112	2198	2052

(as prosody is different for each kind of input, durations are different)

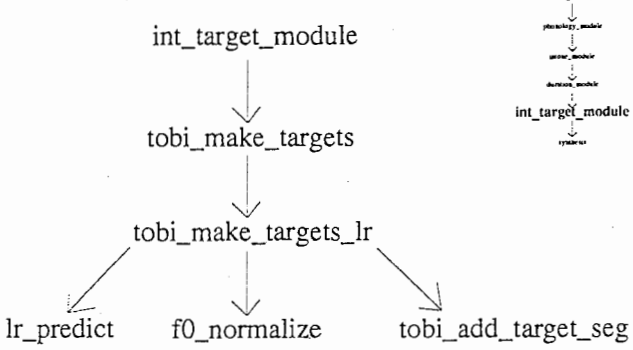
1.3.9 The int_target_module function

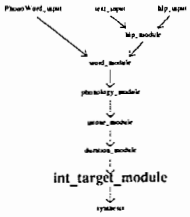
Location

The *int_target_module* function can be found in :

```
~chatr/src/intonation/intonation.c
```

Overview





(the *tobi_make_targets* and *tobi_make_targets_lr* can be found in :

```
~chatr/src/intonation/ToBI.c )
```

Analysis

The *int_target_module* function determines the f0 value for each syllable of the SylStream. At the time this document has been written the method which is used is linear regression for ToBI intonation. The *make_targets_lr* function gives in fact 3 (normalized) f0 values for each syllable. The first one for the first segment of the syllable, the second one for the nucleus segment (the vowel) of the syllable and the third one for the last segment of the syllable. For instance here are the f0 values for the example of this document :

```
        PhonoWord           Text                HLP

* (y uu)                                                    (syllable)
        196.917068          192.78118           176.45158   (f0 before normalisation)
  y     142.290955    y     140.17704     y     131.83081   (1st segment + f0 normalized)
        256.088196          252.19085           205.92985
  uu    172.533966    uu    170.54199     uu    146.89747   (nucleus segment)
        242.849289          239.84178           209.83558
  uu    165.767410    uu    164.23024     uu    148.89373   (last segment)
* (k @ n)
        204.067780          201.49116           173.36627
  k     145.945755    k     144.62881     k     130.25387
        228.281631          228.18071           215.93901
  @     158.321716    @     158.27014     @     152.01327
        213.495285          231.47927           225.05279
```

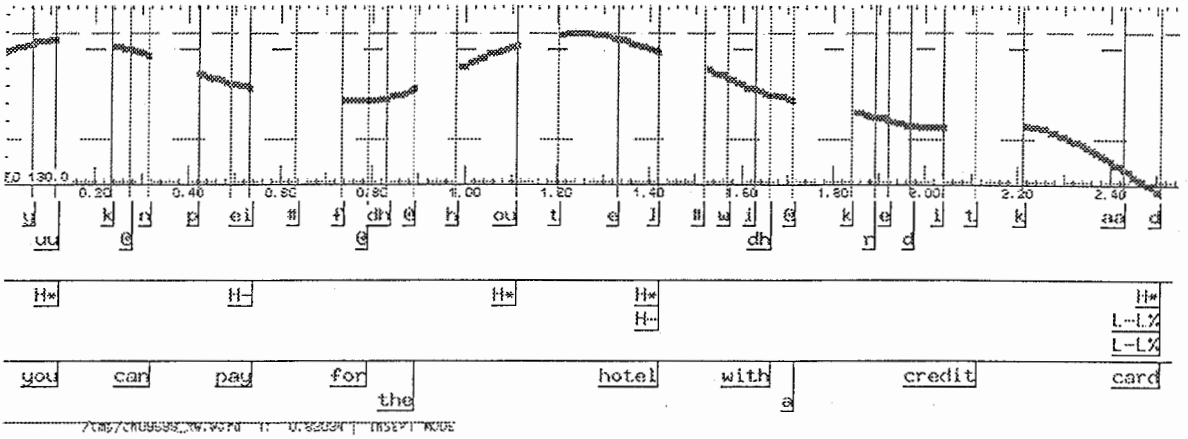
```
  n      150.764252    n      159.95608    n      156.67143
* (p ei)
         192.711884           209.04147           201.94967
  p      140.141632    p      148.48786    p      144.86317
         181.688629           227.94963           229.74664
  ei     134.507523    ei     158.15203    ei     159.07051
         161.305099           191.31132           196.99272
  ei     124.089272    ei     139.42578    ei     142.32962
* (f @)
         169.542007           199.16011           201.15170
  f      128.299255    f      143.43739    f      144.45532
         151.233627           170.03631           171.95320
  @      118.941635    @      128.55189    @      129.53163
         151.086487           161.10058           164.70037
  @      118.866425    @      123.98474    @      125.82463
* (dh @)
         151.363708           164.91391           171.62640
  dh     119.008118    dh     125.93377    dh     129.36460
         156.121429           165.93020           169.92852
  @      121.439842    @      126.45321    @      128.49679
         173.493652           187.45507           172.47857
  @      130.318985    @      137.45481    @      129.80015
* (h ou)
         171.729584           190.19567           176.44268
  h      129.417343    h      138.85557    h      131.82626
         210.256027           230.74173           184.58163
  ou     149.108643    ou     159.57910    ou     135.98617
         218.643097           226.80609           178.81588
  ou     153.395355    ou     157.56755    ou     133.03923
* (t e l)
         223.215790           227.31828           178.08351
  t      155.732513    t      157.82934    t      132.66490
         242.745041           237.77400           179.66101
  e      165.714127    e      163.17337    e      133.47117
         204.412720           193.96093           163.17901
  l      146.122055    l      140.78002    l      125.04705
* (w i dh)
         202.834274           196.86778           163.32327
  w      145.315292    w      142.26574    w      125.12078
         165.008423           156.45224           141.01254
  i      125.982086    i      121.60892    i      113.71752
         159.509415           153.40560           150.44239
  dh     123.171478    dh     120.05175    dh     118.53722
* (@)
         162.552399           155.83990           151.50260
  @      124.726784    @      121.29595    @      119.07910
         155.529129           151.53080           150.36933
  @      121.137108    @      119.09352    @      118.49988
```

```
        157.041901          172.01840          155.39329
  @     121.910309    @     129.56495    @     121.06768
* (k r e)
        163.318283          177.07127          163.75167
  k     125.118233    k     132.14753    k     125.33974
        137.365234          183.52533          139.22203
  e     111.853340    e     135.44627    e     112.80236
        137.160309          185.15049          137.73902
  e     111.748604    e     136.27691    e     112.04439
* (d i t)
        134.786163          186.64566          133.66029
  d     110.535149    d     137.04110    d     109.95970
        121.638107          184.22201          119.69841
  i     103.815033    i     135.80236    i     102.82363
        126.368195          167.65847          105.95538
  t     106.232635    t     127.33655    t     95.799423
* (k aa d)
        145.771805          190.14331          125.73971
  k     116.150032    k     138.82879    k     105.91140
        119.275711          136.78218          70.974113
  aa    102.607590    aa    111.55533    aa    77.920105
        52.435909           65.044205          20.000900
  d     68.445023     d     74.889259    d     51.867126
```

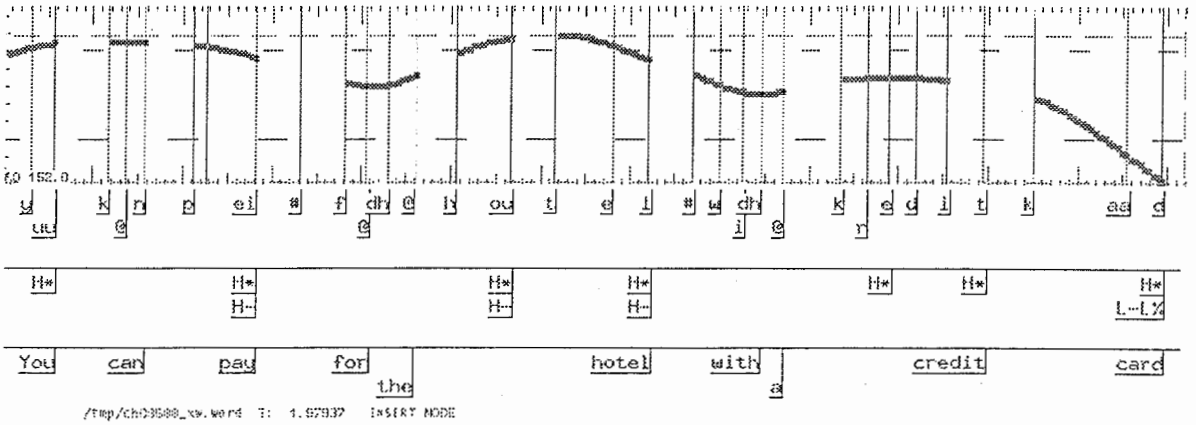
(as for durations, f0 values are different for each kind of input, because they have different prosodies)

Now that duration values and f0 values have been determined, f0 contours can be built :

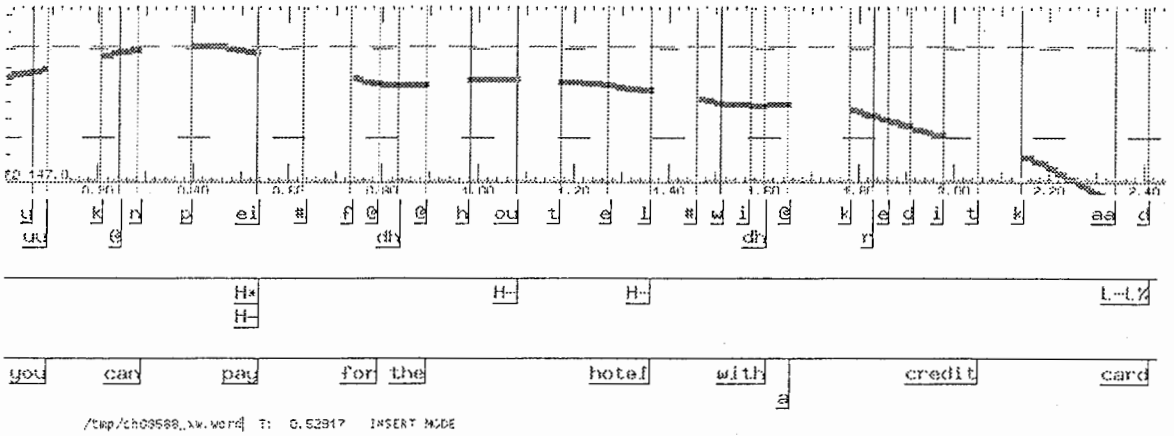
- in the PhonoWord Input case :



- in the Text Input case :



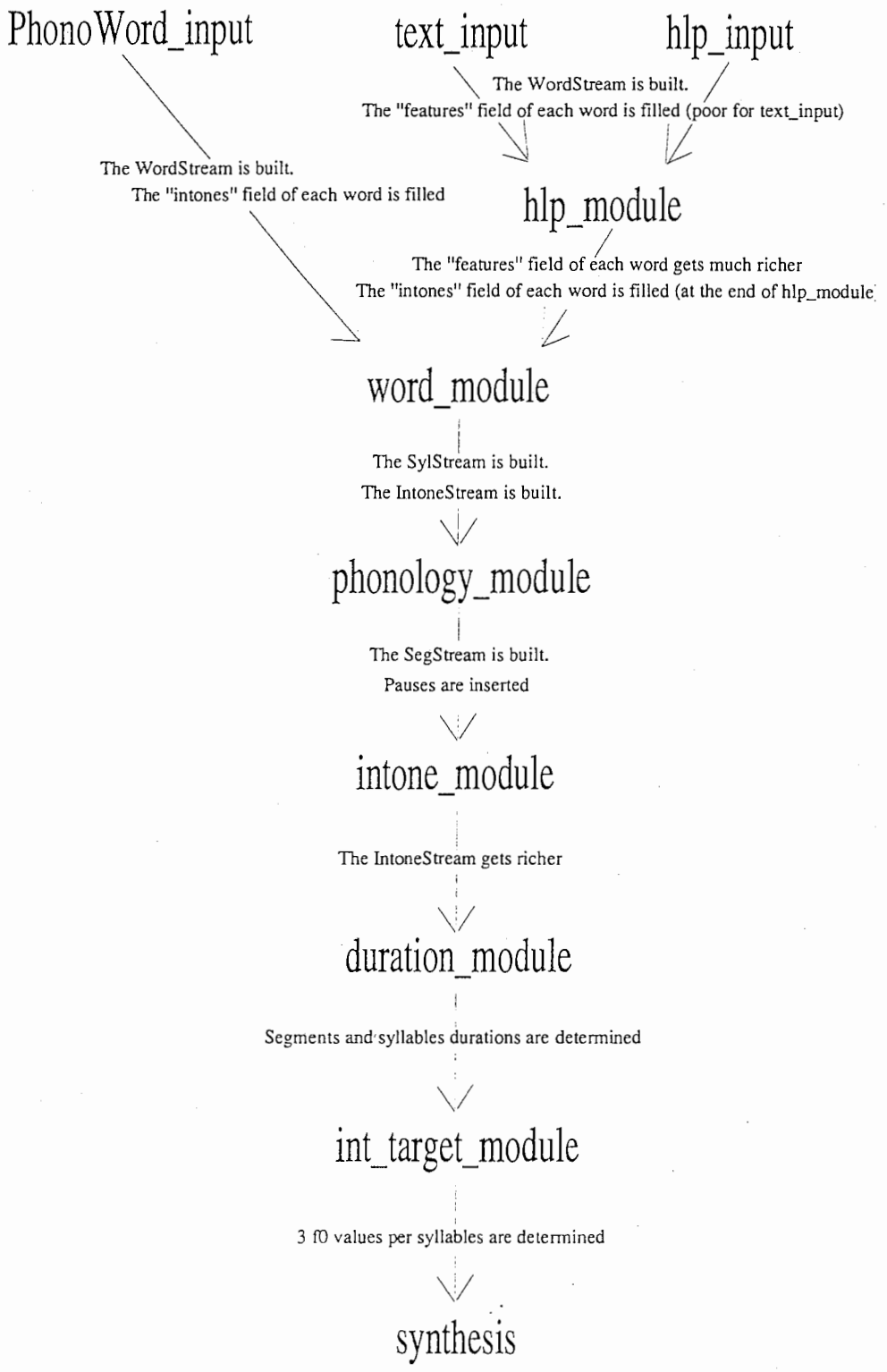
- in the HLP Input case :



Again (cf the *hlp_realise_accents* function), whereas HLP has the most developed code part, its F0 contour is rather flat. But it is just a matter of setting the "HLP_Pattern" variable correctly (easy).
Anyway, at the time this document has been written, it was unfortunately not possible to notice a real difference in the output whatever the input way was (using a raw waveform concatenation without signal processing; problem of database, units selection ?).

1.4 Summary

The following graph quicky summarizes what happens to the WordStream, the SylStream and the IntoneStream through CHATR :



1.5 Getting information from CHATR

To understand CHATR it is important to have the possibility to know what each called function is doing and modifying. Here are some useful lines of code.

1.5.1 Getting information about the WordStream

In all this section "w" is the name of the supposed current WordStream, "utt" the name of the Utterance to be synthesized.

The word itself

```
P_Message (``%s'',SC(w,Word)->text);
```

The features list

```
P_Message (``%s'',pprint(SC(w,Word)->features));
/* pprint is used to print lists, very very useful! */
```

The intones list

```
P_Message (``%s'',pprint(SC(w,Word)->intones));
```

The right and left boundaries

```
P_Message (``%d %d'',SC(w,Word)->left_boundary,
                     SC(w,Word)->right_boundary);
```

The decomposition into syllables

```
P_Message (``%s'',pprint(SC(w,Word)->lexentry));
```

All the words

```
for (w=WORDSTREAM(utt);w!=SNIL;w=SC_next(w))
  P_Message (``%s'',SC(w,Word)->text);

/* w=WORDSTREAM(utt) is the same as */
/* w=utt_stream(``Word'',utt)       */
```

1.5.2 Getting information about the SylStream

In all this section "s" is the name of the supposed current SylStream, "utt" the name of the Utterance to be synthesized.

The syllable itself

```
P_Message (``%s'',SC(s,Syl)->text);
```

The stress value (0 or 1)

```
P_Message (``%d'',SC(s,Syl)->lex_stress);
```

The phrase break before the syllable and the one after

```
P_Message (``%d %d'',SC(s,Syl)->ph_initial,SC(s,Syl)->ph_final);
```

All the syllables

```
for (s=SYLSTREAM(utt);s!=SNIL;s=SC_next(s))
  P_Message (``%s'',SC(s,Syl)->text);
```

1.5.3 Getting information about the IntoneStream

In all this section "intone" is the name of the supposed current SylStream, "utt" the name of the Utterance to be synthesized.

The intone name

```
P_Message (``%s'',SC(intone,Intone)->name);
```

The pitch accentuation (has it a "*" or not)

```
P_Message (``%d'',SC(intone,Intone)->accented);
```

The intone type (pitch accent, boundary tone or phrase accent)

```
P_Message (``%d'',SC(intone,Intone)->type);
```

All the intones

```
for (intone=SYLSTREAM(utt);s!=SNIL;s=SC_next(s))
  P_Message (``%s'',SC(intone,Intone)->name);
```

1.5.4 Miscellaneous

Getting the word matching with a syllable or an intone

```
P_Message (``%s'',SC(Rword1(s),Word)->text);
P_Message (``%s'',SC(Rword1(intone),Word)->text);
```

Getting the syllables matching with a word

```
for (s=Rsyl(w);s!=NIL;s=cdr(s))
  P_Message (``%s'',SC(s,Syl)->text);
```

Getting the intones matching with a word

```
for (intone=Rintone(w);intone!=NIL;intone=cdr(intone))
  P_Message (``%s'',SC(intone,Intone)->name);
```

Checking the presence of an HLP feature

```
if (hlp_has_feat(word,"NAccent","++"))...

if (hlp_has_feat(word,F_PHRASELEVEL,":C"))...

if (hlp_has_feat(word,"Focus",NULL))...
/* means (Focus +) or (Focus ++) or (Focus -) */
```

Chapter 2

The improvements to prosody in CHATR

Two kinds of improvements can be distinguished : the ones that are now part of CHATR and the ones that could be integrated into CHATR.

2.1 The modifications brought to CHATR

These modifications all concern the intonation directory and especially two programs, "intonation.c" and "ToBI.c".

2.1.1 The add_intonation function

This function can be found in :

```
~chatr/src/intonation/intonation.c
```

This function contained a serious mistake. It did not link the newly-built IntoneStream with the SylStream. Without this link f0 and duration values were not determined according to the prosodic information contained in the IntoneStream. See Appendix A.1.1 and A.1.2 to have a look at the modifications.

2.1.2 The tobi_intonation function

This function can be found in :

```
~chatr/src/intonation/ToBI.c
```

One of the first lines of code of this function commanded to delete the existing IntoneStream. But prior to the execution of the *tobi_intonation* function the IntoneStream has already started to be built (by the *add_intonation* function. So to prevent the deletion of the IntoneStream, the line of code in question function has been commented out (in fact the problem here is a bit more complicated and will be discussed again later on). See Appendix A.2.1 and A.2.2 to have a look at the modifications.

2.2 The next possible modifications to CHATR

2.2.1 The tobi_intonation function

This function can be found in :

`~chatr/src/intonation/ToBI.c)`

The problems of this function have already been discussed in the first chapter (see section 1.3.7). To avoid them the best now should be to comment out the pitch accent prediction piece of code, and not to allow the phrase accents and boundary tones prediction piece of code for HLP and Text Inputs. This is undoubtedly a temporary solution; the *tobi_intonation* function will have probably to be rewritten.
(there is also an another mistake : the IntoneStream and the WordStream are not linked together. But this is not as serious as not to link the IntoneStream and the SylStream).
See Appendix B.1.1 and B.1.2 to have a look at the possible modifications.

2.2.2 The tobi_make_targets_lr function

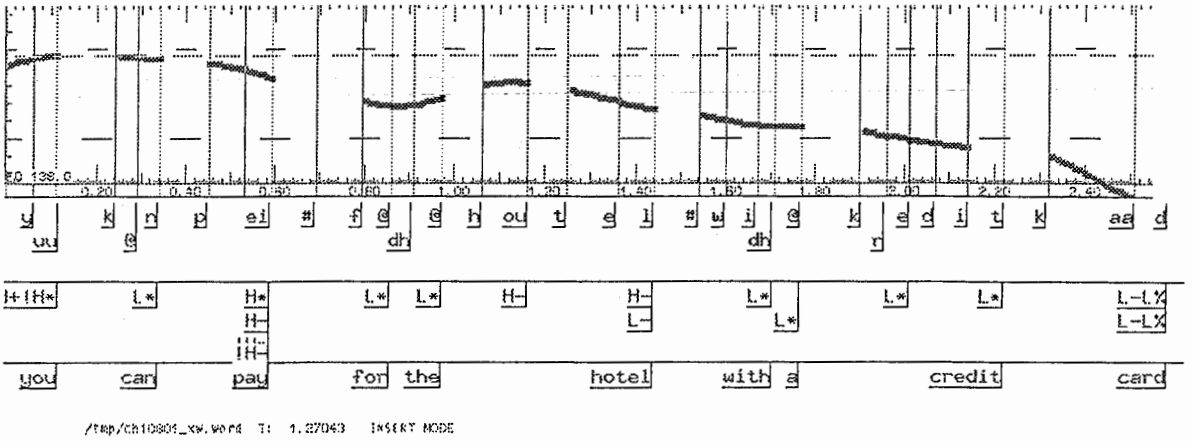
This function can be found in :

`~chatr/src/intonation/ToBI.c`

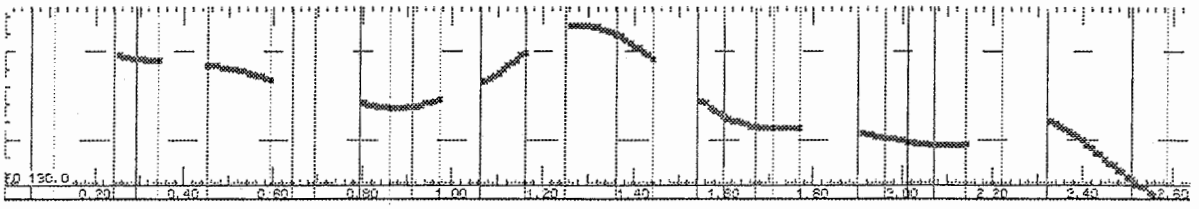
Often, "Focus ++" marked or not, a word sounds the same (approximatively only one time out of three, you can say which word is "Focus ++" marked). So it would be interesting to really increase the f0 values of a "Focus ++" marked word. This have been tried in the *tobi_make_targets_lr* function.
The values of the increase_f0 factor, the decrease_f0 factor (for "Focus -" marked word) and the mode are kept in the "Tony_Params" variable (sorry I lacked imagination the day I created it). Setting up mode to 1 means that only the syllable before the stressed syllable of the "Focus ++" marked word and the syllable after will have their f0 value divided; setting it up to 0 means that all the syllables but the ones of the "Focus ++" marked word will have their f0 value divided) :

```
(set Tony_Params
     '(( mode                    1 )
       ( increase_duration       3 )
       ( decrease_duration       3 )
       ( increase_f0             4 )
       ( decrease_f0             4 )
       ( reduction_set          (the   thi)
                                (have  v)
                                (has   s)
                                (are   r)
                                (is    s)
                                (am    m)
                                (can   c'n)
                                (will  l)
                                (had   d)
                                (would d))))
```

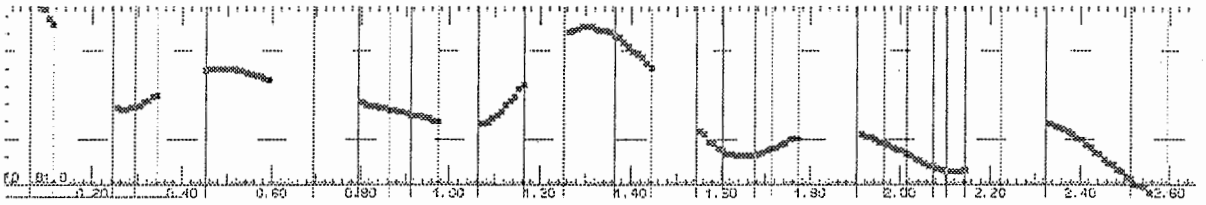
(there is other parameters, used in an another function)
So for instance let's take the HLP example of this document :



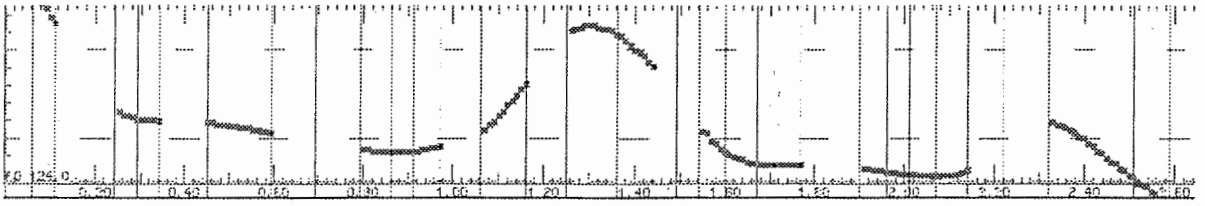
By setting up the increase_f0 factor to 1.5 the f0 values of the stressed syllables matching with the "Focus ++" marked words ("you", "hotel" and "card") are increased by 50% :



By setting up the decrease_f0 factor to 1.5 the f0 values of the syllables just before and just after a stressed syllable of a "Focus ++" marked word, or the ones matching with a "Focus -" marked word are decreased by 50% :



By setting up the mode to 0 all the syllables but the stressed ones matching with a "Focus ++" marked word have their f0 values decreased (by 50%)



See Appendix B.2.1 and B.2.2 to have a look at the possible modifications of the code.

2.2.3 The text_input function

This function can be found in :

```
~chatr/src/input/hlp_input.c
```

When a Text Input is used, the user may want to give some prosody indications (without having to use a rather complicated HLP Input). The solution is to use "escape characters". For instance :

```
chatr> (SayText ``/+/You /r/can pay for the /s+/hotel with a credit
/s+/card'')
```

This means add a "Focus ++" on "you", "hotel" and "card", replace "can" by its reduced form ("c'n") and slow down (increase duration) for "hotel" and "card".
The available escape characters are the following :

/+/	adds a "Focus ++"
/-/	adds a "Focus -"
/s/	increases the duration (slows down)
/f/	decreases the duration (goes faster)
/r/	replaces with the reduced form (if it exists)

The values of the increase_duration and decrease_duration factors, as well as the set of the reducable words are kept in the Tony_Params variable :

```
(set Tony_Params
     '(( mode                    1 )
       ( increase_duration       3 )
       ( decrease_duration       3 )
       ( increase_f0             4 )
       ( decrease_f0             4 )
       ( reduction_set          (the   thi)
                                (have  v)
                                (has   s)
                                (are   r)
                                (is    s)
                                (am    m)
                                (can   c'n)
                                (will  l)
                                (had   d)
                                (would d))))
```

Go to Appendix B.3.1 and B.3.2 to see how to do that.

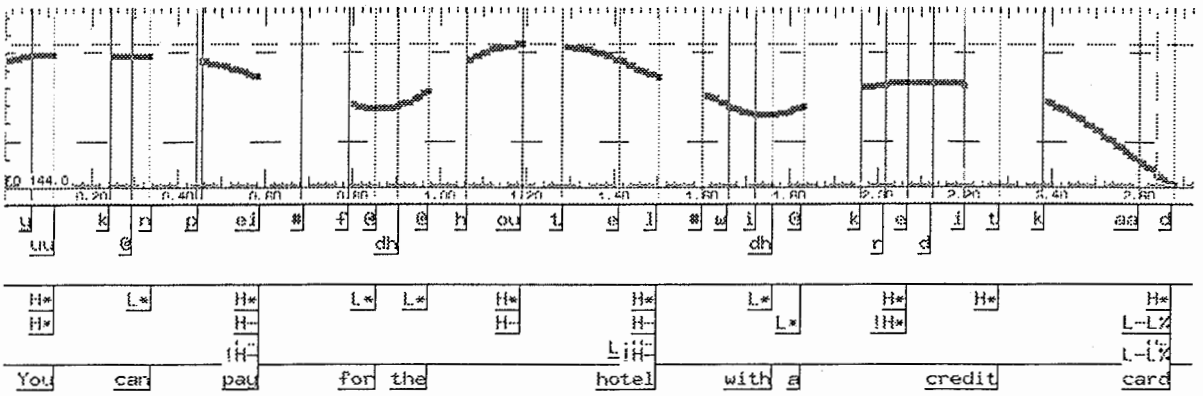
2.2.4 The duration_module function

This function can be found in :

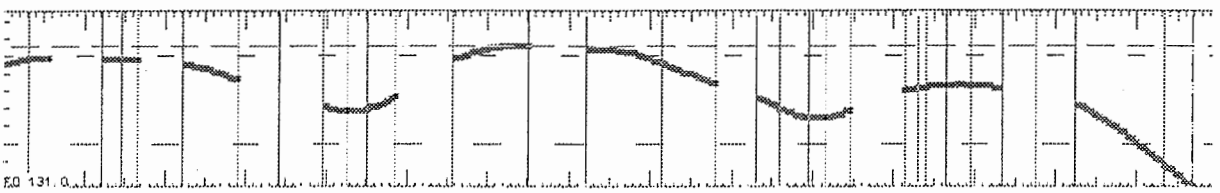
```
~chatr/src/duration/duration.c
```

This function needs to be modified to be able to increase or to decrease duration in case of a "(OPT Slow)" or a "(OPT Fast)" feature (added when a "/s/" or "/f/" escape character has been detected previously).
So for instance let's take the Text example of this document, along with some escape characters :

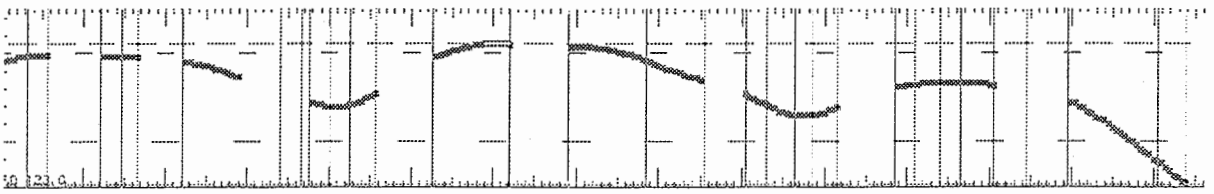
```
chatr> (SayText ''You can pay /f/for the /s/hotel with a credit card'')
```



By setting up the increase_duration factor to 1.5 and the decrease_duration factor to 1.0, the f0_contour becomes :



And by setting up now the decrease_duration factor to 1.5, the f0_contour becomes :



See Appendix B.4.1 and B.4.2 for needed modifications.

Chapter 3

The possible use of a new module for CHATR

The authors of this module are E. Black, S. Eubank, H. Kashioka, C. MacDonald, D. Magerman and T. saiga (ATR-ITL, Dept 3). Basically the module they have created is a parser that takes a text file as input and gives as output a bracketted form of it, along with numerous phrasal and lexical information. By using the bracketing and the phrasal and lexical information it is possible to predict pauses. It may also be possible to predict pitch accents, but this has not really been tested. If this module can run in real-time, integrate it within CHATR will greatly help in the case of Text input.

3.1 Description of the offered output

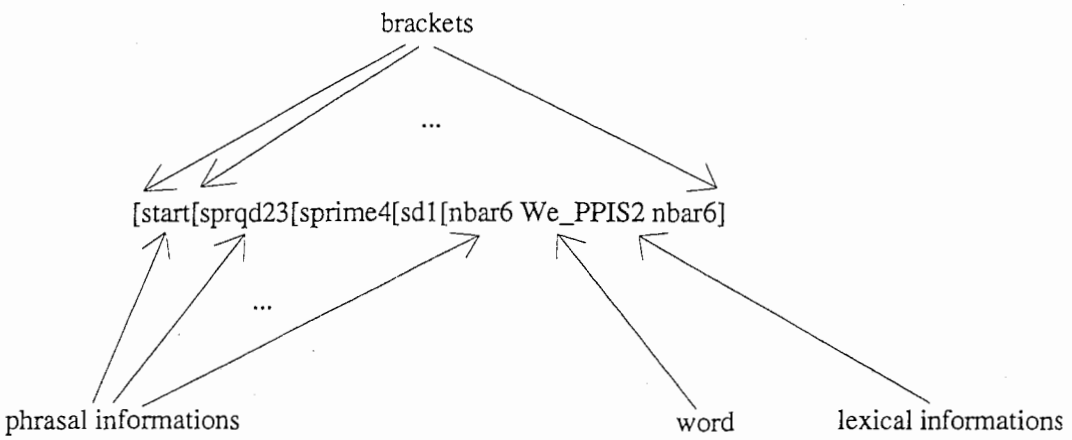
Given the following text input :

```
''We do charge a cancellation fee of three hundred and fifteen dollars
if you cancel less than a week in advance''
```

(I take a new example, longer than the previous one, because it is with long sentences that CHATR has problems for pause prediction; so let's see how this new module handles with it)
, the offered output is :

```
[start [sprpd23 [sprime4 [sd1 [nbar6 We_PPIS2 nbar6] [vbar2 [o8 do_VDO
o8] [v2 charge_VVICOMP-B [nbarq4 [nbar4 [d1 a_AT1 d1] [n4 [n1a cancellation_NN1
FUNCTION n1a] [n1a fee_NN1MONEY n1a] n4] nbar4] [i1e [p1 of_IIOF [nbar1 [n1c
[multiword4 three_MPRICEWORD51 hundred_MPRICEWORD52 and_MPRICEWORD53
fifteen_MPRICEWORD54 dollars_MPRICEWORD55 multiword4] n1c] nbar1] p1] i1e]
nbarq4] v2] vbar2] sd1] [iebar11 [fa1 if_CSIF [sd1 [nbar6 you_PPY
nbar6] [vbar1 [v2 cancel_VVOINCHOATIVE [nbar2 [d25 less_DAR [fc1 than_CSN
[nbarq4 [nbar4 [d1 a_AT1 d1] [n1a week_NR1 n1a] nbar4] [i1e [p1 in_IIIN
[nbar1 [n1a advance_NN1TIME n1a] nbar1] p1] i1e] nbarq4] fc1] d25] nbar2] v2]
vbar1] sd1] fa1] iebar11] sprime4] sprpd23] start]
```

(lots of examples of parsed files can be found in /DB/SBNLP/data/lanctb/skel/)
To try to better understand this structured-up output, here is a close-up on the beginning of this example :



Phrasal information are also called "rules". Every opening bracket is followed by a rule and every closing bracket is preceded by a rule, the same that follows the matching opening bracket.

3.2 The SKEL_to_PhonoWord algorithm

This algorithm I have written takes the parser output as input and gives a PhonoWord Utterance as output. It predicts pauses (translated by a "(:C()" in the PhonoWord output), marks the last content word of each phrase (piece of sentence between two pauses) with a H* and reduces usually reduced words. See Appendix C.1 for the code of this algorithm and C.2 for the meanings of the non-terminal labels (like start, nbar, vbar...). The main idea is looking if a word is strongly linked with the word that follows, and by extension, if a group of words (may be one word) is strongly linked with the group of words that follows it. Here is how it works on the previous example.
First let's hide all the phrasal and lexical information (just for clearness) :

[[[[[We]	[[if
[[do]	[[you]
[charge	[[cancel
[[[a]	[[less
[[cancellation]	[than
[fee]]]	[[[a]
[[of	[week]]
[[[three	[[in
hundred	[[advance]]]]]]]]]]]]]]]]
and	
fifteen	
dollars]]]]]]]]]	

The algorithm starts with the last word, here "advance".
It cannot link it with the following word (does not exist !).

It can just delete the non-useful brackets.

The next word (for the algorithm) is "in".
It has no closing bracket and the following word (in the sentence) "advance" has now no opening bracket. That means they really go together. So they can be linked. This makes a two words long group.
Non-useful brackets are deleted.

And so on till "of". The group under construction with "of" at its head is now 6 words long (until "dollars"). It is time to see if its construction can be achieved. Looking at the phrasal information matching with "of", the answer is yes (because of the "p1" phrasal information). The closing brackets too many are moved to the first previous word (in the sentence) which does not belong to a built group, that is, here, "fee" :

[[[[[We]	[[if
[[do]	[you
[charge	[[cancel
[[[a]	[[less
[[cancellation]	[than
fee]]	[[a
*[of	week]
three	in
hundred	advance]]]]]]]]]]]
and	
fifteen	
dollars]*]]]]	

The current word in the algorithm is now "fee".
It cannot be linked with the following word or group (already built).
There is no non-useful brackets to delete.

The algorithm jumps now to "cancellation" and goes on like previously.

After "We" has been processed, the bracketing is :

[[[[We
[do
[charge
[[a
[cancellation
fee]]]]]]
*[of
three
hundred
and
fifteen
dollars]*

[[if
[you
[[cancel
[[less
[than
[[a
week]
in
advance]]]]]]]]]]]

As there is still groups under construction, the algorithm starts again, from the last word, "advance". After the second loop the bracketing becomes :

[[[*[We
do
charge
a
cancellation
fee]*
*[of
three
hundred
and
fifteen
dollars]*

[[if
[you
[[cancel
[[less
[than
[a
week
in
advance]]]]]]]]]]]

And after the third one :

*[We	*[if
do	you
charge	cancel]*
a	*[less
cancellation	than
fee]*	a
*[of	week
three	in
hundred	advance]*
and	
fifteen	
dollars]*	

There is no more groups under construction, so the algorithm has terminated. From the result a PhonoWord Utterance can be easily be built :

```
(Utterance PhonoWord
        (:D ()
                (:S ()
                        (:C ()
                                (We)
                                (do)
                                (charge)
                                (a)
                                (cancellation)
                                (fee (H*)))
                        (:C ()
                                (of)
                                (three)
                                (hundred)
                                (and)
                                (fifteen)
                                (dollars (H*)))
                        (:C ()
                                (if)
                                (you)
                                (cancel))
                        (:C ()
                                (less)
                                (than)
                                (a)
                                (week)
                                (in)
                                (advance (H*))))))
```

3.3 The comparison with the pause prediction by CHATR

To make this comparison, a natural speaker (Nick Campbell) read a large number of texts from the internet. From the obtained waveforms, pauses and prominence have been extracted. These texts have also been processed by CHATR and by the SKEL_to_PhonoWord algorithm. This has allowed to us evaluate how close to natural speech the pause and prominence predictions by CHATR and by the SKEL_to_PhonoWord algorithm were.

See Appendix C.3 for the technical details of this comparison.

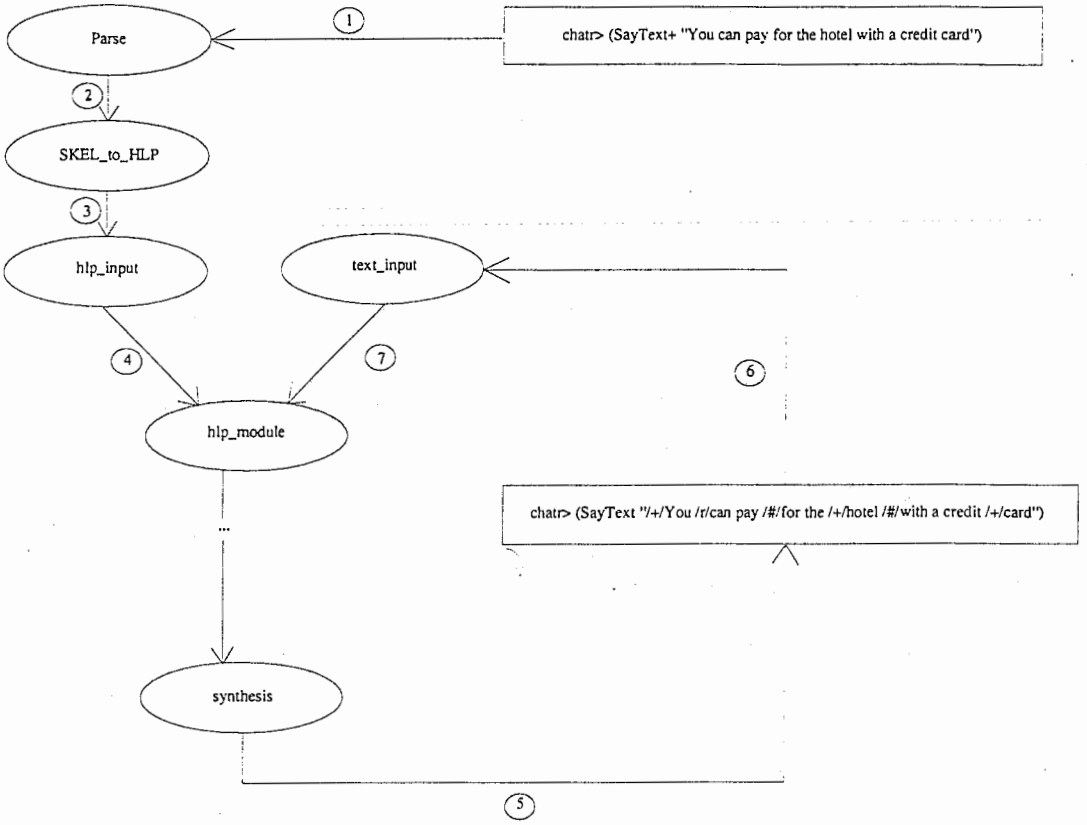
The results of the comparison are nearly always the same, whatever the file chosen for the comparison is. So we can average them and believe that this will be true in the general case :

```
Breaks (pauses) :
                Success rate
SKEL            86%
CHATR           75%

Prominence :
                Success rate
SKEL            74%
CHATR           58%
```

The SKEL_to_PhonoWord algorithm does clearly better, for both pause and prominence prediction, reducing the error rate for pause insertion by 56parser proposed by Dept3 into CHATR.

3.4 Integrating the new module into CHATR

Along with the possible integration of the parser in CHATR, some modifications should be brought about in order to make CHATR's interface more comfortable for users. The easiest way of input for a human user is a text input. Unfortunately prosodically speaking this is the least specified way of input for CHATR. An HLP input is far more specified as far as prosody is concerned. However an HLP input is very constraining to write for a human user. With escape charaters inserted in a Text Input, it is possible to build a good HLP input (escape charaters are translated into HLP features). But if there is too many escape characters to add, it can become constraining again for the user. The best way would be to insert escape characters automatically and give the possibility to the user to modify some of them if the result does not sound totally good. So here is how the parser should be integrated into CHATR, along with some new functionalities :



Step 1 :
The "SayText+" command would be like the "SayText" command, except that instead of inputing the Text Utterance to the *text_input* function, it will input it to the Parser.

Step 2 :
The parser (from Dept 3) will bracket the text given as input and add phrasal and lexical information.

Step 3 :
The *SKEL_to_HLP* function would be a slightly different version of the *SKEL_to_PhonoWord* function. Instead of giving a PhonoWord Utterance as output, it would give a fully-detailed HLP Utterance. For example "(CAT ...)" features for each word (in the parsed text the category of each word is indicated) and "(PhraseLevel :C)" features (the pauses from the pause prediction by the *SKEL_to_PhonoWord* function) could be added.

Step 4 :
The *hlp_input* function is executed.

Step 5 :
After the synthesis, CHATR will return a "SayText" command with the initial text along with escape characters added accordingly to the result of the *hlp_module* function.

Step 6 :
If the user is not pleased with the result of the synthesis, he will have the possiblity to add or delete some escape characters from the offered "SayText" command, and then

send the new Utterance to the text_input function.

Step 7 :
The newly-formed HLP Utterance (from the text_input function) is sent to the *hlp_module* function.

The user could loop steps 5, 6 and 7 as many times as he would like.
In order to make all this work, what has to be done is :

- write the "SayText+" command
- integrate the parser into CHATR
- modify the *SKEL_to_PhonoWord* function in order to get a rich HLP output (and so rename this function as *SKEL_to_HLP*) and integrate it into CHATR too
- write the function that will send back the "SayText" command with text and escape charaters.

During the integration of the *SKEL_to_HLP* function, a lot of attention will have to be paid to not duplicate some functionalities of the *hlp_module* function. For example if "(PhraseLevel :C)" features are added in the *SKEL_to_HLP* function, the *hlp_phr_module* function, which is doing the same job, must be switched off.

Chapter 4

Conclusion

In this conclusion I would like to list all the problems that remain to be solved and the modifications that I think useful to be brought to CHATR :

- set the "HLP_Patterns" variable (for all speakers)
- set the "pause_prediction_method" variable to "by_phrase_break"
- define a list of all the possible features for an HLP Input ("(PROM +)", "(Vphr +)",...?)
- achieve the *aa_assign_accents* function (in hlp_addacc.c)
- allow to mark several words with a "(Focus ++)" feature (see the *hlp_realise_accent* function in hlp.c)
- rewrite the *tobi_intonation* function (in ToBI.c). See section 1.3.7 for reasons
- modify the *reduce_module* function (in reduce.c) so that it can reduce "the" into "thi" when the next word starts with a vowel
- solve the problems of the unit selections (modifying the waveform may be without effect because of a bad unit selection)
- add new escape characters
- make escape characters prevail on the features added by the *hlp_module* function
- add more rules for pause prediction in the *SKEL_to_PhonoWord* algorithm
- integrate the new module into CHATR

This report is now achieved. I hope it will be useful in the coming time. Thanks for reading and excuse the likely mistakes in English.

Appendix A

The modifications brought to CHATR

A.1 The add_intonation function

A.1.1 Before modifications

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

    Stream previous = SNIL;
    Stream start = SNIL;
    List intones;
    Stream newcell;
    Stream words;

    sc_delete_stream("Intone",utt);
    for (words = WORDSTREAM(utt); words != SNIL; words = SC_next(words))
    {
        intones = SC(words,Word)->intones;
while (intones != NIL)
{
    newcell = make_intonation_cell(car(intones));
    SC_previous(newcell) = previous;
    if (start == SNIL)
start = newcell;
    if (previous != SNIL)
    {
SC_next(previous) = newcell;
    }
    link_stream_cells(words,newcell);
    previous = newcell;
    intones = cdr(intones);
}
    }
```

```
    utt_set_stream("Intone",start,utt);
}
```

A.1.2 After modifications

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

    Stream previous = SNIL;
    Stream start = SNIL;
    List intones;
    Stream newcell;
    Stream words,syl;

    sc_delete_stream("Intone",utt);

/* BEGINNING OF THE MODIFICATIONS */

   for (syl = SYLSTREAM(utt); syl != SNIL; syl = SC_next(syl))
/* The loop is now on the SylStream */

    {
        words = Rword1(syl);
/* The Word matching with the current syllable */

intones = SC(words,Word)->intones;
while (intones != NIL)
{
    newcell = make_intonation_cell(car(intones));
    SC_previous(newcell) = previous;
    if (start == SNIL)
start = newcell;
    if (previous != SNIL)
    {
SC_next(previous) = newcell;
    }
    link_stream_cells(words,newcell);
    link_stream_cells(syl,newcell);
/* This links the SylStream and the IntoneStream */

    previous = newcell;
    intones = cdr(intones);
}
    }

/* END OF THE MODIFICATIONS */

    utt_set_stream("Intone",start,utt);
}
```

A.2 The tobi_intonation function (1)

A.2.1 Before modifications

```
void tobi_intonation(Utterance utt)
{
    /* Predict pitch accents, phrase accents and boundary tones */
    /* Create an intone stream accordingly                       */
    /* Accentedness is defined at this point, here we realise    */
    /* the accents themselves                                    */
    Stream s,w,intone;
    List acc_tree, bt_tree, realise_method;
    List accent, tone, newaccent, newtone;

    realise_method = list_str_eval("HLP_realise_strategy",NULL);

    if ((realise_method != NULL) &&
        (list_sequal("Simple_Rules",realise_method)))
        return;   /* its been done by rules or by hand */

    sc_delete_stream("Intone",utt); /* get rid of previous attempts */

    acc_tree = list_str_eval("ToBI_accent_tree",
                "Decision tree for pitch accent realisation not set");

    bt_tree = list_str_eval("ToBI_boundary_tone_tree",
                "Decision tree for boundary tone realisation not set");

    for (s=utt_stream("Syl",utt); s != SNIL; s=SC_next(s))
    {
        w = Rword1(s);
        if (w == SNIL)
            continue;
        /* Predict pitch accent type */
        if ((streq("p",eval_feature("prom",s))) &&
            ((streq("1",eval_feature("stress",s))) ||
             (list_length(Rsyl(w)) == 1)))
        {
            accent = list_last(dt_decide(s,acc_tree));
            newaccent = cons(mkatom(STRVAL(accent)),NIL);

            SC(w,Word)->intones = cons(newaccent,SC(w,Word)->intones);

            intone = make_intonation_cell(newaccent);
            link_stream_cells(intone,s);
            link_stream_cells(intone,s);
            sc_append(intone,"Intone",utt);
        }
```

```
        /* Predict phrase accents and boundary tones */
        tone = list_last(dt_decide(s,bt_tree));
        if (!streq("NONE",STRVAL(tone)))
        {
            newtone = cons(mkatom(STRVAL(tone)),NIL);
            SC(w,Word)->intones = cons(newtone,SC(w,Word)->intones);
            intone = make_intonation_cell(newtone);
            link_stream_cells(intone,s);
            link_stream_cells(intone,s);
            sc_append(intone,"Intone",utt);
        }
    }
}
```

A.2.2 After modifications

```
void tobi_intonation(Utterance utt)
{
    /* Predict pitch accents, phrase accents and boundary tones */
    /* Create an intone stream accordingly                       */
    /* Accentedness is defined at this point, here we realise    */
    /* the accents themselves                                    */
    Stream s,w,intone;
    List acc_tree, bt_tree, realise_method;
    List accent, tone, newaccent, newtone;

    realise_method = list_str_eval("HLP_realise_strategy",NULL);

    if ((realise_method != NULL) &&
        (list_sequal("Simple_Rules",realise_method)))
        return;   /* its been done by rules or by hand */
```

/* BEGINNING OF THE MODIFICATIONS */

```
    /* sc_delete_stream("Intone",utt); (tony) */ /* get rid of previous attempts */
```

/* This line was deleting the IntoneStream */

/* END OF THE MODIFICATIONS */

```
    acc_tree = list_str_eval("ToBI_accent_tree",
                "Decision tree for pitch accent realisation not set");

    bt_tree = list_str_eval("ToBI_boundary_tone_tree",
                "Decision tree for boundary tone realisation not set");

    for (s=utt_stream("Syl",utt); s != SNIL; s=SC_next(s))
    {
        w = Rword1(s);
        if (w == SNIL)
            continue;
        /* Predict pitch accent type */
        if ((streq("p",eval_feature("prom",s))) &&
            ((streq("1",eval_feature("stress",s))) ||
             (list_length(Rsyl(w)) == 1)))
        {
            accent = list_last(dt_decide(s,acc_tree));
            newaccent = cons(mkatom(STRVAL(accent)),NIL);
            SC(w,Word)->intones = cons(newaccent,SC(w,Word)->intones);
            intone = make_intonation_cell(newaccent);
            link_stream_cells(intone,s);
```

```
            link_stream_cells(intone,s);
            sc_append(intone,"Intone",utt);
        }
        /* Predict phrase accents and boundary tones */
        tone = list_last(dt_decide(s,bt_tree));
        if (!streq("NONE",STRVAL(tone)))
        {
            newtone = cons(mkatom(STRVAL(tone)),NIL);
            SC(w,Word)->intones = cons(newtone,SC(w,Word)->intones);
            intone = make_intonation_cell(newtone);
            link_stream_cells(intone,s);
            link_stream_cells(intone,s);
            sc_append(intone,"Intone",utt);
        }
    }
}
```

Appendix B

The possible modifications

B.1 The tobi_intonation function (2)

B.1.1 Before possible modifications

```
void tobi_intonation(Utterance utt)
{
    /* Predict pitch accents, phrase accents and boundary tones */
    /* Create an intone stream accordingly                       */
    /* Accentedness is defined at this point, here we realise    */
    /* the accents themselves                                    */
    Stream s,w,intone;
    List acc_tree, bt_tree, realise_method;
    List accent, tone, newaccent, newtone;

    realise_method = list_str_eval("HLP_realise_strategy",NULL);

    if ((realise_method != NULL) &&
        (list_sequal("Simple_Rules",realise_method)))
        return;   /* its been done by rules or by hand */

    /* sc_delete_stream("Intone",utt); (tony) */ /* get rid of previous attempts */

    acc_tree = list_str_eval("ToBI_accent_tree",
                "Decision tree for pitch accent realisation not set");

    bt_tree = list_str_eval("ToBI_boundary_tone_tree",
                "Decision tree for boundary tone realisation not set");

    for (s=utt_stream("Syl",utt); s != SNIL; s=SC_next(s))
    {
        w = Rword1(s);
        if (w == SNIL)
            continue;
        /* Predict pitch accent type */
        if ((streq("p",eval_feature("prom",s))) &&
```

```
        ((streq("1",eval_feature("stress",s))) ||
         (list_length(Rsyl(w)) == 1)))
    {
        accent = list_last(dt_decide(s,acc_tree));
        newaccent = cons(mkatom(STRVAL(accent)),NIL);

        SC(w,Word)->intones = cons(newaccent,SC(w,Word)->intones);

        intone = make_intonation_cell(newaccent);
        link_stream_cells(intone,s);
        link_stream_cells(intone,s);
        sc_append(intone,"Intone",utt);
    }
    /* Predict phrase accents and boundary tones */
    tone = list_last(dt_decide(s,bt_tree));
    if (!streq("NONE",STRVAL(tone)))
    {
        newtone = cons(mkatom(STRVAL(tone)),NIL);
        SC(w,Word)->intones = cons(newtone,SC(w,Word)->intones);
        intone = make_intonation_cell(newtone);
        link_stream_cells(intone,s);
        link_stream_cells(intone,s);
        sc_append(intone,"Intone",utt);
    }
  }
}
```

B.1.2 After possible modifications

```
void tobi_intonation(Utterance utt)
{
    /* Predict pitch accents, phrase accents and boundary tones */
    /* Create an intone stream accordingly                       */
    /* Accentedness is defined at this point, here we realise    */
    /* the accents themselves                                    */
    Stream s,w,intone;
    List acc_tree, bt_tree, realise_method;
    List accent, tone, newaccent, newtone;

    realise_method = list_str_eval("HLP_realise_strategy",NULL);

    if ((realise_method != NULL) &&
(list_sequal("Simple_Rules",realise_method)))
return;   /* its been done by rules or by hand */

     /* sc_delete_stream("Intone",utt); (tony) */ /* get rid of previous attempts */
    acc_tree = list_str_eval("ToBI_accent_tree",
                "Decision tree for pitch accent realisation not set");

    bt_tree = list_str_eval("ToBI_boundary_tone_tree",
                "Decision tree for boundary tone realisation not set");

    for (s=utt_stream("Syl",utt); s != SNIL; s=SC_next(s))
    {
w = Rword1(s);
if (w == SNIL)
    continue;

/* Predict pitch accent type */
```

/* (1) BEGINNING OF THE MODIFICATIONS */
/* All the pitch accent prediction piece of code is ignored */

```
/*if ((streq("p",eval_feature("prom",s))) &&
    ((streq("1",eval_feature("stress",s))) ||
     (list_length(Rsyl(w)) == 1)))*/

/* the "prom" feature is for syllables linked to a "(Prom +)" or a       */
/* "(HAccent +)" feature. I have never seen a "(Prom +)" feature  and    */
/* "(HAccent +)" features are for HLP. So this prediction will only be   */
/* for HLP and Text Utterance; however for this kind of Utterance this   */
/* work has already been done previously. I prefer commenting out this   */
/* for the moment, otherwise it will create a mess in the IntoneStream.  */
/* tony */
```

```
/*{
/* this must be modified to take in account manual pitch accentuation */
/*  accent = list_last(dt_decide(s,acc_tree));
    newaccent = cons(mkatom(STRVAL(accent)),NIL);
    SC(w,Word)->intones = cons(newaccent,SC(w,Word)->intones);
    intone = make_intonation_cell(newaccent);
    link_stream_cells(intone,s);

/* (2) BEGINNING OF THE MODIFICATIONS */

    link_stream_cells(intone,w);*//* w instead of s previously */
                                  /* tony */

/* This links now the IntoneStream and the WordStream
*/
/* (2) END OF THE MODIFICATIONS */

    /*sc_append(intone,"Intone",utt);
}*/

/* (1) END OF THE MODIFICATIONS */

/* Predict phrase accents and boundary tones */

/* (3) BEGINNING OF THE MODIFICATIONS */

        /* Already done for Text and HLP Utterance. Only useful for
other Utterance types */
        /* like the PhonoWord one */
        /* tony */

tone = list_last(dt_decide(s,bt_tree));

if (!streq("NONE",STRVAL(tone)) &&
(!streq(UTTACTIONFROM(utt),"HLP")) &&
(!streq(UTTACTIONFROM(utt),"Text"))) /* two last conditions by tony */

/* In the case of a Text or an HLP input, the following
instructions will not be executed */

{
    newtone = cons(mkatom(STRVAL(tone)),NIL);
    SC(w,Word)->intones = cons(newtone,SC(w,Word)->intones);
    intone = make_intonation_cell(newtone);
    link_stream_cells(intone,s);

/* (4) BEGINNING OF THE MODIFICATIONS */
```

```
    link_stream_cells(intone,w); /* w instead of s previously */
                                 /* tony */
```

/* Again this links now the IntoneStream and the WordStream */
/* (4) END OF THE MODIFICATIONS */

```
    sc_append(intone,"Intone",utt);
}
```

/* (3) END OF THE MODIFICATIONS */

```
    }
}
```

B.2 The tobi_make_targets_lr function

B.2.1 Before possible modifications

```
void tobi_make_targets_lr(Utterance utt)
{
    /* Predict F0 targets on syllables (start, mid-vowel end) using */
    /* linear regresssion                                           */
    List lr_models, feat_maps;
    List lr_params;
    LR_Model lr_start_model, lr_mid_model, lr_end_model;
    Stream syl,vseg,s;
    float f0_val;

    lr_models = list_str_eval("tobi_lrf0_model","No tobi_lrf0_model set");

    feat_maps = list_str_eval("feature_maps","No feature maps set");

    lr_start_model = make_lr_model(car(lr_models),feat_maps);

    lr_mid_model = make_lr_model(car(cdr(lr_models)),feat_maps);

    lr_end_model = make_lr_model(car(cdr(cdr(lr_models))),feat_maps);

    lr_params = car(cdr(cdr(cdr(lr_models))));

    tobi_model_f0mean = param_get_float(lr_params,"model_f0mean",0.0);

    tobi_model_f0std = param_get_float(lr_params,"model_f0std",1.0);

    /* In PhonoForm input there may already be targets -- delete them */
    for (s=utt_stream("Segment",utt); s != SNIL; s=SC_next(s))
    {

        SC(s,Segment)->num_target = 0;
        xfree(SC(s,Segment)->targ);
        SC(s,Segment)->targ = NULL;
    }

    for (syl=utt_stream("Syl",utt); syl != SNIL; syl=SC_next(syl))
    {
        /* Predict start point */
        f0_val = lr_predict(syl,lr_start_model);
        f0_val = f0_normalize(f0_val);
        vseg = Rseg1(syl);
        tobi_add_target_seg(vseg,
                            SC(vseg,Segment)->start,
                            f0_val);
```

```
        /* Predict mid point */
        f0_val = lr_predict(syl,lr_mid_model);
        f0_val = f0_normalize(f0_val);
        vseg = get_nuclear_seg(syl);
        tobi_add_target_seg(vseg,
                            SC(vseg,Segment)->start +
                            (SC(vseg,Segment)->duration/2),
                            f0_val);
        /* Predict end point */
        f0_val = lr_predict(syl,lr_end_model);
        f0_val = f0_normalize(f0_val);
        vseg = STREAMVAL(list_last(Rseg(syl)));
        tobi_add_target_seg(vseg,
                            SC(vseg,Segment)->start +
                            SC(vseg,Segment)->duration,
                            f0_val);
    }

    fix_last_target(utt);
    free_lr_model(lr_start_model);
    free_lr_model(lr_mid_model);
    free_lr_model(lr_end_model);

    return;
}
```

B.2.2 After possible modifications

```
void tobi_make_targets_lr(Utterance utt)
{
    /* Predict F0 targets on syllables (start, mid-vowel end) using */
    /* linear regresssion                                           */
    List lr_models, feat_maps;
    List lr_params;
    LR_Model lr_start_model, lr_mid_model, lr_end_model;
    Stream syl,vseg,s;
    float f0_val;


/* (1) BEGINNING OF THE MODIFICATIONS */


    List tony_params,p; /*tony*/
    Stream word,current_word,next_syllable_word;/* tony */
    int current_word_is_accented,current_word_is_deaccented,
next_syllable_word_is_accented,there_is_an_accent; /*tony*/
    static float multiply_factor,divide_factor; /* tony */
    int mode; /* tony */
    int stressed_syllable,next_syllable_is_stressed; /* tony */
    int f0_multiplication_on_previous_syllable=FALSE; /* tony */

    tony_params = list_str_eval("Tony_Params",NULL);
    multiply_factor = param_get_float(tony_params,"increase_f0",1.0);
    divide_factor = param_get_float(tony_params,"decrease_f0",1.0);
    mode = param_get_num(tony_params,"mode",1);
/* This was for getting the values of the parameters */


/* (1) END OF THE MODIFICATIONS */

    lr_models = list_str_eval("tobi_lrf0_model","No tobi_lrf0_model set");

    feat_maps = list_str_eval("feature_maps","No feature maps set");

    lr_start_model = make_lr_model(car(lr_models),feat_maps);

    lr_mid_model = make_lr_model(car(cdr(lr_models)),feat_maps);

    lr_end_model = make_lr_model(car(cdr(cdr(lr_models))),feat_maps);

    lr_params = car(cdr(cdr(cdr(lr_models))));

    tobi_model_f0mean = param_get_float(lr_params,"model_f0mean",0.0);
```

```
tobi_model_f0std = param_get_float(lr_params,"model_f0std",1.0);

/* In PhonoForm input there may already be targets -- delete them */
for (s=utt_stream("Segment",utt); s != SNIL; s=SC_next(s))
{

    SC(s,Segment)->num_target = 0;
    xfree(SC(s,Segment)->targ);
    SC(s,Segment)->targ = NULL;
}

for (syl=utt_stream("Syl",utt); syl != SNIL; syl=SC_next(syl))
{
```

/* (2) BEGINNING OF THE MODIFICATIONS */

```
current_word=Rword1(syl);
next_syllable_word=Rword1(SC_next(syl));
```

/* current_word is the word matching with the current syllable, next_syllable_word the one matching with the following syllable */

```
/* checks if there is a "Focus ++" or a "Focus -" on the current */
/* word */ /* tony */
current_word_is_accented=FALSE;
current_word_is_deaccented=FALSE;
if (hlp_has_feat(current_word,"Focus","++"))
  current_word_is_accented=TRUE;
if (hlp_has_feat(current_word,"Focus","-"))
  current_word_is_deaccented=TRUE;
/* checks if there is a "Focus ++" on the word matching with the */
/* next syllable */ /* tony */
next_syllable_word_is_accented=FALSE;
if (next_syllable_word!=SNIL)
  if (hlp_has_feat(next_syllable_word,"Focus","++"))
    next_syllable_word_is_accented=TRUE;

/* checks if the current syllable is stressed or not */
stressed_syllable=FALSE;
if ((streq("1",eval_feature("stress",syl))) ||
     (list_length(Rsyl(current_word)) == 1))
  stressed_syllable=TRUE;/* want to increase the f0 value of only */
                         /* one syllable */ /* tony */

 /* checks if the next syllable is stressed or not */
next_syllable_is_stressed=FALSE;
```

```
      if ((streq("1",eval_feature("stress",SC_next(syl)))) ||
           (list_length(Rsyl(Rword1(SC_next(syl)))) == 1))
        next_syllable_is_stressed=TRUE;
```

/* this was a setting of different boolean variables */

```
      /* Predict start point */

      f0_val = lr_predict(syl,lr_start_model);
      /*f0_val = f0_normalize(f0_val);*/ /* doing this now really modifies */
                                          /* the waveform */        /* tony */
      /* P_Message ("start point before : %f",f0_val); */
      if ((current_word_is_accented) && (stressed_syllable)) /* tony */
```

/* stressed syllable of a "Focus ++" marked word */

```
        {
          f0_multiplication_on_previous_syllable=TRUE;
          f0_val = f0_val*multiply_factor;
         }
      else if (mode==1) /* if (mode 1) has been chosen)*/ /* tony */
            {
              if ((((next_syllable_word_is_accented) &&
(next_syllable_is_stressed)) ||
(f0_multiplication_on_previous_syllable)) ||
(current_word_is_deaccented))
```

/* before or after a stressed syllable of a "Focus ++" marked word, or matching a "Focus -" marked word*/

```
                    f0_val = f0_val/divide_factor;
            }
      else
```

/* mode 0 chosen so all the syllables but the stressed ones of "Focus ++" marked words have their f0 values divided */

```
              f0_val = f0_val/divide_factor;

      /* P_Message ("start point after : %f",f0_val); */
      f0_val = f0_normalize(f0_val);
      vseg = Rseg1(syl);
      tobi_add_target_seg(vseg,SC(vseg,Segment)->start,f0_val);

        /* Predict mid point */

      f0_val = lr_predict(syl,lr_mid_model);
      /*f0_val = f0_normalize(f0_val);*/

      /* P_Message ("mid point before : %f",f0_val); */
```

```
      if ((current_word_is_accented) && (stressed_syllable)) /* tony */
       {
          f0_val = f0_val*multiply_factor;
        }
      else if (mode==1)
        {
          if ((((next_syllable_word_is_accented) &&
(next_syllable_is_stressed))
|| (f0_multiplication_on_previous_syllable)) || (current_word_is_deaccented))

              {
  f0_val = f0_val/divide_factor;
          f0_multiplication_on_previous_syllable=FALSE;
      }
        }
      else
          f0_val = f0_val/divide_factor;

      /* P_Message ("mid point after : %f",f0_val); */
      f0_val = f0_normalize(f0_val);
      vseg = get_nuclear_seg(syl);
      tobi_add_target_seg(vseg,SC(vseg,Segment)->start
+ (SC(vseg,Segment)->duration/2),f0_val);

        /* Predict end point */

      f0_val = lr_predict(syl,lr_end_model);
      /*f0_val = f0_normalize(f0_val);*/

      /* P_Message ("end point before : %f",f0_val); */

      if ((current_word_is_accented) && (stressed_syllable)) /* tony */
        {
          f0_val = f0_val*multiply_factor;
        }
      else if (mode==1)
        {
          if ((((next_syllable_word_is_accented) &&
(next_syllable_is_stressed))
|| (f0_multiplication_on_previous_syllable)) || (current_word_is_deaccented))
            {
              f0_multiplication_on_previous_syllable=FALSE;
              f0_val = f0_val/divide_factor;
             }
        }
      else
          f0_val = f0_val/divide_factor;
```

```
      /* P_Message ("end point after : %f",f0_val); */
      f0_val = f0_normalize(f0_val);
      vseg = STREAMVAL(list_last(Rseg(syl)));
      tobi_add_target_seg(vseg,SC(vseg,Segment)->start +
SC(vseg,Segment)->duration,f0_val);
```

/* (2) END OF THE MODIFICATIONS */

```
    }

    fix_last_target(utt);
    free_lr_model(lr_start_model);
    free_lr_model(lr_mid_model);
    free_lr_model(lr_end_model);

    return;

}
```

B.3 The text_input function

B.3.1 Before possible modifications

```
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 continus in the        */
    /* same way                                                            */
    List 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);
    list_free_tree(input);
}
```

B.3.2 After possible modifications

/* BEGINNING OF A NEW FUNCTION */

```
void get_and_apply_options (Utterance utt) /* tony */
/* interpretes the escape characters */
{
     Stream w;
     int i,j,reduction_asked;
     List tony_params, reduction_set;

     tony_params = list_str_eval("Tony_Params",NULL);
     reduction_set = param_get_list(tony_params,"reduction_set",NIL);
     for (w=WORDSTREAM(utt);w!=SNIL;w=SC_next(w))
      {
        reduction_asked=FALSE;
        i=0;
        if (SC(w,Word)->text[i]=='/') /* I tried to start options with  */
                                      /* '\' but CHATR ate it.          */
          {
            while ((SC(w,Word)->text[++i]!='/') && (SC(w,Word)->text[i]!='\0'))
              {
                switch (SC(w,Word)->text[i]) {
                case '+':
                  hlp_delete_feat(w,"Focus","-");
                  hlp_add_feat(w,"Focus","++");
                  break;
                case '-':
                  hlp_delete_feat(w,"Focus","++");
                  hlp_add_feat(w,"Focus","-");
                  break;
                case 's':
                  hlp_delete_feat(w,"OPT","Fast");
                  hlp_add_feat(w,"OPT","Slow");
                  break;
                case 'f':
                  hlp_delete_feat(w,"OPT","Slow");
                  hlp_add_feat(w,"OPT","Fast");
                  break;
                case 'r':
                  reduction_asked=TRUE;
                  break;
                default :
                  break;
                }
              }
            hlp_delete_feat(w,"LEX",SC(w,Word)->text);
            j=i;
            while (SC(w,Word)->text[j]!='\0')
```

```
            {
              ++j;
              SC(w,Word)->text[j-i-1]=SC(w,Word)->text[j];
            }
```

/* this deletes the escape characters from the text */

```
          if (reduction_asked)
            strcpy(SC(w,Word)->text,
param_get_str(reduction_set,SC(w,Word)->text,NULL));
          hlp_add_feat(w,"LEX",SC(w,Word)->text);
        }
      }
    }
```

/* END OF THE NEW FUNCTION */

```
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 continus in the        */
    /* same way                                                            */
    List input;
    /* tony */
    Stream w;

    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);
```

/* BEGINNING OF THE MODIFICATIONS */

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

/* this function is going to interprete the escape characters */

/* END OF THE MODIFICATIONS */

```
    list_free_tree(input);
}
```

B.4 The duration_module function

B.4.1 Before possible modifications

```
void duration_module(Utterance utt)
{
  Stream w; /* tony */
  int start,end; /* tony */

    if (ci_streq(ch_param.dur_method,"klatt_dur"))
klatt_dur(utt);
    else if (ci_streq(ch_param.dur_method,"nnet_dur"))
nnet_dur(utt);
    else if (ci_streq(ch_param.dur_method,"kaiki_dur"))
kaiki_dur(utt);
    else if (ci_streq(ch_param.dur_method,"jvs_dur"))
jvs_dur(utt);
    else if (ci_streq(ch_param.dur_method,"lr_dur"))
lr_dur(utt);
    else if (ci_streq(ch_param.dur_method,"lr_dur_syl"))
lr_dur_syl(utt);
    else if (ci_streq(ch_param.dur_method,"average+"))
    {
assign_syl_durs(utt);
assign_av_seg_dur(utt);
assign_seg_durs(utt);
    }
    else if (ci_streq(ch_param.dur_method,"average"))
assign_av_seg_dur(utt);
    else
    {
P_Error("Undefined duration type %s", ch_param.dur_method);
list_error(On_Error_Tag);
    }

    dur_mark_all_segs(utt);     /* Mark absolute starts for all segs */
    dur_mark_all_syls(utt);     /* Mark absolute starts for all syllables */
}
```

B.4.2 After possible modifications

```
void duration_module(Utterance utt)
{
    if (ci_streq(ch_param.dur_method,"klatt_dur"))
        klatt_dur(utt);
    else if (ci_streq(ch_param.dur_method,"nnet_dur"))
        nnet_dur(utt);
    else if (ci_streq(ch_param.dur_method,"kaiki_dur"))
        kaiki_dur(utt);
    else if (ci_streq(ch_param.dur_method,"jvs_dur"))
        jvs_dur(utt);
    else if (ci_streq(ch_param.dur_method,"lr_dur"))
        lr_dur(utt);
    else if (ci_streq(ch_param.dur_method,"lr_dur_syl"))
        lr_dur_syl(utt);
    else if (ci_streq(ch_param.dur_method,"average+"))
    {
        assign_syl_durs(utt);
        assign_av_seg_dur(utt);
        assign_seg_durs(utt);
    }
    else if (ci_streq(ch_param.dur_method,"average"))
        assign_av_seg_dur(utt);
    else
    {
        P_Error("Undefined duration type %s", ch_param.dur_method);
        list_error(On_Error_Tag);
    }
/* BEGINNING OF THE MODIFICATIONS */
    modif_segs_dur(utt);/* tony; just below */
/* this function modifies durations if asked */
/* END OF A NEW FUNCTION */
    dur_mark_all_segs(utt);        /* Mark absolute starts for all segs */
    dur_mark_all_syls(utt);        /* Mark absolute starts for all syllables */
}
/* BEGINNING OF A NEW FUNCTION */
void modif_segs_dur(Utterance utt) /* tony */
   /* Increase or decrease duration if asked */
{
    Stream s;
    List sl;
    float multiply_factor,divide_factor;
    List p;
```

```
    Stream w;
    List tony_params;

    tony_params = list_str_eval("Tony_Params",NULL);
    multiply_factor = param_get_float(tony_params,"increase_duration",1.0);
    divide_factor = param_get_float(tony_params,"decrease_duration",1.0);
    for (s = utt_stream("Syl",utt); s != SNIL; s=SC_next(s))
    {
      w=Rword1(s);
      if (hlp_has_feat(w,"OPT","Slow"))
        {
          for (sl=Rseg(s); sl != NIL; sl=cdr(sl))
            {
              SC(STREAMVAL(car(sl)),Segment)->duration=
(int)SC(STREAMVAL(car(sl)),Segment)->duration * multiply_factor;
            }
        }
      if (hlp_has_feat(w,"OPT","Fast"))
        {
          for (sl=Rseg(s); sl != NIL; sl=cdr(sl))
            {
              SC(STREAMVAL(car(sl)),Segment)->duration=
(int)SC(STREAMVAL(car(sl)),Segment)->duration / divide_factor;
            }
        }
    }
  }
```

/* END OF THE NEW FUNCTION */

Appendix C

The SKEL_to_PhonoWord algorithm

C.1 The algorithm

```
/* author : < Tony HEBERT > xtony@itl.atr.co.jp (-> 97/01/31)          */
/* purpose: mainly, to predict pauses                                  */
/* input: fname.parse (or stdin)                                       */
/* output: PhonoWord Utterance, including pauses, focus and reduction */

#include <stdio.h>
#include <string.h>

struct misc {
  char text[256];
  char type[256];
  char head[5];
  int  rank;
};

struct word {
  char text[256];
  int  ob;
  int  cb;
  char cat[16][256];
  int  status;
  int  focus;
};

/* ob : number of opening brackets before the word */
/* cb : ..........closing..........after ......... */

/* status :                                                            */
       /*  1 if at the head of a group under construction              */
       /* -1 if atomic group (one word like ",please")                 */
```

```
      /*  0 if part of a group (but not at its head)                        */
      /*  2 if at the head of a group whose construction has been achieved */
      /* -2 if at the tail of a group whose construction has been achieved */
      /*  3 if punctuation (",", "?",...)                                   */

/* category : v,p...->Ezra BLACK's classification      */
/* focus : if last content word of the phrase 1 else 0 */

/* meaning of "group": I want to split a sentence into meaningful groups   */
/* (=phrases) to predict pauses                                             */



/*                              *** display ***                             */
/*                                                                          */
/* displays wordtab in a PhonoWord way                                      */


void display (struct word wordtab[], int wrd_ctr)
{
  int  i,j,k,w;
  char punct[][16]={",",";",".","?","!","--","(",")"};
  int  ob,cb;
  int  itsapunct;

 ob=cb=0;

 /*for (w=0;w<=wrd_ctr;w++)
       {
 printf ("%d\t%s\t%d\t%d\t",w,wordtab[w].text,wordtab[w].ob,wordtab[w].cb);
 ob+=wordtab[w].ob;
 cb+=wordtab[w].cb;
 for (i=1;i<16;i++)
   {
     printf ("%s ",wordtab[w].cat[i]);
     if (strcmp(wordtab[w].cat[i],"*")==0)
       break;
   }
 printf ("\t%d\n",wordtab[w].status);
       }

    printf ("ob : %d ; cb : %d\n",ob,cb);
    printf ("\n");*/

  if (wrd_ctr!=1)
    {
      printf ("(Utterance PhonoWord\n\t(:D ()\n\t\t(:S ()\n");
      for (w=1;w<wrd_ctr;w++)
```

```
{
  if ((wordtab[w].status==2)||(wordtab[w].status==-1))
    printf ("\t\t\t(:C ()\n");
  printf ("\t\t\t\t(%s",wordtab[w].text);
  if (wordtab[w].focus)
    printf (" (H*)");
  if (w==(wrd_ctr-1))
    printf (")))))\n");
  else if ((wordtab[w+1].status==2)||(wordtab[w+1].status==-1))
    printf ("))\n");
  else
    printf (")\n");
}
      printf ("\n");
    }

/* this was for comparison between this algorithm pause prediction and */
/* chatr pause prediction                                               */

/*  if (wrd_ctr!=1)
      {
for (w=1;w<wrd_ctr;w++)
  {
    j=0;
    if ((wordtab[w].status==2)||(wordtab[w].status==-1))
      {
printf ("#");
j++;
      }
    itsapunct=0;
    for (i=0;i<(sizeof(punct)/16);i++)
      if (strcmp(wordtab[w].text,punct[i])==0)
{
  itsapunct=1;
  break;
}
    if (!itsapunct)
      {
if (wordtab[w].focus)
  {
    printf ("++");
    j++;
    j++;
  }
for (k=1;k<=6-j;k++)
  printf (" ");
printf ("%s\n",wordtab[w].text);
      }
```

```
  }
      }*/
}



/*                          *** reduction ***                          */
/*                                                                     */
/* Reduces non-important words.                                        */

void reduction (struct word wordtab[], int wrd_ctr)
{
  char vowels[]={'a','e','i','o','u','y'};
  int w, i;

  for (w=1;w<(wrd_ctr-1);w++)
    {
      if ((strcmp(wordtab[w].text,"the")==0)||(strcmp(wordtab[w].text,"The")==0))
for (i=0;i!=6;i++) /* checks if the following word starts with a vowel or not */
  if (wordtab[w+1].text[0]==vowels[i])
    {
      strcpy(wordtab[w].text,"thi"); /* if it's the case the becomes thi */
      break;
    }
      if ((w!=1) && (wordtab[w].focus!=1)) /* probably some exceptions */
{
  if (strcmp(wordtab[w].text,"have")==0)
      strcpy(wordtab[w].text,"v");
  if (strcmp(wordtab[w].text,"has")==0)
    strcpy(wordtab[w].text,"s");
  if (strcmp(wordtab[w].text,"are")==0)
    strcpy(wordtab[w].text,"r");
  if (strcmp(wordtab[w].text,"is")==0)
    strcpy(wordtab[w].text,"s");
  if (strcmp(wordtab[w].text,"am")==0)
    strcpy(wordtab[w].text,"m");
  if (strcmp(wordtab[w].text,"can")==0)
    strcpy(wordtab[w].text,"c'n");
  if (strcmp(wordtab[w].text,"will")==0)
    strcpy(wordtab[w].text,"l");
  if (strcmp(wordtab[w].text,"would")==0)
    strcpy(wordtab[w].text,"d");
  if (strcmp(wordtab[w].text,"had")==0)
    strcpy(wordtab[w].text,"d");
}
    }
}
```

```
/*                              *** pausing ***                          */
/*                                                                       */
/* By using the information given by the bracketing and by the grammatical */
/* tagging, it tries to break the sentence into phrases.                 */
/* To achieve this goal, it takes each word one by one, beginning by the  */
/* last one, and tries to determine if it can be linked with the word just */
/* after it. This is the beginning of the construction of groups. After  */
/* this it goes on, takes each group one by one, beginning by the last one */
/* under construction, and tries to determine if it can be linked with the */
/* following group. In fact at the beginning of this procedure, words are */
/* just considered as one word long groups, still under construction.    */
/* For each group under construction this procedure tries to determine if */
/* is time or not to stop its construction.                              */
/* When there is no more group under construction the procedure stops.   */

void pausing (struct word wordtab[], int *wrd_ctr)
{
   int  remaining_wrd_ctr, w, ww,  current_group_size;
   int  i, j, k,  new_group, inside_a_group;
   char cat[][16]={"cc","i","p"};
   char cat2[][16]={"cc","fa","fc","i","p"};

   int ob,cb;
   int live_lock, previous_remaining_wrd_ctr;

   previous_remaining_wrd_ctr=remaining_wrd_ctr=*wrd_ctr-1;
   live_lock=0;
   for (w=1;w<*wrd_ctr;w++)
     if (wordtab[w].status==3)
       --remaining_wrd_ctr;/* a punctuation symbol is considered as a group */
   /*while ((remaining_wrd_ctr!=0) && (live_lock<10)){*/
   while (remaining_wrd_ctr!=0){
     if (previous_remaining_wrd_ctr==remaining_wrd_ctr)
       live_lock++;
     else live_lock=0;
     previous_remaining_wrd_ctr=remaining_wrd_ctr;
     ob=cb=0;
     /*for (w=0;w<=*wrd_ctr;w++)
       {
ob+=wordtab[w].ob;
cb+=wordtab[w].cb;
printf ("%d\t%s\t%d\t%d\t",w,wordtab[w].text,wordtab[w].ob,wordtab[w].cb);
for (i=1;i<16;i++)
  {
    printf ("%s ",wordtab[w].cat[i]);
    if (strcmp(wordtab[w].cat[i],"*")==0)
      break;
```

```
    }
 printf ("\t%d\n",wordtab[w].status);
       }
      printf ("ob : %d ; cb : %d\n",ob,cb);
      printf ("remaining words : %d",remaining_wrd_ctr);
      printf ("\n");*/
      w=*wrd_ctr-1; /* starts by the last word */
      while (w!=0) {
        /* looks for the beginning of a group under construction */
     if (wordtab[w].status!=1)
   {
     w--;
     continue;
   }

/* the algorithm may be stuck; let's give it a hand */
if (live_lock>=10)
  if ((wordtab[w-1].status==-2) || (wordtab[w-1].status==3))
    {
      ww=w;
      while ((wordtab[++ww].status!=2) && (wordtab[ww].status!=3))
      {
if (wordtab[ww].ob>=wordtab[ww].cb)
  wordtab[w].ob+=wordtab[ww].ob>-wordtab[ww].cb;
else
  {
    wordtab[ww].cb+=wordtab[ww-1].cb-wordtab[ww].ob;
    wordtab[ww-1].cb=0;
  }
wordtab[ww].status=0;
wordtab[ww].ob=0;
      }
    }

current_group_size=1;
while (((w+current_group_size)!=(*wrd_ctr)) &&
(wordtab[w+current_group_size].status==0))
/* then calculates its size */
  current_group_size++;

       /* and now tries to link it with the group following it */

       if ((w!=*wrd_ctr-1) && (wordtab[w+current_group_size].status==1))

 if ((wordtab[w+(current_group_size-1)].cb==0) && (wordtab[w+current_group_size].ob==0))
   {
     wordtab[w+current_group_size].status=0;
```

```
     /*printf ("linked : %s\n",wordtab[w+current_group_size].text);*/

     while ((wordtab[w+current_group_size].status==0) &&
((w+current_group_size)!=*wrd_ctr))
/* if it did it, calcultates the new current size of  */
       current_group_size++; /* this group under construction       */
   }

       /* eliminates non-useful brackets */

       if (wordtab[w].ob>wordtab[w+(current_group_size-1)].cb)
 {
   wordtab[w].ob-=wordtab[w+(current_group_size-1)].cb;
   wordtab[w+(current_group_size-1)].cb=0;
 }
       else
 {
   wordtab[w+(current_group_size-1)].cb-=wordtab[w].ob;
   wordtab[w].ob=0;
 }

       /* decides if it's time to achieve the construction of the current */
       /* group; needs more accurate rules                                */

       new_group=0;

       /* when it's a relative clause or when the size is too large        */

for (i=1;i<16;i++)
   {
     if (((strcmp(wordtab[w].cat[i],"fr")==0) ||
(strcmp(wordtab[w].cat[i],"fn")==0)) && (wordtab[w+1].status==0))
 {
   new_group=1;
   break;
 }
     if (current_group_size>=6)
       {
 if ((wordtab[w-1].status==-2) || (wordtab[w-1].status==3))
   {
     new_group=1;
     break;
   }
       else
 for (j=0;j<(sizeof(cat)/16);j++)
     if (strcmp(wordtab[w].cat[i],cat[j])==0)
       {
 new_group=1;
```

```
 break;
       }
       }
     if (strcmp(wordtab[w].cat[i],"*")==0)
       break;
   }

       /* don't want to cut just after a verb */

       for (i=1;i<16;i++)
 {
   if (strcmp(wordtab[w].cat[i],"*")==0)
   break;
   if ((w!=1) && (strcmp(wordtab[w].cat[i],"nbar")==0))
     for (j=1;j<16;j++)
       {
 if (strcmp(wordtab[w].cat[j],"*")==0)
   break;
 if (strcmp(wordtab[w-1].cat[j],"v")==0)
   {
     new_group=0;
     break;
   }
       }
 }

       /* becomes a built group if between two built groups */

       if (((wordtab[w-1].status==-2) || (wordtab[w-1].status==3))
&& ((wordtab[w+current_group_size].status==2)
|| (wordtab[w+current_group_size].status==3)))
 new_group=1;


       if (new_group)
     {
       /*printf ("new group for : %s\n",wordtab[w].text);*/
       if (current_group_size==1) /* a one word long group */
 wordtab[w].status=-1;
       else
 { /* a two or more words long group */
   wordtab[w].status=2; /* tags its head and */
   wordtab[w+(current_group_size-1)].status=-2; /* its tail */
       }

       /* I want a built group to be surrounded by only one opening */
       /* bracket and one closing bracket; the remaining brackets   */
       /* have to be moved around the appropriate group under       */
```

```
        /* construction or till START (wordtab[0]) or END (wordtab[  */
        /* wrd_ctr]).                                                */

        inside_a_group=1;
        if (wordtab[w+(current_group_size-1)].cb>wordtab[w].ob)
 {
   ww=w-1;
   while ((ww!=0) && (inside_a_group))
     {
       if (((wordtab[ww].status==1) ||
(wordtab[ww].status==0))
&& ((wordtab[ww+1].status==2) || (wordtab[ww+1].status==3)
|| (wordtab[ww+1].status==-1)))
 inside_a_group=0;
       else --ww;
     }
   wordtab[ww].cb+= wordtab[w+(current_group_size-1)].cb-wordtab[w].ob;
 }
       else
 {
   ww=w+1;
   while ((ww!=*wrd_ctr) && (wordtab[ww].status!=1))
     ++ww;
   wordtab[ww].ob+=wordtab[w].ob-wordtab[w+(current_group_size-1)].cb;
 }
       wordtab[w].ob=wordtab[w+(current_group_size-1)].cb=1;
       remaining_wrd_ctr-=current_group_size;
     }

/* the size of a group can be too big; let's cut it into 2 parts */

if ((new_group) && (current_group_size>=8))
  {
    new_group=0;
    for (i=(w+current_group_size/2+1+(current_group_size % 2));
i>=(w+current_group_size/2-2);i--)
      {
/*printf ("%s : \n",wordtab[i].text);*/
for (j=1;j<16;j++)
  {
    if (strcmp(wordtab[i].cat[j],"*")==0)
      break;
    for (k=0;k<(sizeof(cat2)/16);k++)
      if (strcmp(wordtab[i].cat[j],cat2[k])==0)
{
  /*printf ("cat : %s\n",cat2[k]);*/
  new_group=1;
  for (j=1;j<16;j++)
```

```
    {
      if (strcmp(wordtab[i-1].cat[j],"*")==0)
break;
      if (strcmp(wordtab[i-1].cat[j],"d")==0)
{
  i--;  /* if there is a determiner before */
  break;/* it has to be incorporated.      */
}
    }
  wordtab[i-1].status=-2;
  wordtab[i-1].cb=1;
  wordtab[i].status=2;
  wordtab[i].ob=1;
  break;
}
    if (new_group) break;
  }
if (new_group) break;
}
  }
w--; /* OK, next ! */
      }
   }
   /*if (live_lock>=10)
     {
       printf("***\n\n");
       printf ("THE PAUSING ALGORITHM HAS NOT TERMINATED\n");
       printf("\n\n***\n");
       *wrd_ctr=1;
     }*/
 }

void intonation (struct word wordtab[], int wrd_ctr)
{
    int i, w, last_content_word;

    w=1;
    last_content_word=0;
    while (w!=wrd_ctr)
      {
for (i=1;i<16;i++)
  {
    if (strcmp(wordtab[w].cat[i],"*")==0)
      break;
    if (strcmp(wordtab[w].cat[i],"n")==0)
      last_content_word=w;
    if (strcmp(wordtab[w].cat[i],"multiword")==0)
      {
```

```
if (wordtab[w].status==-2)
  wordtab[last_content_word].focus=1;
++w;
while ((w!=wrd_ctr) && (strcmp(wordtab[w].cat[1],"*")==0))
  {
    if (wordtab[w].status==-2)
      wordtab[w].focus=1;
    last_content_word=w;
    ++w;
  }
w--;
      }
  }
if (wordtab[w].status==-2)
  wordtab[last_content_word].focus=1;
if (w!=wrd_ctr)
  w++;
      }
  }
```

```
/*                          *** SKEL_to_PhonoWord ***                        */
/*                                                                            */
/*                                                                            */
/* Fills an array made to contain the different words of a sentence, and     */
/* for each word, the number of opening brackets before, the number of       */
/* closing brackets after, the category (classication), the status (except */
/* for punctuation it will be set to 1 at the beginning) and a focus flag    */
/* (to mark focused words).                                                   */
/* Then for each filled array, applies Reduction, Pausing and intonation.    */
/* Ends by displaying the result, through a PhonoWord output                  */

void SKEL_to_PhonoWord (FILE *fp)
{
    int  wrd_ctr;
    struct word wordtab[100];
    int  i,j,w;
    int  lbl_ctr, wrd_found, interesting;
    char c;
    int  label;

    int  number_of_newlines;
    char current_label[256];
    char punct[][16]={",",";",".","?","!","--","(",")"};

    char rep;
```

```
    number_of_newlines=0;
    i=wrd_found=lbl_ctr=0;
    strcpy(wordtab[0].text,"START");                /* not really */
    wordtab[0].ob=wordtab[0].cb=wordtab[0].focus=0;  /* important  */
    strcpy(wordtab[0].cat[1],"*");
    wordtab[0].status=-2; /* important : to do as if there was an already */
                          /* built group before the 1st word.             */
    wordtab[1].ob=wordtab[1].cb=wordtab[1].focus=0;
    strcpy(wordtab[1].cat[1],"start");
    wrd_ctr=1; /* words counter, also counts punctuation */
    while (((c=getc(fp))!='[') && (c!=EOF) && (number_of_newlines<50))
      {
if (c=='\n')
  number_of_newlines++;
if (isalpha(c))
  number_of_newlines=0;
      }
    while ((c!=EOF) && (number_of_newlines<50)) {
      /*printf ("c : %c\n",c);*/
      switch(c){
  case '[': /* label start */
    /*printf("case [ -> %c\n",c);*/
    number_of_newlines=0;
    i=0;
    wordtab[wrd_ctr].ob++;
    lbl_ctr++;
    while ((c!='\n') && (c!=' '))
      {
c=getc(fp);
/*printf ("%c\n",c);*/
if(isalpha(c))
  wordtab[wrd_ctr].cat[lbl_ctr][i++] = c;
else
  while ((c!='\n') && (c!=' '))
    {
      c=getc(fp);
      /*printf ("%c\n",c);*/
    }
      }
    /*printf ("%c\n",c);*/
    wordtab[wrd_ctr].cat[lbl_ctr][i]='\0';
    /*printf ("label : %s\n",wordtab[wrd_ctr].cat[lbl_ctr]);*/
    while ((c=='\n') || (c==' '))
      {
c=getc(fp);
/*printf ("%c\n",c);*/
      }
    label=1;
```

```
    /*printf("fin cas [ -> %c\n",c);*/
    break;

  case ']': /* label end */
    /*printf("cas ] -> %c\n",c);*/
    number_of_newlines=0;
    if (strcmp(current_label,"start")!=0)
      while (((c=getc(fp))==' ') || (c=='\n'));
    else
      {
strcpy(wordtab[wrd_ctr].text,"END");
strcpy(wordtab[wrd_ctr].cat[1],"*");
wordtab[wrd_ctr].ob=wordtab[wrd_ctr].cb=wordtab[wrd_ctr].focus=0;
wordtab[wrd_ctr].status=2;  /* to do as if              */
                            /* there was a built group */
                            /* after the last word.     */

/* thanks to the following loop, all the brackets       */
/* surrounding a  punctuation symbol will be transfered */
/* either to the previous word or to the following one. */

for (w=0;w<wrd_ctr;w++)
  if (wordtab[w].status==3) /* it's a "," , "?",... */
    {
      if (wordtab[w].cb>wordtab[w].ob)
wordtab[w-1].cb+=(wordtab[w].cb-wordtab[w].ob);
      else wordtab[w+1].ob+=(wordtab[w].ob-wordtab[w].cb);
      wordtab[w].cb=wordtab[w].ob=0;
    }
reduction (wordtab,wrd_ctr);
pausing (wordtab,&wrd_ctr);
intonation (wordtab,wrd_ctr);
display (wordtab,wrd_ctr);
i=wrd_found=0;
lbl_ctr=1;
wordtab[1].cb=wordtab[1].focus=0;
wordtab[1].ob=1;
strcpy(wordtab[1].cat[1],"start");
wrd_ctr=1;
strcpy(current_label,"******");
while ((strcmp(current_label,"start")!=0) && (c!=EOF) && (number_of_newlines<50))
  {
    c=getc(fp);
    /*printf ("entre deux starts; c : %c\n",c);*/
    if (c=='\n')
      number_of_newlines++;
    if (isalpha(c))
      number_of_newlines=0;
```

```
    if (c=='[')
      {
for (j=0;j<5;j++)
  {
    c=getc(fp);
    if (c==EOF)
      break;
    current_label[j]=c;
  }
current_label[5]='\0';
      }
  }
if (strcmp(current_label,"start")==0)
  {
    strcpy(current_label,"******");
    while (((c=getc(fp))==' ') || (c=='\n'));
  }
      }
    /*printf("end case ] -> %c\n",c);*/
    break;

  default:
    /*printf("default -> ");*/
    i=0;
    number_of_newlines=0;
    while ((c!=']') && (c!='['))
      {
/*printf("%c\n",c);*/
wordtab[wrd_ctr].text[i]=current_label[i++]=c;
c=getc(fp);
/*printf("%c\n",c);*/
if (c=='_')
  {
    wordtab[wrd_ctr].text[i]='\0';
    strcpy(wordtab[wrd_ctr].cat[++lbl_ctr],"*");
    /*printf ("word : %s\n",wordtab[wrd_ctr].text);*/
    wordtab[wrd_ctr].status=1;
    for (j=0;j<(sizeof(punct)/16);j++)
if (strcmp(wordtab[wrd_ctr].text,punct[j])==0)
  {
    wordtab[wrd_ctr].status=3;
    break;
  }
    lbl_ctr=0;
    wrd_ctr++;
    wordtab[wrd_ctr].ob=wordtab[wrd_ctr].cb=wordtab[wrd_ctr].focus=0;
    i=0;
    while ((c!=' ') && (c!='\n'))
```

```
      c=getc(fp);
    while ((c==' ') || (c=='\n'))
      c=getc(fp);
  }
if (c==']')
  {
    current_label[i]='\0';
    /*printf("current label : %s\n",current_label);*/
    wordtab[wrd_ctr-1].cb++;
  }
      }
    /*printf("end default -> %c\n",c);*/
    break;
  }
      /*printf("end switch\n");*/
    }

  }


main(int argc,char **argv)
{
    FILE *fn;
    char *filename;

    if (argc==1)
SKEL_to_PhonoWord(stdin);
    else
if (argc==2){
    if( (fn=fopen(argv[1],"r")) == NULL) {
fprintf(stderr,"Unable to open parse file %s.\n",argv[1]);
exit(-1);
    }
    SKEL_to_PhonoWord(fn);
}
else
{
    fprintf(stderr,"Call with %s fname.parse\n",argv[0]);
    exit;
}
}
```

C.2 Meanings of the nonterminal labels used in the treebank

```
becomp  Conjoined Phrasal Object of a "be" Verb, Non-Traditional,
        e.g. <He was> pissed off and in a rage
cc  Coordinating Conjunctive Phrase,
        e.g. <We like baseball, football> , and <tennis.>
coord  Coordinate Constituent (of any type),
        e.g. Chocolate is good but ice cream is better <.>
d  Determiner Phrase,
        e.g. All the <food was eaten.>
dqzero  Multiword Interrogative Detemriner "no matter",
        e.g. <We can meet on> no matter <what topic.>
fa  Adverbial Clause,
        e.g. <We wept> because it was all so sad <.>
fabar  Adverbial Clause (Pre- or Post-Modified) ,
        e.g. <We wept> precisely because it was all so sad <.>
fc  Comparative Phrase,
        e.g. <Joe is taller> than Bill <.>
fn  Nominal Clause,
        e.g. <He got> what he said he wanted <.>
fr  Relative Clause,
        e.g. <The boy> that you saw last night <is Joe's little brother.>
g  Possessive Phrase,
        e.g. <I liked> that guy's <ideas.>
i  Interrupter,
        e.g. <The girl> in the blue velvet band
ibbar  Interrupter Phrase (Beginning),
        e.g. To tell the truth, <I don't care.>
iebar  Interrupter Phrase (Ending),
        e.g. <I don't care> , to tell the truth <.>
iibar1  Interrupter Phrase (Intermediate),
        e.g. <I don't care> , to tell the truth , <what the hell you do.>
j  Adjective Phrase,
        e.g. <The> very big <book>; <the book is> very big <.>
jzero  Multiword Adjective,
        e.g. <A> home loving <man>; <a> stone dead <racoon>
l  Defective Phrase,
        e.g. <With> Bill out of town <, we'd better postpone the picnic.>;
             John eats in restaurants and <Mary at home.>
m  Number Phrase,
        e.g. <Give me> six to ten <tickets, please.>
messcomp Conjoined Message Phrase;
        e.g. <He told me> that a letter had come and from whom it had come <.>
multiword Multiword Lexical Unit,
        e.g. <Come at> six fifty-five; <We paid> five hundred pounds;
             <the> United States <is far from here>
```

n Noun Group,
e.g. <The> bank account <is not registered to Sam.>
nabarq Modified Noun Phrase, Pronominal,
e.g. Something so good <must be fattening, sinful or expensive.>
nallbarq1 Modified Noun Phrase, General,
e.g. <We accorded Bill> very precisely a forty-nine-percent share in the business<.>
nbar Noun Phrase,
e.g. <I see> my old friend <over there.>
nbarq Modified Noun Phrase,
e.g. <I see> my old friend from years ago <over there.>
nqbarq1 Modified Noun Phrase, Interrogative,
e.g. <Bill's relatives,> a great number of whom <were present, were all tall people.>
nr Adverbial Noun Group,
e.g. <Next> Wednesday morning <, let's meet, ok?>
o Auxilliary Phrase,
e.g. <Bill> might <be there.>; <Bill> would not have been <doing something like that.>
p Prepositional Phrase,
e.g. <The hole> in the wall <needs repairing.>
pr Modified Prepositional Phrase,
e.g. <The hole> right in this wall <needs repairing.>
prepcomp Coordinated Object Of Preposition, Non-Traditional,
e.g. <You can find out about> maintenance and obtaining service contracts
pzero Conjoined Preposition,
e.g. <It's either> under or above <the picture.>
quo Quoted or Parenthesized Constituent,
e.g. <He spoke of a> "problem" <to solve.>; <The solution> (to the problem) <seemed clear.>
r Adverb Phrase,
e.g. <A> really clearly <mistaken idea>; <His budget lasted> far longer than he had anticipated <.>
rand Randomly-Occurring Constituent (Viz. Noise),
e.g. <I thought> , <that I'd win.>; <He is actually> * <a member of the group.>
rqzero Multiword Interrogative Adverb "no matter",
e.g. No matter <when you want to meet, it's fine with me.>
rzero Complex Adverb (Adverb followed by quoted paraphrase)
e.g. <We> fortuitously (accidentally) <ran into Bill.>
sc Imperative Sentence,
e.g. Go to the Devil <!>
sd Declarative Sentence,
e.g. Bill is here <.>
si Interrupter Sentence,
e.g. <Bill is here,> you know <.>; <"Bill is here,"> said Tim <.>;
sprime Sentence With Or Without Modification,

```
        e.g. In fact, Bill is here, as everyone knows <.>; Bill is here,
             in fact <.>
sprpd  Sentence With Terminal Punctuation, If Any,
        e.g. In fact, Bill is here, as everyone knows .; Bill is here, in fact
start  Start Symbol of grammar
swh  Interrogative Setnece,
        e.g. Is Bill here <?>; What is your name, please <?>
t  Non-Finite Clause Used Nominally,
        e.g.  To sing well <was his ambition>; Singing well <is tough>;
              <Pages> not included <are 5-9.>
uhzero  Multiword Interjections,
        e.g. Gee whiz
v  Basic Verb Phrase,
        e.g. <Bill> saw Sally <.>; <Bill> told Sam to leave
vbar  Complex Verb Phrase,
        e.g. <Bill> didn't see Sally <.>; <Bill> has told Sam to leave;
             <Bill> saw Sally <.>(<--here same as v)
vr  Basic Verb Phrase, Modified,
        e.g. <Bill> recently saw Sally <.>; <Bill> told Sam to leave once again
vrbar  Complex Verb Phrase, Modified,
        e.g. <Bill> obviously didn't see Sally <.>; <Bill> has told Sam to
             leave once again
vzero  Multiword Verb or Complex Verb (Verb followed by quoted paraphrase),
        e.g. <to> cold boot <the system>; <We> laughed (chortled)
```

C.3 The comparison with the pause prediction by CHATR

The sentences used for the comparison are in the following files :

/DB/SBNLP/data/lanctb/orig/baa304.orig
/DB/SBNLP/data/lanctb/orig/baa305.orig
/DB/SBNLP/data/lanctb/orig/baa308.orig
/DB/SBNLP/data/lanctb/orig/baa405.orig
/DB/SBNLP/data/lanctb/orig/baa393.orig

The labelling of the waveforms (produced by the natural speaker) is in :

~shimoda/TONY/LBL/

Here are the explanations given by Shimoda-san about how the labelling has been made :

```
Speech Files :   /dept2/work21/temp/nick90/wav/US***.d
Label Files  :   ~shimoda/TONY/LBL/US***.lab    (words)
                                        .breaks (breakindex)
                                        .aprom  (prominence)
                                        .bprom  (prominence)
                                        .cprom  (prominence)
note ;

<<words>>
These labelling was made by auto, so there are error (lag) sometimes.
I have corrected major error only because to correct minor error take
a lot of time.  Please understand.

<<breakindex>>
Nick-san told me you need BI3 and BI4 mostly.  I have labelled BI2
sometimes but I'm not sure about it.  The judgement of BI2 is
difficult and it may make a difference depends on person. Also they
need a lot of time to think.  If BI2 is useful for you, please refer
it.

        BI4     ....    end of sentence
        BI3     ....    before pause (#)
        BI2     ....    reset of intonational phrase

        {exception}
        ★ [ BI4 before pause (#) ]
                ------> if there is a big pause in one long sentence

<<prominence>>
I have labelled three kinds of prominence.
```

```
P        ....    strongest prominence word of sentence
(.aprom)

pr       ....    prominence word of phrase (between BI)
(.bprom)

wpr      ....    more weak(?) prominence than "pr" (above),
(.cprom)           felt something (like a prominence)

{exception}
★ [ more than one "P","pr","wpr" in one sentence or phrase ]
        ------> couldn't choose one prominence (same power)

★ [ no prominence ]
        ------> didn't feel any prominence
```

Here is now the shell script I've used to make the comparison :

```
#!/bin/csh -f
# baa304 <-> (US001 -> US005)
# baa305 <-> (US080 -> US116)
# baa306 <-> ???
# baa307 <-> ???
# baa308 <-> (US119 -> US156) (US017 in fact but it creates problems)
# baa405 <-> (US058 -> US079)
# baa393 <-> (US006 -> US057)

# use : bash$ compare US001 US005 baa304

set shimoda = ~shimoda/TONY/LBL/
set xtony = ~xtony/LBLSPECIAL/
set skel = /DB/SBNLP/data/lanctb/skel/
set filename1 = $1
set filename2 = $2


#@ i=1
#@ limit = $#argv / 2 + 2
#while ($i != $limit)
#set filename1 = $argv[$i]
#@ i++
#set filename2 = $argv[$i]
#ls $shimoda | awk '$1 >= "'$filename1'" && $1 <=
"'$filename2'.z" && ($1 ~ /lab$/
|| $1 ~ /bprom$/ ) {printf ("%s%s\n","'$shimoda'",$1)}' >> tmp1
#@ i++
#end
#cat tmp1

# get the list of the US***.lab and US***.bprom files matching with the
# third argument of "compare". For "compare US001 US005 baa304" the result
# is : US001.bprom US001.lab US002.bprom US002.lab US003.bprom US003.lab
# US004.bprom US004.lab US005.bprom US005.lab
# US***.aprom files are included in US***.bprom files. That's why I'm
# not using them directly.

ls $shimoda | awk '$1 >= "'$filename1'" && $1 <= "'$filename2'.z" &&
($1 ~ /lab$/
|| $1 ~ /bprom$/ ) && $1 !~ /^US009/ {printf ("%s%s\n","'$shimoda'",$1)}' > tmp1

ls $xtony | awk '$1 >= "'$filename1'" && $1 <= "'$filename2'.z" && ($1
~ /lab$/
|| $1 ~ /bprom$/ ) {printf ("%s%s\n","'$xtony'",$1)}' >> tmp1
```

```
sort tmp1 > tmp3

# save the content of the previous files into tmp2

cat `cat tmp3` > tmp2

# cleans it

awk '($2 == "121" || $2 == "76") && $3 != ""' tmp2 | sort > tmp1

awk '{ if ($3 == "#") {printf ("#")} else {if ($3 =="pr") {printf
("++")}
else {printf (" %s\n",$3)}}}' tmp1 > tmp2

# the natural utterances are now ready to be compared with the results
# of the SKEL_to_PhonoWord algorithm and of CHATR

#set n = `wc -l tmp2` creates an error in awk
set n = `awk 'END {print NR}' tmp2`

# applies the SKEL_to_PhonoWord algorithm on the sentences matching with
# the natural waveforms

/export/atrh17/xtony/Chatr/skel/SKEL_to_PhonoWord $skel$argv[$#argv].parse > tmp3

# the output of the SKEL_to_PhonoWord algorithm has been modified to
# allow the comparison with natural utterances and CHATR
# these modifications are commented out in the "display" function
# of SKEL_to_PhonoWord.c

cat tmp2 tmp3 > tmp1

# to allow the comparison with the natural utterances and with the
# results of the SKEL_to_PhonoWord algorithm, I've made CHATR give
# a special output, by adding the following piece of code at the
# end of the "hlp_module" function in hlp.c :
#
#for (word=WORDSTREAM(utt);word != SNIL;word=SC_next(word))
#      {
# if (hlp_has_feat(word,F_PHRASELEVEL,":C"))
#   printf ("#");
# if ((hlp_has_feat(word,"NAccent","+")) ||
(hlp_has_feat(word,"NAccent","++"))
|| (hlp_has_feat(word,"HAccent","+")) || (hlp_has_feat(word,"HAccent","++")))
#   printf ("++");
# printf (" ");
# printf
("%s\n%s\n%s\n",SC(word,Word)->text,pprint(SC(word,Word)->features),
```

```
pprint(SC(word,Word)->intones));
#       }

cat tmp2 ~xtony/$argv[$#argv].tts > tmp3

# comparison between the natural utterances and the results of
# SKEL_to_PhonoWord algorithm

echo "SKEL"
awk -f awk1 n=$n start=10000 tmp1

# comparison between the natural utterances and the results of CHATR

echo " "
echo "CHATR"
awk -f awk1 n=$n start=10000 tmp3


rm tmp1 tmp2 tmp3
```

It needs the following "awk1" file :

```
{
# last processing before comparison

if (NF == 2)
  {
    spec[NR]=$1;
    word[NR]=$2;
    if ((spec[NR]=="##") || (spec[NR]=="#"))
      {spec1[NR]="#"}
    if (spec[NR]=="++")
      {spec2[NR]="++"}
    if ((spec[NR]=="#++") || (spec[NR]=="##++"))
      {
       spec1[NR]="#";
       spec2[NR]="++";
      }
  }
else {word[NR]=$1};
if (NR >= n)
  {
     if ((word[NR] == word[1]) && (NR<start))
      {start = NR;print start}
  }
}

END {
print NR;
for (i=1;i<start;i++)
  {

# it happens (often) that words from the natural utterances and
# words from the results of CHATR or of the SKEL_to_PhonoWord algorithm
# don't match exactly. The following lines help to find the next match

    if (i<(start-10) && (word[i]!=word[start+i-1]))
      {
        #printf ("problem with : %d %s %d %s\n",i,word[i],start+i-1,word[start+i-1]);
        OK=0;
        j=start+i-1;
k=i+1;
        while  ((j<=start+i+29) && (word[j]!=word[k]))
          {
            k++;
            if (k==i+11)
              {
```

```
                j++;
                k=i+1;
              }
            }
        if (word[j]==word[k])
          {
            #printf ("it's good\n");
            OK=1;
            #print j,word[j],k,word[k];
          }
    if (OK==1)
      {
#print i,j,k;
         if ((j-start-i)>0)
          {
            for (l=start+i;l<=NR-(j-start-i);l++)
              {
                if ((spec1[l]=spec1[l+(j-start-i)]) == 0)
                  {spec1[l]=""};
                if ((spec2[l]=spec2[l+(j-start-i)]) == 0)
                  {spec2[l]=""};
                word[l]=word[l+(j-start-i)];
                #print l,spec1[l],spec2[l],word[l];
              }
            NR-=(j-start-i);
          }
         #printf ("New NR : %d\n",NR);
         if (j == start+i-1)
           {offset=0}
         else
           {offset=1};
         for (m=i+offset;m<=NR-(k-i-1);m++)
           {
             if ((spec1[m]=spec1[m+k-i-offset]) == 0)
               {spec1[m]=""};
             if ((spec2[m]=spec2[m+k-i-offset]) == 0)
               {spec2[m]=""};
             word[m]=word[m+k-i-offset];
             #print m,spec1[m],spec2[m],word[m];
           }
         NR-=k-i-offset;
         start-=k-i-offset;
         #printf ("New final NR : %d New start : %d\n",NR,start);
       }
    else
     {
       break;
     };
```

```
}
total++;
printf("%s",spec1[i]);
if (spec1[i]!="")
  {printf(" ");}
else
  {printf("  ")};
printf("%s",spec2[i]);
if (spec2[i]!="")
  {printf("    ");}
else
  {printf("      ")};
printf("%s",word[i]);
for (s=1;s<=20-length(word[i]);s++)
  {printf(" ")};
if ((spec1[i]!=spec1[start+i-1]) || (spec2[i]!=spec2[start+i-1]))
  {printf("MISMATCH")}
else
  {printf("        ")};

# tests if pauses have been inserted correctly

if (spec1[i]!=spec1[start+i-1])
  {errorbreak++}
else
  {successbreak++}

# tests if prominence have been inserted correctly

if (spec2[i]!=spec2[start+i-1])
  {errorprom++}
else
  {successprom++}

for (s=1;s<=10;s++)
  {printf(" ")};
printf("%s",spec1[start+i-1]);
if (spec1[start+i-1]!="")
  {printf(" ")}
else
  {printf("  ")};
printf("%s",spec2[start+i-1]);
if (spec2[start+i-1]!="")
  {printf("    ")}
else
  {printf("      ")};
printf("%s\n",word[start+i-1]);
#printf("%s  %s  %s%s\n",spec[i],word[i],spec[start+i-1],word[start+i-1]);
```

```
  }
printf ("Breaks : Errors : %d\tSuccess : %d\tRate :
   %%d%\n",errorbreak,successbreak,
successbreak*100/total);
printf ("Prominence : Errors : %d\tSuccess : %d\tRate :
       %%d%\n",errorprom,successprom,
successprom*100/total);
}
```

The results of the comparison are :

baa304 <-> (US001 -> US005)

Breaks :

	Errors	Success	Rate
SKEL	27	214	88%
CHATR	64	201	75%

Prominence :

	Errors	Success	Rate
SKEL	56	185	76%
CHATR	106	159	60%

baa305 <-> (US080 -> US116)

Breaks :

	Errors	Success	Rate
SKEL	66	349	84%
CHATR	141	453	76%

Prominence :

	Errors	Success	Rate
SKEL	107	308	74%
CHATR	235	359	60%

baa308 <-> (US119 -> US156) (US017 in fact but creates problems)

Breaks :

	Errors	Success	Rate
SKEL	42	223	84%
CHATR	285	926	76%

Prominence :

	Errors	Success	Rate
SKEL	69	196	73%
CHATR	477	734	60%

baa405 <-> (US058 -> US079) cannot be used : too many unparsed parts in baa405.parse

Breaks :

	Errors	Success	Rate
SKEL	-	-	-
CHATR	246	717	74%

Prominence :

	Errors	Success	Rate
SKEL	-	-	-
CHATR	399	564	58%

baa393 <-> (US006 -> US057)

Breaks :

	Errors	Success	Rate
SKEL	43	372	89%
CHATR	202	619	75%

Prominence :

	Errors	Success	Rate
SKEL	109	306	73%
CHATR	375	446	54%

(most of the time the comparison with CHATR has been made on more words than with SKEL)
(it would surely be useful later to know what kind of mistakes it is)

Index

S

T

U

V

W