非公開

TR-I-0140

複合音声単位を用いる規則合成実験システム (付録 プログラムリスト)

Speech Synthesis System Using Non-Uniform Units.

武田 一哉　安部勝雄　海木延佳　匂坂 芳典
Kazuya TAKEDA, Katsuo ABE, Nobuyoshi KAIKI and
Yoshinori SAGISAKA

1990.2

内容更概

種々の複合音声単位を選択的に用いる規則合成実験システムのプロトタイプを構築した。本報告書はこの実験システム解説書の別冊であり、システムを構成するプログラムのリストを掲載した。

ATR 自動翻訳電話研究所
ATR Interpreting Telephony Research Laboratories

本報告書の内容

種々の複合音声単位を選択的に用いる規則合成実験システムのプロトタイプを構築した。本システムは、ルールによる韻律情報の生成、エキスパートシステムを用いた最適な複合音声単位の選択、及ぶケプストラムパラメータによる素片接続･編集･合成といった処理を統合化したものである。

本稿は、システム解説書の別冊付録であり、全てのプログラムのリストを掲載している。

掲載プログラムの内容

APPENDIX A　システムプログラム

APPENDIX B　ライブラリプログラム

APPENDIX C　単位選択プログラム

Appendix1

システムプログラム

dur_new.c
mkppitch.c

mkuafile.c
mkunit.c
durset.c
mkcep.c
mdpower.c
mkpitch.c
lma_pwr.c

```
/* ========================================================================== *
 | DURATION                                                                   |
 |                                                                            |
 | Predict duration of phonemes for speech synthesis.                         |
 | This routine read xxx.PH file, format of which is defined in the header    |
 | file "/usr4/takeda/SYN/include/Synthesis.h", and predict phoneme's         |
 | duration.                                                                  |
 | The factores used for prediction are phoneme's name, preceding phoneme,    |
 | following phoneme and mora length of the phrase.                           |
 | According to some statistics, duration of the pre-pauseal position phoneme|
 | is lengthed by the factor Pre_pause, and duration of the sentencial ending|
 | phoneme is shoteded by the factor Sentence_end, respectively.              |
 |  This routine write out roman transcription of the synthesizing speech     |
 | and predicted duration of each phoneme onto the xxx.ST and xxx.DR.         |
 |                                                                            |
 * -------------------------------------------------------------------------- *
 | [commands]                                                                 |
 |   duration_gen file-name-header                                            |
 | [compile]                                                                  |
 |   compile commands are written in makefile.duration                        |
 * -------------------------------------------------------------------------- *
 | [History]                                                                  |
 | Originally coded by K.Takeda, ATR. Interpreting Telephony Research Lab.    |
 |                    May 18, 1988( 2nd anversary of Mr. and Mrs. Takeda )    |
 | [Takeda, May 19, 1988 ]                                                    |
 | some bugs concerening with phrase concatinateion are fixed.                |
 | [Takeda.1 May 30,  1988]                                                   |
 | long vowel's processing will be done as follows                            |
 |      input PH file --> O[o:]                                               |
 |              string file --> O[o:]                                         |
 |              duration file -> o,u xxx.x [ms]                               |
 |      input PH file --> E[e:]                                               |
 |              string file --> E[e:]                                         |
 |              duration file -> e,i xxx.x [ms]                               |
 | [Takeda.2 June 8, 1988]                                                    |
 |      bug fixing of pause processing                                        |
 | [Takeda.3 Jul 27, 1988]                                                    |
 |      Majar change for using new dat types                                  |
 * ========================================================================== */
#include "/SYN/include/Synthesis.h"
#include <fcntl.h>

/***** edited by K.Abe '89 Oct.03 ***********************************************/
/*      'Max_phrase' and 'Max_phoneme' are defined in 'Synthesis.h'.            */
/***** edited by K.Abe '89 Oct.03 ***********************************************/
/***** edited by K.Abe '89 Sep.22 ***********************************************/
/* #define Max_phrase   256                                                     */
/* #define Max_phoneme  2048                                                    */
/*******************************************************************************/
/* #define Max_phrase   20                                                      */
/* #define Max_phoneme  128                                                     */
/***** edited by K.Abe '89 Sep.22 ***********************************************/

#define Max_file_name   256

#define Top     0
#define Mid     1
#define Tail    2

#define Pre_pause       1.2             /* Phrasal effect on duration */
#define No_mod          1.0
#define Sentence_end    0.9             /* sentencial effect on duration */

/* #define Pre_pause    1.5             KIN Phrasal effect on duration */
/* #define Sentence_end 0.7             KIN sentencial effect on duration */
```

```
char *prog;

typedef struct  /* data structure for duration prediction */
        {

/***** edited by K.Abe '89 Oct. 16 ************************************************/
          char phn[32];                 /* phoneme name */
          char pre[32];                 /* preceding phoneme name */
          char fol[32];                 /* following phoneme name */
/***** edited by K.Abe '89 Oct. 16 ************************************************/
/*        char phn[10];
/*        char pre[10];
/*        char fol[10];
/***** edited by K.Abe '89 Oct. 16 ************************************************/

          int mora;                     /* mora length of phrase */
          int loc;                      /* location in the phrase */
          int sent;                     /* location in the sentence */
        } DurationInf;

typedef struct
        {

/***** edited by K.Abe '89 Oct. 16 ************************************************/
          char phn[32];                 /* phoneme name */
/***** edited by K.Abe '89 Oct. 16 ************************************************/
/*        char phn[10];
/***** edited by K.Abe '89 Oct. 16 ************************************************/

          float length;                 /* duration in mirisecond */
        } Duration;

main(argc,argv)
int argc;
char *argv[];
{
  Phrase phrase_inf[Max_phrase];                /* phrase information table */
  DurationInf duration_inf[Max_phoneme];        /* data used for duration
                                                   prediction */
  Duration duration[Max_phoneme];               /* out put data */

  FILE *fp_phrase;                      /* file pointers */
  FILE *fp_duration;
  FILE *fp_strings;

  int phrase_size;                      /* size of synthesis phrases */
  int phoneme_size;                     /* size of phonemes */

  char *file_name_header;
  char phrase_file[256], dur_file[256], str_file[256];

  register int i;

                                        /* parse command line */
  prog = argv[0];
  if(argc==1)
  {
    fprintf(stderr,"\t[%s] Generate phoneme duration.\n", prog );
    fprintf(stderr,"\t[Usage]: %s file-name-header.\n", prog );
    exit(0);
  }
  file_name_header = argv[1];

                                        /* create and open out put files */
```

```
  sprintf( dur_file, "%s.DR", file_name_header );
  if(( fp_duration = fopen( dur_file, "w" )) == NULL )
  {
    fprintf( stderr, "%s: duration file can't create or open.\n", prog );
    exit(1);
  }
  sprintf( str_file, "%s.ST", file_name_header );
  if(( fp_strings = fopen( str_file, "w" )) == NULL )
  {
    fprintf( stderr, "%s: strings file can't create or open.\n", prog );
    exit(1);
  }

  /* get phrase information table */
  if(( phrase_size = ReadPhrase( file_name_header, phrase_inf )) <= 0 )
  {
    fprintf( stderr, "%s: phrase file read error.\n", prog );
    exit(1);
  }
  /* parse phrase information table */
  if(( phoneme_size = parse_phrase( phrase_size, phrase_inf, duration_inf ))
        <= 0 )
  {
    fprintf( stderr, "%s: parse phrase error.\n", prog );
    exit(1);
  }
  /* write out to  string file */
  if( make_string( fp_strings, phoneme_size, duration_inf ) < 0 )
  {
    fprintf( stderr, "%s: make_string error.\n", prog );
    exit(1);
  }
  /* duration set  */
  if(( phoneme_size = duration_set( phoneme_size, duration, duration_inf ))
       <= 0 )
  {
    fprintf( stderr, "%s: duration_set error.\n", prog );
    exit(1);
  }
  /* write out to file */
  fprintf( fp_duration, "%5s %7.2f\n", "pau", 100.0 );
  for( i = 0; i < phoneme_size; i++ )
  {
    /* following if block is add for long vowel processing
            added at May 30, 1988 */
    if( strcmp( duration[i].phn, "O" ) == 0 )
    {
      strcpy( duration[i].phn, "o,u" );
    }
    else if( strcmp( duration[i].phn, "E" ) == 0 )
    {
      strcpy( duration[i].phn, "e,i" );
    }
    fprintf( fp_duration, "%5s %7.2f\n", duration[i].phn, duration[i].length );
  }
  fprintf( fp_duration, "  pau %7.2f\n", 100.0 );
  fclose(fp_duration);
}

parse_phrase( phrase_size, phrase_inf, duration_inf )
Phrase phrase_inf[];
DurationInf duration_inf[];
int phrase_size;
{
```

```
/***** edited by K.Abe '89 Oct.03 ************************************************/
  char phonemes[256][10];
/****** edited by K.Abe '89 Oct.03 ***********************************************/
/*  char phonemes[50][10];                                                       */
/***** edited by K.Abe '89 Oct.03 ************************************************/

  char strings[20][1024];

  int mora[20];
  int pause[20];
  int offset = 0;
  int type;
  int phoneme_size;
  register int i, j;

/*
  Concatinate phrases of which pause type equal to 0.
  If the phrase's pause type is 0, then next phrase will be uttered at the
  same time, without pause.
*/

  strcpy( strings[0], phrase_inf[0].string );
  mora[0] = phrase_inf[0].mora;

  for( i = 1, j = 0; i < phrase_size; i++ )
  {
     if( phrase_inf[i-1].pause == 0 )
     {
          mora[j] += phrase_inf[i].mora;
          strcat( strings[j], phrase_inf[i].string );
          pause[j] = phrase_inf[i].pause;
     }
     else
     {
/* the next line is added as [Takeda.2]. */
          pause[j] = phrase_inf[i-1].pause;
          j++;
          mora[j] = phrase_inf[i].mora;
          strcpy( strings[j], phrase_inf[i].string );
          pause[j] = phrase_inf[i].pause;
     }
  }
/* number of phrases separated by pause came to be j+1 */
  phrase_size = j+1;

/* set parameters that are needed to predict duration of phoneme */
  for( i = 0; i < phrase_size; i++ )
  {
     if(( phoneme_size = parse_phoneme( strings[i], phonemes ))
          <= 0 )
     {
          fprintf( stderr, "%s: error in parse_phoneme.\n", prog );
          return(-1);
     }

     /* copy the first phoneme's data */

     duration_inf[offset].mora = mora[i];
     duration_inf[offset].loc = Top;          /* location is clearly top */
     duration_inf[offset].sent = 0;
     /* preceding phoneme set to be pau */
     strcpy( duration_inf[offset].pre, "pau" );
     strcpy( duration_inf[offset].fol, phonemes[1] );
     strcpy( duration_inf[offset].phn, phonemes[0] );
```

```
    for( j = 1; j < phoneme_size - 1; j++ )
    {
      duration_inf[offset+j].mora = mora[i];
      duration_inf[offset+j].loc  = Mid;
      duration_inf[offset+j].sent = 0;
      strcpy( duration_inf[offset+j].phn, phonemes[j] );
      strcpy( duration_inf[offset+j].pre, phonemes[j-1] );
      strcpy( duration_inf[offset+j].fol, phonemes[j+1] );
    }
    /* copy the last phoneme's data */

    duration_inf[offset+j].mora = mora[i];
    duration_inf[offset+j].loc = Tail;    /* location is clearly tail */
    duration_inf[offset+j].sent = 0;
    strcpy( duration_inf[offset+j].pre, phonemes[j-1] );
    /* following phoneme set to be "pau" */
/*
    strcpy( duration_inf[offset+j].fol, "pau" );
*/
    sprintf( duration_inf[offset+j].fol, "P%d", pause[i] );
/*  [Takeda.2] for fixing of bug associated with pause processing */
    strcpy( duration_inf[offset+j].phn, phonemes[j] );

    /* count up phoneme size */
    offset += phoneme_size;

    /* generate pause as if it is a phoneme */
    duration_inf[offset].mora = 0;
    duration_inf[offset].loc  = 0;
    duration_inf[offset].sent = 0;
    strcpy( duration_inf[offset].pre, "");
    strcpy( duration_inf[offset].fol, "" );

    if     ( pause[i] == 0 )
        strcpy( duration_inf[offset].phn, "P0" );
    else if( pause[i] == 1 )
        strcpy( duration_inf[offset].phn, "P1" );
    else if( pause[i] == 2 )
        strcpy( duration_inf[offset].phn, "P2" );
    else {
        fprintf( stderr, "%s: invalid pause of phrase inf %d.\n",
                 prog, phrase_inf[i].pause );
        return(-1); }
    offset++;
  }

  /* return value meaning phoneme size is offset - 1.
     Because of omitting of last pause */
  return( offset - 1 );
}

/*
 Horrible program.
 If you wish to be only user of this program don't read !!
 Are you the hacker that I've been waiting for.
 Anyway this program INTENDING devide strings into phoneme array.
*/
parse_phoneme( string, array )
char string[];
char array[50][10];                    /* output phoneme array */
{
    int        length;                 /* length of text string */
    int        array_len = 0;          /* length of phoneme array */
    char       c;
    char       phoneme[2];
```

```
    char        buf[256];               /* string buffer */
    int         i;

    phoneme[1] = '\0';
    length = strlen( string );
    strcpy( buf, "" );

    for( i = 0; i < length; i++ ){
        phoneme[0] = c = string[i];
        if( isVorNorLV(c) == 1 ){
            strcat( buf, ""); /* add nil to the end of buffer */
            if( strlen( buf ) != 0 ){
                /* copy buffer to array */
                strcpy( array[array_len], buf );
                strcpy( buf, "" );
                array_len++;
            }
            array[array_len][0] = c;
            array[array_len][1] = '\0';
            array_len++;
        }
        else { strcat( buf, phoneme ); }
    }
    return( array_len );
}

duration_set( size, duration, duration_inf )
int size;
Duration duration[];
DurationInf duration_inf[];
{

/***** edited by K.Abe '89 Oct. 16 ************************************************/
    char phn[32], pre[32], fol[32];
/***** edited by K.Abe '89 Oct. 16 ************************************************/
/*     char phn[10], pre[10], fol[10];                                             */
/***** edited by K.Abe '89 Oct. 16 ************************************************/

    float pdp();
    float pred;
    float mod_coef;

    int loc, mora;
    register int i, j;

    for( i = 0, j = 0; i < size; i++ )
    {
      if( strcmp( duration_inf[i].phn, "P0" ) != 0 )
      {
         strcpy( phn, duration_inf[i].phn );
         strcpy( pre, duration_inf[i].pre );
         strcpy( fol, duration_inf[i].fol );
         loc = duration_inf[i].loc;
         mora = duration_inf[i].mora;
         mod_coef = 1.0;

/* case long vowel */
         if( strcmp( pre, "O" ) == 0 ) strcpy( pre, "o" );
         if( strcmp( fol, "O" ) == 0 ) strcpy( fol, "o" );
         if( strcmp( pre, "E" ) == 0 ) strcpy( pre, "e" );
         if( strcmp( fol, "E" ) == 0 ) strcpy( fol, "e" );
/* long vowel end */
         if( pre[0] == 'P' )   strcpy( pre, "pau" );
         if( fol[0] == 'P' )
         {
```

```
                switch( fol[1] )
                {
                case '0':
                case '1':
                     mod_coef = Pre_pause;
                     break;
                case '2':
                     mod_coef = No_mod;
                     break;
                default:
                     fprintf( stderr, "set duration: invalid pause type.\n" );
                     mod_coef = No_mod;
                     break;
                }
                strcpy( fol, "pau" );
           }
           if( i == ( size - 1 ))        mod_coef = Sentence_end;
           pred = pdp( phn, loc, pre, fol, mora );
/*         pred *= mod_coef;      */
           if( phn[0] == 'P' )   strcpy( phn, "pau" );
           strcpy( duration[j].phn, phn );
           duration[j].length = pred;
           j++;
        }
     }
     return(j);
}

make_string( fp, phoneme_size, duration_inf )
FILE            *fp;
int             phoneme_size;
DurationInf     duration_inf[];
{
  char  strings[1024];
  int   i;

  strcpy( strings, "#" );
  for( i = 0; i < phoneme_size; i++ ) {
    if( duration_inf[i].phn[0] == 'P' )
      strcat( strings, "/" );
    else
      strcat( strings, duration_inf[i].phn ); }

  strcat( strings, "#" );
  fprintf( fp, "%s", strings );
  fclose(fp);
  return(i);
}
```

```
/*
  Mkppitch
  Make Point Pitch pettern

  Generates pitch period of each mora.
  Originally coded by Kazuya Takeda              Jul 19, 1989
*/

#include "/SYN/include/Synthesis.h"

#define E          stderr

/***** edited by K.Abe '89 Oct.03 ***********************************************/
/*      'Max_phrase' and 'Max_mora' are defined in 'Synthesis.h'                 */
/***** edited by K.Abe '89 Oct.03 ***********************************************/
/* #define Max_phrase   128                                                     */
/* #define Max_mora     32                                                      */
/***** edited by K.Abe '89 Oct.03 ***********************************************/

char *prog;

main(argc,argv)
int argc;
char *argv[];
{
  Phrase phrase[64];
  char file[256];
  int  phrases;
  int  ppitch[Max_phrase][Max_mora];

  prog = argv[0];

  /* Read phrase information */
  if(( phrases = ReadPhrase( argv[1], phrase )) <= 0 )
  {
    fprintf(E,"%s: Phrase information file (%s) read error.\n", prog, file );
    exit(2);
  }


  /* call pitch patter generate routine */
  set_F0( phrases, phrase, ppitch );

  /* write out to xxx.PP file */
  PrintOutPointPitch( phrases, phrase, ppitch, argv[1] );

}

PrintOutPointPitch( size, ph, pit, file )
Phrase ph[];
int size, pit[Max_phrase][Max_mora];
char *file;
{
  char file_name[256];
  FILE *out_fp;
  register int i, j;

  sprintf( file_name, "%s.PP", file );
  out_fp = fopen( file_name, "w" );

  for( i = 0; i < size; i++ )
  {
    for( j = 0; j < ph[i].mora; j++ )
        fprintf( out_fp, "%5d", pit[i][j] );
```

```
    fprintf( out_fp, "\n" );
  }
}
```

```
/*
  Mkpitch

  Generate pitch period of each synthesized frame
  Originally coded by Kazuya Kin TAKEDA. ATR,    Jul 28, 1988
  [Takeda 1]                                     Sep 26, 1988
  Point pitch patern is read from file or generate from rule.
  [Takeda 2]                                     May 31, 1989
  The interpolation is decided form the concatenating condition
  between two phrases.
  {This function was deleted, with revision [Takeda 3]}
  [Takeda 3]                                     June 19, 1989
  Point pitch patern is read from .PP file
  [Takeda 4, Sep 5, 1989]
  To adapt adaptive junction controlling.
  [Takeda 5, Oct 26, 1989]
  To adapt complicated junctions.
  [Takeda 6, Nov 10, 1989]
  You should consider the case that unit boundary can be located
  in the middle of a phoneme, even before or after the phoneme
  inter phrase boundary lies.
  [Takeda 7, Nov 16, 1989]
  If you use naural prosodic parameters, you will encounter
  unexpecting shortness of vowel duration.
  In such case, mora start frame and mora center frame can be
  coinside. This is to deal with this.
*/
#include "/SYN/include/Synthesis.h"
#include <fcntl.h>

#define Reset           0
#define Continue        1

#define E               stderr
/***** edited by K.Abe '89 Oct.03 ***********************************************/
/*      'Max_phoneme','Max_frame','Max_phrase' and 'Max_mora' are defined       */
/*      in 'Synthesis.h'                                                        */
/***** edited by K.Abe '89 Oct.03 ***********************************************/
/***** edited by K.Abe '89 Sep.22 ***********************************************/
/* #define Max_phoneme  1024                                                    */
/* #define Max_frame    8192                                                    */
/* #define MaxPhrase    128                                                     */
/* #define MaxMora      32                                                      */
/********************************************************************************/
/* #define MaxPhrase    10                                                      */
/* #define MaxMora      10                                                      */
/***** edited by K.Abe '89 Sep.22 ***********************************************/

#define Duplicate       8
#define Insertion       4
#define Fusion          2
#define Pause           1

#define PhPause                 1
#define PhUsualMatch            2
#define PhFusion                4
#define PhDuplicate             8  /* [Takeda 4] */

#define Sampling        12000.0
#define Wachou          20.0

char *prog;

typedef struct
        {
```

```
          int size;
          int stt;
          int end;
          float data[30];
          int frame[30];
        } PhrasePitch;

struct
{
  int size;
  int pp[5];
} table[32][64];


main(argc,argv)
int argc;
char *argv[];
{
  PhrasePitch php[64];

/***** edited by K.Abe '89 Sep.22 ******************************/
  PhonemeOutLine phone[Max_phoneme];
  FrameOutLine frame[Max_frame];
/************************************************************/
/*  PhonemeOutLine phone[256];                                */
/*  FrameOutLine frame[4096];                                 */
/***** edited by K.Abe '89 Sep.22 ******************************/

  /* [Takeda 5]  table is generated in order to make string
                 to correspond with phone */
  int mtsize;                          /* size of table */

  Phrase phrase[64];
  FILE *fp_pt_pattern, *fopen();

  short ppitch[Max_phrase][Max_mora];

  char file[256];

  short freq_to_period();

  short *pitch_array;
  float *voiced;
  short *index_array;

  int fd;
  int frms, phones, phrases, voiced_frames;
  register int i;


  prog = argv[0];

  /* [Takeda 1] Pitch pattern can be read from file */
  /* No, it can not !! */
  if( argc == 1 )
  {
    fprintf(E,"[%s]; generate pitch period file for synthesis.\n", prog );
    fprintf(E,"[Usage]: %s file-name-header\n", prog );
    exit(1);
  }

  /* Open poitn pitch data file [Takeda 3] */
  sprintf( file, "%s.PP", argv[1] );
  if(( fp_pt_pattern = fopen( file, "r" )) == NULL )
  {
```

```
    fprintf(E, "%s: input pitch paern file %s can't read.\n",
            prog, file );
    exit(1);
  }

  /* Get frame data from xxx.FR file */
  if(( frms = ReadFrameOutLine( argv[1], frame )) <= 0 )
  {
    fprintf(E,"%s: frame data file read error.\n", prog );
    exit(1);
  }

  /* Read phrase information */
  if(( phrases = ReadPhrase( argv[1], phrase )) <= 0 )
  {
    fprintf(E,"%s: Phrase information file read error.\n", prog );
    exit(2);
  }

  /* Rewrite long vowel symbol into consequtive vowel sequence
  Parse_phrase_string( phrases, phrase );
  */

  /* Read unit phoneme (xxx.PN) informations */
  if(( phones = ReadPhonemeOutLine( argv[1], phone )) <= 0 )
  {
    fprintf(E,"%s: Phone file read error.\n", prog );
    exit(3);
  }

  /* Read poitn pitch data  [Takeda 3] */
  if( ReadPointPitch( fp_pt_pattern, phrases, phrase, ppitch ) < 0 )
  {
    fprintf( E, "%s: Point pitch pattern read error.\n", prog );
    exit(4);
  }

  /* create pitch data file */
  sprintf( file, "%s.PT", argv[1] );
  if(( fd = open( file, O_WRONLY|O_CREAT, 420 )) < 0 )
  {
    fprintf(E,"%s: pitch file %s can't create.\n", prog, file );
    exit(3);
  }

  /* Get work area */
  pitch_array = (short*)malloc(frms*sizeof(short)); /* Pitch preiod */
  index_array = (short*)malloc(frms*sizeof(short)); /* ptr for Voiced frame */
  voiced = (float*)malloc(frms*sizeof(float));    /* Voiced frame */

  if(( mtsize = get_table( phrases,
                           phrase,
                           phones,
                           phone )) <= 0 )
  {
    fprintf(E,"%s: error in Match .PH and .PN files\n", prog );
    exit(2);
  }

/*
  debug_print_table( phrases, phrase, phone );
*/

   /* Detect voiced_frame */
   if(( voiced_frames = set_arrays( frms,
```

```
                                        frame,
                                        index_array )) <= 0 )
    {
      fprintf(E,"%s: error in set pitch array.\n", prog );
      exit(2);
    }
  /*
    debug_print_frm( frms, index_array );
    */

    if( set_phrase_data( phrases,
                         phrase,
                         phone,
                         php,
                         ppitch,
                         index_array ) <= 0 )
    {
      fprintf(E,"%s: error in set phrasal pitch.\n", prog );
      exit(2);
    }


    if( wachou_seibun( php, phrase, phrases ) <= 0 )
    {
      fprintf(E,"%s: Wachou-seibun set error.\n", prog );
      exit(4);
    }

/*
  debug_print_php( phrases, php );
  fflush(stdout);
*/

    /*
      [Takeda May 31, 1989]
       The variable "phrase" is added to the arguments
     */

    if( linear_interpolate( php,
                            voiced,
                            phrases,
                            phrase) <= 0 )
    {
      fprintf(E,"%s: linear interpolation.\n", prog );
      exit(4);
    }

/*
    debug_print_php( phrases, php );
*/

    for( i = 0; i < frms; i++ )             /* Copy pitch data */
    {
      int idx;
      if(( idx = *(index_array+i)) <= 0 )
      {
        *(pitch_array+i) = 0;
      }
      else
      {
        *(pitch_array+i) = freq_to_period(*(voiced+idx));
      }
    }

    if( write( fd, pitch_array, frms*sizeof(short)) != frms*sizeof(short))
```

```
    {
      fprintf(E,"%s: Pitch data write error.\n", prog );
      exit(5);
    }
    exit(0);
}


set_arrays( frms, frame, index )
int frms;               /* size of frame */
FrameOutLine frame[];           /* frame data */
short index[];          /* index array to voicd frame array */
{
  register int i;
  int count = 0;                /* Counter for voiced frames */
  for( i = 0; i < frms; i++ )
  {
/************************************ 20 DEC 88 SAGI ***/
    if(( is_voiced(frame[i].phn ) == 1) ||
       ( *(frame[i].eve) == 'j' ) ||
       ( *(frame[i].eve) == 'w' ))    /* [Takeda jul 19] */
/*********************************************************/
    {
      index[i] = count++;
    }
    else if( is_including_vowel(frame[i].phn) == 1 )
    {
      index[i] = -1 * ( count++ );
    }
    else
    {
      index[i] = 0;
    }
  }
  return(count);
}


mkarray(s,l)
char *s, *l[];
{
  register int i;
  char buf[64];
  int count = 0;
  strcpy(buf,"");
  for( i = 0; i < strlen(s); i++ )
  {
    if( s[i]==',')
    {
      if( strcmp( buf, "" ) != 0 )
      {
        l[count] = (char*)malloc(strlen(buf)+1);
        strcpy(l[count++],buf);
        strcpy( buf, "" );
      }
    }
    else if( s[i]=='a'||s[i]=='i'||s[i]=='u'||s[i]=='e'||s[i]=='o'||s[i]=='N'
           ||s[i] == 'O'||s[i]=='E' )
    {
      if( strcmp( buf, "" ) != 0 )
      {
        l[count] = (char*)malloc(strlen(buf)+1);
        strcpy(l[count++],buf);
        strcpy( buf, "" );
      }
```

```
      l[count] = (char*)malloc(2);
      sprintf(l[count++],"%c",s[i]);
      strcpy(buf,"");
    }
    else
    {
      sprintf(buf,"%s%c",buf,s[i]);
    }
  }
  return(count);
}

break_down_lb( s, l )
char s[], *l[];
{
  int count = 0;
  char buf[64];
  register int i;

  strcpy( buf, "" );
  for( i = 0; i < strlen(s); i++ )
  {
    if( s[i] == ',' )
    {
      l[count] = (char*)malloc(strlen(buf)+1);
      strcpy( l[count++], buf );
      strcpy( buf, "" );
    }
    else
    {
      sprintf( buf, "%s%c", buf, s[i] );
    }
  }
  l[count] = (char*)malloc(strlen(buf)+1);
  strcpy(l[count++], buf );
  return(count);
}

mark_ref_frame( stt, end, phone, p_pitch, mora, php, index )
int stt, end;           /* start and end phoneme no */
PhonemeOutLine phone[];         /* phone information */
float p_pitch[];        /* point pitch array */
int mora;               /* mora length */
PhrasePitch *php;       /* reference table */
short index[];          /* index array */
{
  register int i = 0, j = 0, k = 0, wi;
  int vowels;
  int work_length; /* whole duraiton of the vowel portion */
  int off_set; /* the start offset of the vowel portion */

  for( i = stt; i <= end; i++ )
  {
    if(( strcmp( phone[i].phn, "pau" ) != 0 )
       /*
         (!(phone_flag[i]&Pause))
         && !(phone_flag[i]&Insertion))
         The insertion flag is needless any more
         */
       &&
    (( vowels = how_many_vowels(phone[i].phn)) > 0 ))
    {
        /* [Takeda 4] */

      /* Here is the start frame of the vowel portion */
```

```
        off_set = (int)index[phone[i].stt];
        /* Here is the default length of the vowel portion */
        work_length = (int)index[phone[i].end - 1 ] -
                      (int)index[phone[i].stt];

        /* if units are overlapped to adjust the concatenating point */
        if( phone[i].inf >= PhDuplicate)
        {
          while( phone[i].inf >= PhDuplicate )
          {
            work_length += ((int)index[phone[i].end-1] -
                            (int)index[phone[i].stt]);
            /* summ up the length of vowel portion */
            i++;  /* Skip the overlapped phoneme */
          }
          --i;
        }
        for( k = 0; k < vowels; k++ )
        {
          php->data[j] = p_pitch[j];
          /* Hacked by T.Noripy.Yamazaki,
             This Hack is re-Hacked by K. Kinpy. Takeda
             if( p_pitch[j] < 100) php->data[j] = p_pitch[j] + 30; */
          /*   :-)  X-)  :-P  :-<  :->   */
            php->frame[j] = (k+1)*((float)work_length/(float)(vowels+1))
                            + off_set;
          if( php->frame[j] < 0 ) php->frame[j] = -1 * php->frame[j];
          j++;
        }
      }
    }
    if( j != mora )
    {
      fprintf(E,"%s: Mora count and vowels not coinside.\n", prog );
      return(-1);
    }
    php->size = mora;
    return(mora);
}

/*
complete_php_array( stt, end, php, index )
int stt, end;
PhrasePitch *php;
short index[];
{
  register int i;
  i = stt;
  while( index[i] == 0 )        i++;
  php->stt = ( index[i] > 0 ? index[i] : -1 * index[i] );
  i = end;
  while( index[i] == 0 )        i--;
  php->end = ( index[i] > 0 ? index[i] : -1 * index[i] );
}
*/

is_voiced(s)
char *s;
{
  char c;
  if( isvowel(s)==1||(c=s[0])=='b'||c=='d'||c=='g'||c=='m'||c=='n'||
      c=='z'||c=='j'||c=='r'||c=='y'||c=='w')
       return(1);
  else return(0);
}
```

```
isvowel(s)
char *s;
{
  if( s[0]=='a'||s[0]=='i'||s[0]=='u'||s[0]=='e'||s[0]=='o'||s[0]=='N')
        return(1);
  else return(0);
}

is_including_vowel(s)
char *s;
{
  if( strcmp( s, "pau" ) == 0 ) return(0);
  return( how_many_vowels(s) > 0 ? 1 : 0 );
}

how_many_vowels(s)
char *s;
{
  int count = 0;
  register int i;
  for( i = 0; i < strlen(s); i++ )
  {
    if( s[i]=='a'||s[i]=='i'||s[i]=='u'||s[i]=='e'||s[i]=='o'||s[i]=='N')
         count++;
  }
  return(count);
}


Parse_phrase_string( size, phr )
int size;
Phrase phr[];
{
  char buf[256];
  char *ptr;
  register int i, j;
  unsigned int flag = 0;
  for( i = 0; i < size; i++ )
  {
    strcpy(buf,"");
    ptr = phr[i].string;
    for( j = 0; j < strlen(ptr); j++ )
    {
      switch(*(ptr+j))
      {
      case 'A':
        strcat(buf,"a,a");
        flag = 1;
        break;
      case 'I':
        strcat(buf,"i,i");
        flag = 1;
        break;
      case 'U':
        strcat(buf,"u,u");
        flag = 1;
        break;
      case 'E':
        strcat(buf,"e,i");
        flag = 1;
        break;
      case 'O':
        strcat(buf,"o,u");
        flag = 1;
```

```
        break;
      default:
        sprintf(buf,"%s%c",buf,*(ptr+j));
      }
    }
    if( flag & 1 )
    {
      free( phr[i].string );
      phr[i].string = (char*)malloc(strlen(buf)+1);
      strcpy(phr[i].string, buf);
    }
  }
}


short freq_to_period(f)
float f;
{
  if( f == 0.0 ) return(0);
  return((short)(Sampling/f));
}


wachou_seibun( php, phrase, phrases )
PhrasePitch php[];
Phrase phrase[];
int phrases;
{
  struct { int size; float baias[64]; } wachou[32];
  float step;
  int w_cnt = 0, p_cnt = 0, b_cnt = 0, d_cnt = 0;
  int wachou_no_kazu;
  register int i, j = 0;

  for( i = 0; i < phrases; i++ )
  {
    wachou[j].size = php[i].size;
    while( phrase[i].pause == 0  && i < phrases )
    {
      wachou[j].size += php[i+1].size;
      i++;
    }
    j++;
  }
  wachou_no_kazu = j;
  for( i = 0; i < wachou_no_kazu; i++ )
  {
    step = Wachou / wachou[i].size;
    for( j = 0; j < wachou[i].size; j++ )
    {
/*
      wachou[i].baias[j] = Wachou - step * (float)j;
      wachou[i].baias[j] = Wachou/2.0 - step * (float)j;
*/
      wachou[i].baias[j] = 0.0 - step * (float)j;

    }
  }
  for( p_cnt = 0; p_cnt < phrases; p_cnt++ )
  {
    b_cnt = 0;
    for( d_cnt = 0; d_cnt < php[p_cnt].size; d_cnt++ )
        php[p_cnt].data[d_cnt] += wachou[w_cnt].baias[b_cnt++];
    while( phrase[p_cnt].pause == 0 )
    {
```

```
      for( i =  0; i < php[p_cnt].size; i++ )
           php[p_cnt].data[d_cnt++] += wachou[w_cnt].baias[b_cnt++];
      p_cnt++;
    }
    w_cnt++;
  }
}

/* [Takeda3]
   Routine for reading poitn pitch pattern. */
ReadPointPitch( fp, size, ph, pp )
FILE *fp;
int size;
Phrase ph[];
short pp[Max_phrase][Max_mora];
{
  char line[256];
  char *av[256];
  register int i, j;

  for( i = 0; i < size; i++ )
  {
    if( fgets( line, 256, fp ) == 0 ) return(-1);
    line[strlen(line)-1] = '\0';

    if( ParseLine( line, ' ', av ) != ph[i].mora )
    {
      fprintf(E, "ReadPitchPtaern: mora length and ref points incoinside.\n" );
      return(-1);
    }

    for( j = 0; j < ph[i].mora; j++ )
    {
      sscanf( av[j], "%d", &pp[i][j] );
    }
  }
  return(1);
}


ParseLine( line, del, av )
char *line;
char del;
char *av[];
{
  char c;
  char *buf;
  register int ac = 0;

  buf = (char*)malloc(1024);
  *buf = '\0';
  while( c = *line++ )
  {
    if( c == del )
    {
      if( *buf != '\0' )
      {
        av[ac] = (char*)malloc(strlen(buf)+1);
        strcpy( av[ac], buf );
        ac++;
        *buf = '\0';
      }
    }
    else
    {
```

```
        sprintf( buf, "%s%c", buf, c );
      }
    }
    av[ac] = (char*)malloc(strlen(buf)+1);
    strcpy( av[ac], buf );
    ac++;
    return(ac);
  }


debug_print_php( size, php )
 int size;
 PhrasePitch php[];
 {
   register int iz, iiz;
   for( iz = 0; iz < size; iz++ )
   {
     printf( "[%d to %d], size = %d\n",
            php[iz].stt, php[iz].end, php[iz].size );
     for(iiz = 0; iiz < php[iz].size; iiz++ )
          printf( "%f (data)  %d frame\n",
                 php[iz].data[iiz], php[iz].frame[iiz]);
   }
 }

 debug_print_table( size, phrase, phone )
 int size;
 Phrase phrase[];
 PhonemeOutLine phone[];
 {
   register int i, j, k;
   int psize;
   int buf;
   char *av[48];

   for( i = 0; i < size; i++ )
   {
     psize = mkarray( phrase[i].string, av );
     for( j = 0; j < psize; j++ )
     {
       printf( "(%s) table[%d][%d] = ", av[j], i, j );
       for( k = 0; k < table[i][j].size; k++ )
       {
         buf = table[i][j].pp[k];
         printf( " %d %s", buf, phone[buf].phn );
       }
       printf( "\n" );
     }
   }
 }
 get_table( phrs, phr, phns, phn )
 int phrs;              /* size of phrase */
 Phrase phr[];          /* phrase information */
 int phns;              /* size of unit-phoneme */
 PhonemeOutLine phn[];  /* unit-phoneme information */
 {
   int off_set = 0;     /* start off (no of the roman phoneme sequence)
                           of the phrase */
   register int iphr;
   register int ipp = 0;
   register int jpp = 0;
   register int rp_c = 0;  /* index for the roman phoneme in the unit string */
   register int irphn;
   int tab_size = 0;
```

```
    int roman_phs;              /* phrase length in the roman phoneme */
    char *r_phn[48];            /* roman phoneme sequence array */

    int index = 0;
    int pau = 0;

    for( iphr = 0; iphr < phrs; iphr++ )
    {
      /* Obtaine roman phoneme sequence from .PH file */
      roman_phs = mkarray( phr[iphr].string, r_phn );

      for( irphn = 0; irphn < roman_phs; irphn++, index++ )
      {
        /* initialize table */
        table[iphr][irphn].size = 0;

        pau = 0;
        for( ipp = 0; ipp < phns; ipp++ )
        {
          if( ! strcmp( phn[ipp].phn, "pau" ))
          { pau++ ; continue; }

          for( jpp = 0; jpp < phn[ipp].rp_size; jpp++ )
          {
            if( index + pau == phn[ipp].rp[jpp] )
            {
              table[iphr][irphn].pp[  table[iphr][irphn].size ++ ] = ipp;
              tab_size++;
            }
          }
        }
      }
    }
    return(tab_size);
}
set_phrase_data( phrases, phrase, phone, php, ppitch, index )
int phrases;
Phrase phrase[];
PhonemeOutLine phone[];
PhrasePitch php[];
short ppitch[Max_phrase][Max_mora];
short index[];
{
    struct { int stt_phn, end_phn, vs; } m[64];
    char *ph_seq[64];

    register int ph, pp, i, j, im;
    int mora_count;
    int phn_size;
    int v_size, vs;
    int stt, end, length, off_set;
    int center;  /* hypothetic center of assimilated consequtive phrases */

    for( ph = 0; ph < phrases; ph++ )       /* Frase wise processing */
    {
      phn_size = mkarray( phrase[ph].string, ph_seq );

      for( pp = 0, mora_count = 0, i = 0 ; pp < phn_size; pp++ )
      {
        if(! ( v_size = how_many_vowel( ph_seq[pp] )))
             continue;

        mora_count += v_size;
        m[i].stt_phn = table[ph][pp].pp[0];
        m[i].end_phn = table[ph][pp].pp[ table[ph][pp].size - 1 ];
```

```
        m[i].vs = v_size;
        i++;
      }
      vs = i;
      if( phrase[ph].mora != mora_count )
      {
        fprintf(stderr,"%s: Mora count and vowels not coinside.\n");
        return(-1);
      }

      for( i = 0; i < mora_count; i++ )
           php[ph].data[i] = (float)ppitch[ph][i];

      length = 0;
      v_size = 0;
      im = 0;
      for( i = 0; i < vs; i++ )
      {
        register int k = 0;
        stt = phone[ m[i].stt_phn ].stt;
        end = phone[ m[i].end_phn ].end - 1;
        off_set = index[stt];
        while( ! off_set )
             off_set = index[stt + k++];

        off_set = ( off_set < 0 ? -1 * off_set : off_set );

        v_size += m[i].vs;

        if(( i < vs - 1 ) && ( m[i].stt_phn == m[i+1].stt_phn ))
             continue;

        for( j = 0; j < v_size; j++ )
        {
          length = end - stt;

          php[ph].frame[im] = off_set + (j+1)*
               (int)((float)length / (float)(v_size + 1));
          im++;
          length = 0;
        }
        v_size = 0;
      }
      php[ph].size = mora_count;

      stt = phone[ table[ph][0].pp[0] ].stt;
      end = phone[table[ph][phn_size-1].pp[ table[ph][phn_size-1].size-1]].end;

      i = stt;
      while( index[i] == 0 )     i++;
      php[ph].stt = ( index[i] > 0 ? index[i] : -1 * index[i] );
      i = end;
      while( index[i] == 0 )     i--;
      php[ph].end = ( index[i] > 0 ? index[i] : -1 * index[i] );

    }

/*
  [Takeda 6] there would be inter phrase assimilations.
*/

    for( ph = 1; ph < phrases; ph++ )
    {
      if( php[ph].stt <= php[ph-1].end )
           /* This is the case that bothers me !! */
```

```
      {
        center = php[ph].stt + php[ph-1].end;
        center /= 2;
        php[ph].stt = center +1;
        php[ph-1].end = center;
      }
    }
  }

  how_many_vowel( s )
  char *s;
  {
    register int i;
    int count = 0;
    char c;

    for( i = 0; i < strlen(s); i++ )
    {
      c = *(s+i);
      if( c == 'a' || c == 'i' || c == 'u' || c == 'e' || c == 'o' || c == 'N' )
      { count++; continue; }

      if( c == 'A' || c == 'I' || c == 'U' || c == 'E' || c == 'O' )
           count += 2;
    }
    return(count);
  }

  debug_print_frm( size, index )
  int size;
  short index[];
  {
    register int i, j;

    for( i = 0; i < size - 8 ; i += 8 )
    {
      for( j = 0; j < 8; j++ )
           printf( "%5d", index[ i +j] );
      printf( "\n" );
    }
  }

linear_interpolate( php, voiced, phrases, phrase )
PhrasePitch php[];
float *voiced;
int phrases;
/* [Takeda May 31, 1989] phrase is added to args. */
Phrase phrase[];
{
  float ratio;
  PhrasePitch *pr;
  register int i, j, k;

/**** Slight modification for the phrase initial portion ****/
  int    F0s ;
  float  F0sratio ;
  pr   = &php[0] ;
  F0s  = pr->data[0]-10 ;
/*************************** Y. SAGISAKA 29 NOV '88 ********/

  for( i = 0; i < phrases; i++ )
  {
    pr = &php[i];
    ratio = (pr->data[1] - pr->data[0])
             / (float)(pr->frame[1] - pr->frame[0]);
```

```
/*************************************************/
    if( pr->frame[0] != pr->stt )
    /* [Takeda 7]
        In some case, start frame and mora-center frmae are
        coinside. */
    {
      F0sratio = ( pr->data[0] - F0s )
            / (float)(pr->frame[0] - pr->stt );
/************************* Y. SAGISAKA 29 NOV '88 **/
      for( j = pr->stt; j < pr->frame[0]; j++ )
      {
/*********************************************************************/
        *(voiced+j) = F0sratio * (float)(j - pr->stt) + F0s ;
/************************************ Y. SAGISAKA 29 NOV '88 *******/
      }
    }
    for( j = 1; j < pr->size; j++ )
    {
      if( pr->frame[j] == pr->frame[j-1] )
           continue;

      ratio = (pr->data[j] - pr->data[j-1]) /
                    (float)(pr->frame[j] - pr->frame[j-1]);

      for( k = pr->frame[j-1]; k < pr->frame[j]; k++ )
      {
        *(voiced+k) = ratio*(float)(k - pr->frame[j-1]) + pr->data[j-1];
      }
    }
    for( j = pr->frame[pr->size-1]; j < pr->end; j++ )
    {
/* ------------------------------------------
  [Takeda May 31, 1989]
  these lines are add to enable to consider the
  condition of next phrase pitch.
  ------------------------------------------- */
      if( i < ( phrases - 1) )
      {
        if( phrase[i].pitch == Continue )
             ratio = ( php[i+1].data[0] - pr->data[pr->size - 1] )/
              (float)(php[i+1].frame[0] - (pr->frame[pr->size  - 1]));
      }
     /* ------------------------------------------------------------ */
      *(voiced+j) = ratio*(float)(j-pr->frame[pr->size-1])
                                  + pr->data[pr->size-1];
    }
/*****************************/
    F0s = pr->data[pr->size-1] ;
/**** Y. SAGISAKA 29 NOV '88 **/
  }
/*************************/
  return(1);
/**** K. Abe 13 SEP '89 ****/
}
```

```
/*
  mkunit_file.c
  read unit select output and generate unit file and unit attribute file

  Originally coded by Kazuya K. Takeda [July 7, 1988]

  [Takeda 1] July 11, 1989
  Input file, xxx.ett, format is changed.
  The stt and end location of the speech segment in a unit
  are count by phonetic symbol trancsription of label.

  [Takeda 2], Nov 13, 1989
  The pause segment beccome to be included int the .ett file
  description inorder to let know the 'mkunit program whether
  the unit concatenation should be done at inter segment or not.

*/
#include "/SYN/include/Synthesis.h"
#include <stdio.h>

#define E                       stderr

/***** edited by K.Abe '89 Oct.03 ***********************************************/
/*      'Max_unit' is defined in 'Synthesis.h'                                   */
/***** edited by K.Abe '89 Oct.03 ***********************************************/
/***** edited by K.Abe '89 Sep. 22 **********************************************/
/* #define Max_unit     512                                                     */
/********************************************************************************/
/* #define Max_unit     128                                                     */
/***** edited by K.Abe '89 Sep. 22 **********************************************/

#define Start_In_Fusion         2
#define End_In_Fusion           4

char *prog;

main(argc,argv)
int argc;
char *argv[];
{
  UnitAttribute *unit[Max_unit], *read_label(), *set_pause();
  Entity ent[Max_unit];
  FILE *fpin, *fpout, *fopen();

  int units;
  char fin[256], fout[256];
  char *RomanFilter();

  register int i;

  prog = argv[0];

  if(argc==1)
  {
    fprintf(E,"\t[%s] create unit attribute file from entity file.\n", prog );
    fprintf(E,"\t[Usage]: %s file-name-header.\n", prog );
    exit(0);
  }

  sprintf(fin, "%s.ett", argv[1] );
  sprintf(fout, "%s.oua", argv[1] );
  if(( fpin = fopen( fin, "r" )) == NULL )
  {
    fprintf(E,"%s: entity file %s can't open.\n", prog, fin );
    exit(1);
```

```
  }
  if(( fpout = fopen( fout, "w" )) == NULL )
  {
    fprintf(E,"%s: (write-out)unit attribute file %s can't open.\n"
              , prog, fout );
    exit(1);
  }
  if(( units = GetEntity( fpin, ent )) <= 0 )
  {
    fprintf(E,"%s: entity file read failed.\n", prog );
    exit(1);
  }
  for( i = 0; i < units; i++ )
  {
    char *buf_text; /* in order to convert "O" to "ou" .*/

    if( ent[i].word == 0 )
    {
      unit[i] = set_pause();
      PutUnitAttribute( fpout, unit[i] );
      continue;
    }

    if(( unit[i] = read_label(ent[i]))
                    == (UnitAttribute*)NULL )
    {
      fprintf(E,"%s: error in read_label( unit # = %d ).\n", prog, i );
      exit(2);
    }
    buf_text = RomanFilter(ent[i].roman);
    unit[i]->roman = (char*)malloc(strlen(buf_text)+1);
    strcpy( unit[i]->roman, buf_text );
    PutUnitAttribute( fpout, unit[i] );
  }
  exit(0);
}

/*
  read_label
  indata;
       word_id: word no in which the unit is including.
       start;   start phoneme no.
       end;     end  phoneme no.
  return;
       pointer for unit attribute data.
*/
UnitAttribute *read_label(ent)
Entity ent;
{
  struct { int stt, end; char *sym; } work[3][256];
  UnitAttribute *unit;
  Phone ph[256];
  Event ev[512];
  Alophone al[128];
  FILE *fp, *open_label_file();

  int word;              /* word # */
  int stt, end;          /* start/end phoneme of unit  start with 1 */
                         /* [Takeda 1]
                            these two parameters are changed to be counted
                            by label, so you don't need match
                            phonemic symbol and label any more !. */
  char *roman;           /* roman description of the word */
  char l[256];           /* read buffer */
  int size[3];           /* size of each layer of label */
```

```
  /* [Takeda 1]
      l_stt and l_end are needless !.
  int l_stt, l_end;       label no.
                          associating first and last phoneme of unit */
  int layer = 0;         /* layer no of work array
                                  0; phonemic label layer
                                  1; event label layer
                                  2; alophonic variation layer
                          */
  int label = 0;         /* symbol no */
  unsigned int fusion_flag = 0;

  register int i, j, k;

  unit = (UnitAttribute*)malloc(sizeof(UnitAttribute));
  /* set word no of the unit */
  unit->word = word = ent.word;

  if( word < 1 || word > 5240 )
  {
    fprintf(E,"read_label: Bad word no (%d).\n", word );
    return((UnitAttribute*)NULL);
  }
  if(( fp = open_label_file(word)) == NULL )
  {
    fprintf(E,"read_label: label file (%04d) can't open.\n", word );
    return((UnitAttribute*)NULL);
  }

  /* set word and location attribute of unit */
  unit->stt = stt = ent.stt;
  unit->end = end = ent.end;
  unit->wd_roman = (char*)malloc(strlen(ent.wd_roman)+1);
  strcpy( unit->wd_roman, ent.wd_roman );

  /* read label */
  while( fgets( l, 256, fp ) != 0 )
  {
    char lb[64];          /* read buffer for label symbol */
    float f_stt, f_end; /* read buffer for label start and end */

    if( l[0] == '#' )
    /* read next layer */
    {
      size[layer] = label;
      if( ++layer > 2 ) break;
      label = 0;
    }
    else
    {
      sscanf( l, "%f %s %f", &f_stt, lb, &f_end );
      work[layer][label].stt = f_stt/Ratio;
      work[layer][label].end = f_end/Ratio;
      work[layer][label].sym = (char*)malloc( strlen(lb)+1);
      strcpy( work[layer][label].sym, lb );
      label++;
    }
  }

  /* set Phoneme data */
  for( i = 0; i < size[0]; i++ )
  {
    ph[i].stt = work[0][i].stt;
    ph[i].end = work[0][i].end;
    ph[i].phn = (char*)malloc(strlen(work[0][i].sym)+1);
```

```
    strcpy( ph[i].phn, work[0][i].sym );
  }
  /* set Event data */
  for( i = 0; i < size[1]; i++ )
  {
    ev[i].stt = work[1][i].stt;
    ev[i].end = work[1][i].end;
    ev[i].eve = (char*)malloc(strlen(work[1][i].sym)+1);
    strcpy( ev[i].eve, work[1][i].sym );
  }
  /* set Alophone data */
  for( i = 0; i < size[2]; i++ )
  {
    al[i].stt = work[2][i].stt;
    al[i].end = work[2][i].end;
    al[i].alp = (char*)malloc(strlen(work[2][i].sym)+1);
    strcpy( al[i].alp, work[2][i].sym );
  }

  /* [Takeda 1] You don't have to match string and label
                nay more ! */
  /* Match strings and label
  fusion_flag = match( work[0], stt, end, &l_stt, &l_end );
  */

  /* set pre/fol char */
  /* the index variable l_stt and l_end are changed to be
     stt and end, respectively. */

  stt--; end--;          /* stt and end should be counted from 0 */

/***** edited by K.Abe '89 Oct. 17 ********************************************/
  if( stt == 0 )         /* if the unit start from initial of word */
  {
    unit->pre = (NeighborPhone*)malloc(sizeof(NeighborPhone));
    unit->pre->stt = 0;
    unit->pre->end = 0;
    unit->pre->phn = (char*)malloc(4);
    strcpy( unit->pre->phn, "Top" );
  }
  else
  {
    unit->pre = (NeighborPhone*)malloc(sizeof(NeighborPhone));
    unit->pre->stt = ph[stt-1].stt;
    unit->pre->end = ph[stt-1].end;
    unit->pre->phn = (char*)malloc(strlen(ph[stt-1].phn)+1);
    strcpy( unit->pre->phn, ph[stt-1].phn );
  }
  if( end == size[0] - 1 )      /* if the unit end at the end of the word */
  {
    unit->fol = (NeighborPhone*)malloc(sizeof(NeighborPhone));
    unit->fol->stt = 0;
    unit->fol->end = 0;
    unit->fol->phn = (char*)malloc(4);
    strcpy( unit->fol->phn, "End" );
  }
  else
  {
    unit->fol = (NeighborPhone*)malloc(sizeof(NeighborPhone));
    unit->fol->stt = ph[end+1].stt;
    unit->fol->end = ph[end+1].end;
    unit->fol->phn = (char*)malloc(strlen(ph[end+1].phn)+1);
    strcpy( unit->fol->phn, ph[end+1].phn );
  }
/***** edited by K.Abe '89 Oct. 17 ********************************************/
```

```
/***** edited by K.Abe '89 Oct. 17 ***********************************************/
/*  if( stt == 0 )                                                                */
/*  {                                                                             */
/*    unit->pre = (char*)malloc(4);                                               */
/*    strcpy( unit->pre, "Top" );                                                 */
/*  }                                                                             */
/*  else                                                                          */
/*  {                                                                             */
/*    unit->pre = (char*)malloc(strlen(ph[stt-1].phn)+1);                         */
/*    strcpy( unit->pre, ph[stt-1].phn );                                         */
/*  }                                                                             */
/*  if( end == size[0] - 1 )                                                      */
/*  {                                                                             */
/*    unit->fol = (char*)malloc(4);                                               */
/*    strcpy( unit->fol, "End" );                                                 */
/*  }                                                                             */
/*  else                                                                          */
/*  {                                                                             */
/*    unit->fol = (char*)malloc(strlen(ph[end+1].phn)+1);                         */
/*    strcpy( unit->fol, ph[end+1].phn );                                         */
/*  }                                                                             */
/***** edited by K.Abe '89 Oct. 17 ***********************************************/

  /* set ph field of unit data */
  for( i = stt; i <= end; i++ )
  {
    unit->phn[i-stt] = (Phone*)malloc(sizeof(Phone));
    unit->phn[i-stt]->stt = ph[i].stt;
    unit->phn[i-stt]->end = ph[i].end;
    unit->phn[i-stt]->phn = (char*)malloc(strlen(ph[i].phn)+1);
    strcpy( unit->phn[i-stt]->phn, ph[i].phn );
  }

/***** edited by K.Abe '89 Oct. 17 ***********************************************/
  if( strcmp(unit->pre->phn, "Top") == 0 &&
     ( strcmp(unit->phn[0]->phn, "p") == 0 ||
       strcmp(unit->phn[0]->phn, "t") == 0 ||
       strcmp(unit->phn[0]->phn, "k") == 0 ) )
  {
    unit->phn[0]->stt -= 10;
    printf( " Boundary is moved. unit->phn[0]->stt = %d\n",
           unit->phn[0]->stt );
  }
/***** edited by K.Abe '89 Oct. 17 ***********************************************/

  unit->phn_size = end - stt + 1;

  for( i = 0; i < unit->phn_size; i++ )
  {
    int eve_size = 0;   /* event included in the i-th phoneme */
    int alp_size = 0;   /* alophonic symbols included in the i-th phoneme */

    /* search event that is included in the phoneme */
    for( j = 0; j < size[1]; j++ )
    {
      if( unit->phn[i]->stt <= ev[j].stt &&
          unit->phn[i]->end >= ev[j].end )
      {
        unit->phn[i]->eve[eve_size] = (Event*)malloc(sizeof(Event));
        unit->phn[i]->eve[eve_size]->stt = ev[j].stt;
        unit->phn[i]->eve[eve_size]->end = ev[j].end;
        unit->phn[i]->eve[eve_size]->eve = (char*)malloc(strlen(ev[j].eve)+1);
        strcpy( unit->phn[i]->eve[eve_size]->eve, ev[j].eve );
        eve_size++;
      }
```

```
    }
    /* set including event size */
    unit->phn[i]->eve_size = eve_size;

    /* search alophonic symbol that is included in the phoneme */
    for( j = 0; j < size[2]; j++ )
    {
      if( unit->phn[i]->stt <= al[j].stt &&
          unit->phn[i]->end >= al[j].end )
      {
        unit->phn[i]->alp[alp_size] = (Alophone*)malloc(sizeof(Alophone));
        unit->phn[i]->alp[alp_size]->stt = al[j].stt;
        unit->phn[i]->alp[alp_size]->end = al[j].end;
        unit->phn[i]->alp[alp_size]->alp = (char*)malloc(strlen(al[j].alp)+1);
        strcpy( unit->phn[i]->alp[alp_size]->alp, al[j].alp );
        alp_size++;
      }
    }
    /* set included alophonic symbol size */
    unit->phn[i]->alp_size = alp_size;
  }

  /* set unit full length */
  unit->length = unit->phn[ unit->phn_size - 1 ]->end
               - unit->phn[0]->stt + 1;
  return(unit);
}


FILE *open_label_file(no)
int no;
{
  FILE *fp;
  char f[256];

  sprintf( f, "/MHT/LBL/D%d/MHT_1_%04d.LB", no/1000, no );
  if(( fp = fopen( f, "r" )) == NULL )
  {
    fprintf(E,"open_label_file: can't open %s.\n", f );
    return((FILE*)NULL);
  }
  return(fp);
}

char *RomanFilter(str)
char *str;
{
  char ret[100];
  char c;
  register int i = 0;

  while( c = *str++ )
  {
    if( c == 'O' )
    {
      ret[i++] = 'o'; ret[i] = 'u';
    }
    else if( c == 'E' )
    {
      ret[i++] = 'e'; ret[i] = 'i';
    }
    else
    {
      ret[i] = c;
    }
```

```
    i++;
  }
  ret[i] = '\0';
  return( ret );
}

/* [Takeda 1] This is needless now !
;;match( label, stt, end, l_stt, l_end )
;;struct { int stt, end; char *lbl; } label[];
;;int stt, end, *l_stt, *l_end;
;;{
;;  int off_set = 0;
;;  int retcode = 0;
;;  register int i = 0;
;;
;;  while( off_set < stt )
;;       off_set += count_phoneme(label[i++].lbl);
;;  if( off_set != stt )        retcode = Start_In_Fusion|retcode;
;;  *l_stt = i-1;
;;  while( off_set < end )
;;       off_set += count_phoneme(label[i++].lbl);
;;  if( off_set != end )        retcode = End_In_Fusion|retcode;
;;  *l_end = i-1;
;;  return(retcode);
;;}
;;
;;
;;int count_phoneme(s)
;;char *s;
;;{
;;  register int i;
;;  int count = 0;
;;
;;  for( i = 0; i < strlen(s); i++ )
;;  {
;;    if( s[i] == ',' ) count++;
;;  }
;;  return(count+1);
;;}
;;
*/

/*
  [Takeda 2]
  This ruotine inserts a puase segment same as
  selected unit segments.
  */
UnitAttribute *set_pause()
{
  UnitAttribute *u;
  NeighborPhone *nbp;


  nbp = (NeighborPhone*)malloc(sizeof(NeighborPhone));
  nbp->phn = (char*)malloc(4);
  strcpy( nbp->phn, "nil" );
  nbp->stt = nbp->end = 0;

  u = (UnitAttribute*)malloc(sizeof(UnitAttribute));
  u->pre = u->fol = nbp;
  u->length = 0;
  u->word = 0;
  u->stt = u->end = 0;
  u->roman = u->wd_roman = (char*)malloc(4);
  strcpy( u->roman, "pau");
```

```
  u->phn_size = 0;
  return(u);
}
```

```
/*****************************************************************************/
/*                                                                           */
/*      Segment Out Unit From Word Data ( mkunit.c )                         */
/*                                                                           */
/*      speech synthesis unit is cut out from word data according to         */
/*      acoustical characteristics.                                          */
/*                                                                           */
/*---------------------------------------------------------------------------*/
/*                                                                           */
/*      Originally coded by K.Takeda                    Jul. 14, 1988        */
/*                                                                           */
/*      [ Takeda 1 ]                                    Jul. 28, 1988        */
/*      Changed data interface for New data structures                       */
/*                                                                           */
/*      [ Abe 1 ]                                       Jun. 09, 1989        */
/*      Changed shape() routine using both units                             */
/*                                                                           */
/*      [ Abe 2 ]                                       Oct. 17, 1989        */
/*      Changed UnitAttribute structure                                      */
/*                                                                           */
/*      [ Abe 3 ]                                       Oct. 18, 1989        */
/*      Changed UnitAttribute structure                                      */
/*      [Takeda 2], Nov 13, 1989
        The input .oua file may includes a pause segment as an unit,
        To adapt this, some changes are needed.
 */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "/SYN/include/Synthesis.h"
#include "mkunit.h"

#define E                       stderr

/***** edited by K.Abe '89 Oct.03 ********************************************/
/*      'Max_unit' is defined in 'Synthesis.h'                               */
/***** edited by K.Abe '89 Oct.03 ********************************************/
/* #define Max_unit     512                                                  */
/***** edited by K.Abe '89 Oct.03 ********************************************/

#define Max_rules               256
#define Samp_Freq               12000.0         /* sampling frequency */

#define End_Edge                0
#define Start_Edge              1

char    *prog;

typedef struct
{
  int   end, start;
} Junction;

typedef struct
{
  int   stt;                    /* search start frame */
  int   end;                    /* serach end frame */

/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 **************************************/
  int   lmt_bw;                 /* limit backward search area from origin       */
  int   lmt_fw;                 /* limit forward search area from origin        */
```

```
/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ***********************************/
/*  int lmt;                                                                    */
/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ***********************************/

} SearchArea;                    /* Origin is unit boundary */

typedef struct
{
  int   end;                     /* code of previous unit end */
  int   fol;                     /* code of following phoneme of previous unit */
  int   pre;                     /* code of previous phoneme of current unit */
  int   stt;                     /* code of current unit start */
} PhonemeCode;

main( argc, argv )
int     argc;
char    *argv[];
{
  UnitAttribute u[Max_unit];     /* original unit attributes are
                                    loaded on this array.
                                    Boundaries will be modified by
                                    following proceduers */
  Junction      min_dist();
  FILE          *fp, *fopen();

  float         *cep_p;          /* previous unit cepstrum */
  float         *cep;            /* current unit cepstrum */
  float         *rd_cep();       /* read cepstrum */
  int           units;           /* total unit number */
  char          f[256];          /* unit filename */

  register int  i;               /* work */

/*------- check arguement consistency ----------------------------------------*/
  prog = argv[0];

  if( argc == 1 )
  {
    fprintf( E, "[%s] make synthesis unit files.\n", prog );
    fprintf( E, "\t[ USAGE ] %s filename header.\n", prog );
    exit( 0 );
  }

/*------- open old unit attribute file ---------------------------------------*/
  sprintf( f, "%s.oua", argv[1] );
  if( ( fp=fopen(f, "r") ) == NULL )
  {
    fprintf( E, "%s: old unit attribute file %s can't open.\n", prog, f );
    exit( 1 );
  }

/*------- read old unit attr%%[ PrinterError: out of paper ]%%  %%[ PrinterError: out
  if( ( units=GetUnitAttribute(fp, u) ) <= 0 )
  {
    fprintf( E, "%s: old unit attribute file %s read error.\n", prog, f );
    exit( 1 );
  }
  fclose( fp );

/*------- open write out unit file -------------------------------------------*/
  sprintf( f, "%s.ua", argv[1] );
  if( ( fp=fopen(f, "w") ) == NULL )
  {
    fprintf( E, "%s: modified unit attribute file %s can't open.\n", prog, f );
    exit( 1 );
```

```
  }

/*------- execute modification on each unit -----------------------------------*/
  for( i=0; i<units; i++ )
  {
    /* Skip pause segment */
    if( ! strcmp( u[i].roman, "pau" ))
         continue;

    printf( "\n***  %2d  ***  ", i );
    if( i == 0 ) printf( "\n" );

/*------- read unit's cepstrum data -------------------------------------------*/
    if( ( cep = rd_cep(u[i]) ) == (float*)NULL )
    {
      fprintf( E, "%s: can't read unit cepstrum file.\n", prog );
      exit( 2 );
    }


    /* [Takeda 2]
       the execution control has been changed
       to consider pause condidtion.
    if( i > 0 )
    {
      cep_p = rd_cep( u[i-1] );
    }
    deleted
    */

/*------- segmentating the unit out from word data ----------------------------*/

    /* If the preceding segment is 'Puase, no need to
       adjust beggining edge.*/

    if(( i == 0 ) || ( ! strcmp( u[i-1].roman, "pau" )))
      start_p( u[i], cep );
    else
    {
      cep_p = rd_cep( u[i-1] );          /* read cepstrum in previous unit */
      shape( u[i-1], u[i], cep_p, cep );
    }

    /* If following segmnet is 'Puase, no need to
       adjust ending edge. */
    if(( i == units-1 ) || ( ! strcmp( u[i+1].phn, "pau" ) ))
    {
      end_p( u[i], cep );
    }

/*------- cut fricative edge --------------------------------------------------*/
/*******************************************/
    fric_cut( fp, &u[i] );
/********************** 14 DEC 88 Sagisaka ***/

    free( cep );
    free( cep_p );
  }

/*------- write unit attribute data -------------------------------------------*/
  for( i=0; i<units; i++ )
  {
      PutUnitAttribute( fp, &u[i] );
  }
```

```
  exit( 0 );
}

/******** read cepstrum data ****************************************************/
/*                                                                              */
/*      u:      [in ] unit attribute data                                       */
/*                                                                              */
/********************************************************************************/
float   *rd_cep( u )
UnitAttribute   u;
{
  struct stat   buf;
  float         *cep;           /* cepstrum coefficients */
  int           fd;             /* file descriptor */
  char          f[256];         /* unit filename */

  sprintf( f, "/MHT/CEP/D%d/MHT_%04d.CEP", (u.word)/1000, u.word );

/*------- open file ------------------------------------------------------------*/
  if( ( fd=open(f, O_RDONLY) ) < 0 )
  {
    fprintf( E, "rd_cep: can't open cepstrum file %s.\n", f );
    return( (float*)NULL );
  }

/*------- get file status ------------------------------------------------------*/
  if( stat(f, &buf) != 0 )
  {
    fprintf( E, "rd_cep: can't know size of %s.\n", f );
    return( (float*)NULL );
  }
  cep = (float*)malloc( buf.st_size );

/*------- read cepstrum --------------------------------------------------------*/
  if( read(fd, cep, buf.st_size) != buf.st_size )
  {
    fprintf( E, "rd_cep: read cepstrum error.\n" );
    return( (float*)NULL );
  }

/***** edited by K.Abe '89, Sep., 18 ******************************************/
  close( fd );
/***** edited by K.Abe '89, Sep., 18 ******************************************/

  return( cep );
}

/******** segmentation ( re-shape ) the unit ************************************/
/*                                                                              */
/*      u_p:    [i/o] previous unit attribute data                              */
/*      u:      [i/o] current unit attribute data                               */
/*      cep_p:  [in ] previous unit cepstrum                                    */
/*      cep:    [in ] current unit cepstrum                                     */
/*                                                                              */
/********************************************************************************/
shape( u_p, u, cep_p, cep )
UnitAttribute   u_p, u;
float           *cep_p, *cep;
{
  SearchArea    fr;                     /* optimal junction point search area
                                           for front unit */
  SearchArea    bk;                     /* optimal junction point search area
                                           for back unit */
  unsigned int  func;
```

```
  unsigned int  check_phn();            /* check phoneme kinds */

/*-------------------------------------------------------------------------------*/
/*------- check phoneme around unit boundary ------------------------------------*/
/*-------------------------------------------------------------------------------*/

/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ***********************************/
  fr.lmt_bw = u_p.phn[u_p.phn_size-1]->end - u_p.phn[u_p.phn_size-1]->stt;
  bk.lmt_bw = u.pre->end - u.pre->stt;
  fr.lmt_fw = u_p.fol->end - u_p.fol->stt;
  bk.lmt_fw = u.phn[0]->end - u.phn[0]->stt;
/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ***********************************/
/*  fr.lmt = u_p.phn[u_p.phn_size-1]->end - u_p.phn[u_p.phn_size-1]->stt;       */
/*  bk.lmt = u.phn[0]->end - u.phn[0]->stt;                                     */
/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ***********************************/

  func = check_phn( Unit_end_phoneme(u_p), u_p.fol->phn, u.pre->phn, Unit_start_phone
                    &fr, &bk );

/*-------------------------------------------------------------------------------*/
/*------- search unit end & start point -----------------------------------------*/
/*-------------------------------------------------------------------------------*/

/***** [ Abe 3 ] edited by K.Abe '89 Oct. 18 ***********************************/
  switch( func )
  {
    case MinDist:
      min_distance( u_p, u, cep_p, cep, fr, bk );
      break;
    case MinEnergy:
      min_power( u_p, u, cep_p, cep, fr, bk );
      break;
    case MinLowPower:
      min_low_freq_power( u_p, u, cep_p, cep, fr, bk );
      break;
    default:
      end_p( u_p, cep_p );
      start_p( u, cep );
      break;
  }
/***** [ Abe 3 ] edited by K.Abe '89 Oct. 18 ***********************************/
/*  if( func == MinDist )                                                       */
/*  {                                                                           */
/*    end_stt_p( u_p, u, cep_p, cep, fr, bk );                                  */
/*  }                                                                           */
/*  else                                                                        */
/*  {                                                                           */
/*    end_p( u_p, cep_p );                                                      */
/*    start_p( u, cep );                                                        */
/*  }                                                                           */
/***** [ Abe 3 ] edited by K.Abe '89 Oct. 18 ***********************************/

}

/******** search start point ******************************************************/
/*                                                                                */
/*      u:      [i/o] current unit attribute data                                 */
/*      cep:    [in ] current unit cepstrum                                       */
/*                                                                                */
/**********************************************************************************/
start_p( u, cep )
UnitAttribute   u;
float           *cep;
{
  SearchArea    area;                   /* optimal junction point search area */
```

```
  int           pre, fol;              /* code of neighboring phoneme */
  int           onset, new_onset;      /* current old/new unit end frame */
  unsigned int  sel_func();            /* select criterion to re-shape
                                          according to context */
  unsigned int  func;                  /* selected criterion */

/*-------------------------------------------------------------------------*/
/*------- unit start point search -----------------------------------------*/
/*-------------------------------------------------------------------------*/
  pre = class( u.pre->phn, Start_Edge );
  fol = class( Unit_start_phoneme(u), Start_Edge );
  onset = Unit_start_frame( u );
  func = sel_func( pre, fol, &area );

  switch( func )
  {
    case MinEnergy:
      new_onset = min_pow( cep, onset, area ) + 1;
      break;
    case MinLowPower:
      new_onset = min_low_freq_pow( cep, onset, area ) + 1;
      break;
    case MaxDist:
      new_onset = max_change( cep, onset, area );
      break;
  }

/*------- type phoneme symbol ------------------ [ K.Abe, Jun.08.1989 ] --------*/
  printf( " %-13s                %3s", u.wd_roman, u.pre->phn );
  printf( " (%3d -> %3d) %3s ( %-10s )\n",
    onset, new_onset, Unit_start_phoneme(u), u.roman );
/*------- type phoneme symbol ------------------ [ K.Abe, Jun.08.1989 ] --------*/

  if( new_onset != onset )
  {
    u.phn[0]->stt = new_onset;
    u.phn[0]->eve[0]->stt = new_onset;
  }
}

/******** search end point ************************************************/
/*                                                                        */
/*      u_p:    [i/o] previous unit attribute data                        */
/*      cep_p:  [in ] previous unit cepstrum                              */
/*                                                                        */
/**************************************************************************/
end_p( u_p, cep_p )
UnitAttribute   u_p;             /* previous unit attribute data */
float           *cep_p;          /* previous unit cepstrum */
{
  SearchArea    area;                  /* optimal junction point search area */
  int           pre, fol;              /* code of neighboring phoneme */
  int           p_offset, p_new_offset; /* previous old/new unit end frame */
  int           onset, new_onset;      /* old/new unit start frame */
  unsigned int  sel_func();            /* select criterion to re-shape
                                          according to context */
  unsigned int  func;                  /* selected criterion */

/*-------------------------------------------------------------------------*/
/*------- unit end point search -------------------------------------------*/
/*-------------------------------------------------------------------------*/
  pre = class( Unit_end_phoneme(u_p), End_Edge );
  fol = class( u_p.fol->phn, End_Edge );
  p_offset = Unit_end_frame( u_p );
  func = sel_func( pre, fol, &area );
```

```
  switch( func )
  {
    case MinEnergy:
      p_new_offset = min_pow( cep_p, p_offset, area ) + 1;
      break;
    case MinLowPower:
      p_new_offset = min_low_freq_pow( cep_p, p_offset, area ) + 1;
      break;
    case MaxDist:
      p_new_offset = max_change( cep_p, p_offset, area );
  }

/*------- type phoneme symbol ------------------ [ K.Abe, Jun.08.1989 ] --------*/
  printf( " %-13s ( %-10s ) %3s (%3d -> %3d)",
    u_p.wd_roman, u_p.roman, Unit_end_phoneme(u_p), p_offset, p_new_offset );
  printf( " %3s\n", u_p.fol->phn );
/*------- type phoneme symbol ------------------ [ K.Abe, Jun.08.1989 ] --------*/

  if( p_new_offset != p_offset )
  {
    u_p.phn[u_p.phn_size-1]->end = p_new_offset;
    u_p.phn[u_p.phn_size-1]->eve[u_p.phn[u_p.phn_size-1]->eve_size-1]->end
      = p_new_offset;
  }
}

/******** search end & start point ***********************************************/
/*                                                                               */
/*      u_p:    [i/o] previous unit attribute data                               */
/*      u:      [i/o] current unit attribute data                                */
/*      cep_p:  [in ] previous unit cepstrum                                     */
/*      cep:    [in ] current unit cepstrum                                      */
/*      fr:     [in ] optimal point search area in front unit                    */
/*      bk:     [in ] optimal point search area in back unit                     */
/*                                                                               */
/*********************************************************************************/
min_distance( u_p, u, cep_p, cep, fr, bk )
UnitAttribute   u_p, u;
float           *cep_p, *cep;
SearchArea      fr, bk;
{
  Junction      min_dist(), junc;
  int           p_offset, p_new_offset; /* previous old/new unit end frame */
  int           onset, new_onset;       /* current old/new unit end frame */

/*-------------------------------------------------------------------------------*/
/*------- unit end & start point search -----------------------------------------*/
/*-------------------------------------------------------------------------------*/
  p_offset = Unit_end_frame( u_p );
  onset = Unit_start_frame( u );
  junc = min_dist( cep_p, cep, p_offset, onset, fr, bk );

/*------- type phoneme symbol ------------------ [ K.Abe, Jun.08.1989 ] --------*/
  printf( " %-13s ( %-10s ) %3s (%3d -> %3d)",
    u_p.wd_roman, u_p.roman, Unit_end_phoneme(u_p), p_offset, junc.end );
  printf( " %3s\n", u_p.fol->phn );
  printf( " %-13s                %3s", u.wd_roman, u.pre->phn );
  printf( " (%3d -> %3d) %3s ( %-10s )\n",
    onset, junc.start, Unit_start_phoneme(u), u.roman );
/*------- type phoneme symbol ------------------ [ K.Abe, Jun.08.1989 ] --------*/

  u_p.phn[u_p.phn_size-1]->end = junc.end+1;
  u_p.phn[u_p.phn_size-1]->eve[u_p.phn[u_p.phn_size-1]->eve_size-1]->end = junc.end+1
  u.phn[0]->stt = junc.start;
```

```
  u.phn[0]->eve[0]->stt = junc.start;
}

/******** search end & start point ( using minimum power ) **********************/
/*                                                                              */
/*      u_p:    [i/o] previous unit attribute data                              */
/*      u:      [i/o] current unit attribute data                               */
/*      cep_p:  [in ] previous unit cepstrum                                    */
/*      cep:    [in ] current unit cepstrum                                     */
/*      fr:     [in ] optimal point search area in front unit                   */
/*      bk:     [in ] optimal point search area in back unit                    */
/*                                                                              */
/********************************************************************************/
min_power( u_p, u, cep_p, cep, fr, bk )
UnitAttribute   u_p, u;
float           *cep_p, *cep;
SearchArea      fr, bk;
{
  int   p_offset, p_new_offset;         /* previous old/new unit end frame */
  int   onset, new_onset;               /* current old/new unit end frame */

/*------------------------------------------------------------------------------*/
/*------- unit end point search ------------------------------------------------*/
/*------------------------------------------------------------------------------*/
  p_offset = Unit_end_frame( u_p );
  p_new_offset = min_pow( cep_p, p_offset, fr ) + 1;

/*------- type phoneme symbol --------------------------------------------------*/
  printf( " %-13s ( %-10s ) %3s (%3d -> %3d)",
    u_p.wd_roman, u_p.roman, Unit_end_phoneme(u_p), p_offset, p_new_offset );
  printf( " %3s\n", u_p.fol->phn );

  if( p_new_offset != p_offset )
  {
    u_p.phn[u_p.phn_size-1]->end = p_new_offset;
    u_p.phn[u_p.phn_size-1]->eve[u_p.phn[u_p.phn_size-1]->eve_size-1]->end
      = p_new_offset;
  }

/*------------------------------------------------------------------------------*/
/*------- unit start point search ----------------------------------------------*/
/*------------------------------------------------------------------------------*/
  onset = Unit_start_frame( u );
  new_onset = min_pow( cep, onset, bk ) + 1;

/*------- type phoneme symbol --------------------------------------------------*/
  printf( " %-13s                %3s", u.wd_roman, u.pre->phn );
  printf( " (%3d -> %3d) %3s ( %-10s )\n",
    onset, new_onset, Unit_start_phoneme(u), u.roman );

  if( new_onset != onset )
  {
    u.phn[0]->stt = new_onset;
    u.phn[0]->eve[0]->stt = new_onset;
  }
}

/******** search end & start point ( using min. low frequency spectrum power ) **/
/*                                                                              */
/*      u_p:    [i/o] previous unit attribute data                              */
/*      u:      [i/o] current unit attribute data                               */
/*      cep_p:  [in ] previous unit cepstrum                                    */
/*      cep:    [in ] current unit cepstrum                                     */
/*      fr:     [in ] optimal point search area in front unit                   */
/*      bk:     [in ] optimal point search area in back unit                    */
```

```
/*                                                                              */
/********************************************************************************/
min_low_freq_power( u_p, u, cep_p, cep, fr, bk )
UnitAttribute   u_p, u;
float           *cep_p, *cep;
SearchArea      fr, bk;
{
  int   p_offset, p_new_offset;         /* previous old/new unit end frame */
  int   onset, new_onset;               /* current old/new unit end frame */

/*------------------------------------------------------------------------------*/
/*------- unit end point search ------------------------------------------------*/
/*------------------------------------------------------------------------------*/
  p_offset = Unit_end_frame( u_p );
  p_new_offset = min_low_freq_pow( cep_p, p_offset, fr ) + 1;

/*------- type phoneme symbol --------------------------------------------------*/
  printf( " %-13s ( %-10s ) %3s (%3d -> %3d)",
    u_p.wd_roman, u_p.roman, Unit_end_phoneme(u_p), p_offset, p_new_offset );
  printf( " %3s\n", u_p.fol->phn );

  if( p_new_offset != p_offset )
  {
    u_p.phn[u_p.phn_size-1]->end = p_new_offset;
    u_p.phn[u_p.phn_size-1]->eve[u_p.phn[u_p.phn_size-1]->eve_size-1]->end
      = p_new_offset;
  }

/*------------------------------------------------------------------------------*/
/*------- unit start point search ----------------------------------------------*/
/*------------------------------------------------------------------------------*/
  onset = Unit_start_frame( u );
  new_onset = min_low_freq_pow( cep, onset, bk ) + 1;

/*------- type phoneme symbol --------------------------------------------------*/
  printf( " %-13s                %3s", u.wd_roman, u.pre->phn );
  printf( " (%3d -> %3d) %3s ( %-10s )\n",
    onset, new_onset, Unit_start_phoneme(u), u.roman );

  if( new_onset != onset )
  {
    u.phn[0]->stt = new_onset;
    u.phn[0]->eve[0]->stt = new_onset;
  }
}

/******** select function ******************************************************/
/*                                                                              */
/*      pre:    [in ] previous phoneme class                                    */
/*      fol:    [in ] following phoneme class                                   */
/*      area:   [out] optimal junction point search area                        */
/*                                                                              */
/********************************************************************************/
unsigned int    sel_func( pre, fol, area )
int             pre, fol;
SearchArea      *area;
{
  int   func;

  if( pre == Vowels )
  {
    switch( fol )
    {
      case Vowels:                      /* a, i, u, e, o, N */
      case Nasals:                      /* m, n, g */
```

```
      case VoicedStops:                 /* b, d */
      case VoicedAffricates:            /* j */
      case VoicedFricatives:            /* z */
        func = MaxDist;
        area->stt = -5; area->end = 5;
        break;
      case SemiVowels:                  /* r, w, y */
        func = MaxDist;
        area->stt = -5; area->end = 2;
        break;
      case VoicelessStops:              /* p, t, k */
      case VoicelessAffricates:         /* ts, ch */
        func = MinEnergy;
        area->stt = -3; area->end = 5;
        break;
      case VoicelessFricatives:         /* s, sh, h, f */
        func = MinLowPower;
        area->stt = -5; area->end = 5;
        break;
      default:
        func = MinEnergy;
        area->stt = -5; area->end = 5;
        break;
    }
  }
  else
  {
    func = MaxDist;
    area->stt = -5;     area->end = 5;
  }
  return( func );
}

/******** check phoneme coincidence around junction ****************************/
/*                                                                             */
/*      u_en:   [in ] end phoneme of previous                                  */
/*      u_fol:  [in ] following phoneme of previous unit                       */
/*      u_pre:  [in ] previous phoneme of current unit                         */
/*      u_st:   [in ] start phoneme of current unit                            */
/*      fr:     [out] optimal point search area in front unit                  */
/*      bk:     [out] optimal point search area in back unit                   */
/*                                                                             */
/*******************************************************************************/
unsigned int    check_phn( u_en, u_fol, u_pre, u_st, fr, bk )
char            *u_en, *u_fol, *u_pre, *u_st;
SearchArea      *fr, *bk;
{
  PhonemeCode   code;
  int           junc;
  int           flag_fr, flag_bk, flag_vw, flag_fN, flag_bn;
  unsigned int  junc_0(), junc_1(), junc_2(), junc_3(), junc_4();
  unsigned int  func;
  char          *top(), *tail();

/*------- phoneme coincidence around junction ----------------------------------*/
  flag_fr = strcmp( u_en, u_pre );
  flag_bk = strcmp( top(u_fol), top(u_st) );
  flag_vw = strcmp( tail(u_en), top(u_st) );

  if( flag_fr == 0 && flag_bk == 0 ) junc = SamePhoneme;
  if( flag_fr == 0 && flag_bk != 0 ) junc = SameFrontPhoneme;
  if( flag_fr != 0 && flag_bk == 0 ) junc = SameBackPhoneme;
  if( flag_fr != 0 && flag_bk != 0 && flag_vw == 0 ) junc = SameVowel;

  if( flag_fr != 0 && flag_bk != 0 && flag_vw != 0 ) junc = DifferentPhoneme;
```

```
/******** '89.9/08 ***************** Specials***********************************/
  flag_fN = strcmp( u_en, "N" );
  flag_bn = strcmp( u_st, "n" );
  if( flag_fN == 0 && flag_bn == 0 ) junc = SameVowel;


/*  [Takeda Oct 23, 1989]
    To Adapt the case when
    preceding unit is NnO and
    following unit is osa
*/

  if( checkSameFrontPhoneme( u_en, u_pre ) )
       junc = SameFrontPhoneme;

/******** edited by k.abe ************************************************/

/*------- classify each phoneme ----------------------------------------------*/


 Next:
  code.end = class( u_en, End_Edge );
  code.fol = class( u_fol, End_Edge );
  code.pre = class( u_pre, Start_Edge );
  code.stt = class( u_st, Start_Edge );

/*------- type out phoneme coincidence around junction -----------------------*/
  wr_junc( junc );

/*------- decide function ----------------------------------------------------*/
  switch( junc )
  {
    case SamePhoneme:
      func = junc_0( code, fr, bk );
      break;
    case SameFrontPhoneme:
      func = junc_1( code, fr, bk );
      break;
    case SameBackPhoneme:
      func = junc_2( code, fr, bk );
      break;
    case SameVowel:
      func = junc_3( code, fr, bk );
      break;
    case DifferentPhoneme:
      func = junc_4( code, fr, bk );
      break;
  }

  return( func );
}

/******** same phoneme  ***************************************************/
/*                                                                        */
/*      code:    [in ] phoneme code                                       */
/*      fr:      [out] optimal point search area in front unit            */
/*      bk:      [out] optimal point search area in back unit             */
/*                                                                        */
/**************************************************************************/
unsigned int     junc_0( code, fr, bk )
PhonemeCode      code;
SearchArea       *fr, *bk;
{
```

```
  unsigned int  func;

/*------- decide shape method ---------------------------------------------------*/
  if( code.end == Vowels )
  {
    switch( code.fol )
    {
      case VoicelessStops:              /* p, t, k*/
      case VoicelessAffricates:         /* ts, ch */
        func = MinEnergy;
        fr->stt = -5;   fr->end = 5;
        bk->stt = -5;   bk->end = 5;
        break;
      case VoicelessFricatives:         /* s, sh, h, f */
        func = MinLowPower;
        fr->stt = -5;   fr->end = 5;
        bk->stt = -5;   bk->end = 5;
        break;
      case Nasals:                      /* m, n, g */
      case Vowels:                      /* a, i, u, e, o, N */
      case SemiVowels:                  /* r, w, y */
      case VoicedAffricates:            /* j */
      case VoicedFricatives:            /* z */
      case VoicedStops:                 /* b, d */
        func = MinDist;
        fr->stt = -5;   fr->end = 5;
        bk->stt = -5;   bk->end = 5;
        break;
    }
  }
  else
  {
    func = MinDist;
    fr->stt = -5;               fr->end = 5;
    bk->stt = -5;               bk->end = 5;
  }
  return( func );
}

/******** same front phoneme *****************************************************/
/*                                                                                */
/*      code:   [in ] phoneme code                                                */
/*      bw:     [out] backward search area                                        */
/*      fw:     [out] forward search area                                         */
/*                                                                                */
/**********************************************************************************/
unsigned int    junc_1( code, fr, bk )
PhonemeCode     code;
SearchArea      *fr, *bk;
{
  unsigned int  func;

/*------- decide shape method ---------------------------------------------------*/
  if( code.end == Vowels )
  {
    func = MinDist;

/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ***************************************/
    fr->stt = -fr->lmt_bw/2;            fr->end = -1;
    bk->stt = -bk->lmt_bw/2;            bk->end = -1;
/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ***************************************/
/*    fr->bw = -fr->lmt/2;              fr->fw = -1;                              */
/*    bk->bw = -bk->lmt/2;              bk->fw = -1;                              */
/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ***************************************/
```

```
  }
  else
  {
    func = MinDist;

/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ************************************/
    fr->stt = -5;                       fr->end = -1;
    bk->stt = -5;                       bk->end = -1;
/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ************************************/
/*    fr->bw = -5;                      fr->fw = 5;                               */
/*    bk->bw = -5;                      bk->fw = 5;                               */
/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ************************************/

  }
  return( func );
}

/******** same back phoneme ***********************************************************/
/*                                                                                    */
/*      code:   [in ] phoneme code                                                    */
/*      bw:     [out] backward search area                                            */
/*      fw:     [out] forward search area                                             */
/*                                                                                    */
/**************************************************************************************/
unsigned int    junc_2( code, fr, bk )
PhonemeCode     code;
SearchArea      *fr, *bk;
{
  unsigned int  func;

/*------- decide shape method ---------------------------------------------------*/

/***** [ Abe 3 ] edited by K.Abe '89 Oct. 18 ************************************/
  switch( code.fol )
  {
    case VoicelessStops:                /* p, t, k */
    case VoicelessAffricates:           /* ts, ch */
      func = MinEnergy;
      fr->stt = 1;      fr->end = 5;
      bk->stt = 1;      bk->end = 5;
      break;
    case VoicelessFricatives:           /* s, sh, h, f */
      func = MinLowPower;
      fr->stt = 1;      fr->end = 5;
      bk->stt = 1;      bk->end = 5;
      break;
    case Nasals:                        /* m, n, g */
    case Vowels:                        /* a, i, u, e, o, N */
    case SemiVowels:                    /* r, w, y */
    case VoicedAffricates:              /* j */
    case VoicedFricatives:              /* z */
    case VoicedStops:                   /* b, d */
      func = MinDist;
      fr->stt = 1;      fr->end = fr->lmt_fw/2;
      bk->stt = 1;      bk->end = bk->lmt_fw/2;
      break;
    default:
      func = MinDist;
      fr->stt = 1;      fr->end = 5;
      bk->stt = 1;      bk->end = 5;
      break;
  }
/***** [ Abe 3 ] edited by K.Abe '89 Oct. 18 ************************************/
/*  if( code.fol == Nasals || code.fol == Vowels )                                */
/*  {                                                                             */
```

```
/*    func = MinDist;                                                    */
/*    fr->bw = 1;              fr->fw = fr->lmt_bw/2;                    */
/*    bk->bw = 1;              bk->fw = bk->lmt_bw/2;                    */
/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ***********************************/
/**    fr->bw = 1;              fr->fw = fr->lmt/2;                      **/
/**    bk->bw = 1;              bk->fw = bk->lmt/2;                      **/
/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ***********************************/
/*  }                                                                    */
/*  else                                                                 */
/*  {                                                                    */
/*    func = MaxDist;                                                    */
/*    fr->bw = 1;              fr->fw = 5;                               */
/*    bk->bw = 1;              bk->fw = 5;                               */
/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ***********************************/
/**    fr->bw = -5;     fr->fw = 5;                                      **/
/**    bk->bw = -5;     bk->fw = 5;                                      **/
/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ***********************************/
/*  }                                                                    */
/***** [ Abe 3 ] edited by K.Abe '89 Oct. 18 ***********************************/

  return( func );
}

/******** same vowel ************************************************************/
/*                                                                          */
/*      code:   [in ] phoneme code                                          */
/*      bw:     [out] backward search area                                  */
/*      fw:     [out] forward search area                                   */
/*                                                                          */
/****************************************************************************/
unsigned int    junc_3( code, fr, bk )
PhonemeCode     code;
SearchArea      *fr, *bk;
{
  unsigned int  func;

/*------- decide shape method ------------------------------------------------*/
  func = MinDist;

/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ***********************************/
  fr->stt = -fr->lmt_bw/2;              fr->end = -1;
  bk->stt = 1;                          bk->end = bk->lmt_fw/2;
/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ***********************************/
/*  fr->bw = -fr->lmt/2;                fr->fw = -2;                      */
/*  bk->bw = 1;                         bk->fw = bk->lmt/2;               */
/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ***********************************/

  return( func );
}

/******** different phoneme *****************************************************/
/*                                                                          */
/*      code:   [in ] phoneme code                                          */
/*      bw:     [out] backward search area                                  */
/*      fw:     [out] forward search area                                   */
/*                                                                          */
/****************************************************************************/
unsigned int    junc_4( code, fr, bk )
PhonemeCode     code;
SearchArea      *fr, *bk;
{
  unsigned int  func;
  char  *top(), *tail();

/*------- decide shape method ------------------------------------------------*/
```

```
  if( code.end == Vowels )
  {
    func = MaxDist;
    fr->stt = -5;                fr->end = 5;
    bk->stt = -5;                bk->end = 5;
  }
  else
  {
    func = MaxDist;
    fr->stt = -5;                fr->end = 5;
    bk->stt = -5;                bk->end = 5;
  }
  return( func );
}

/******** classify phoneme ***************************************************/
/*                                                                           */
/*      s:              [in ] phoneme symbol                                 */
/*      se_flag:        [in ]                                                */
/*                                                                           */
/*****************************************************************************/
class( s, se_flag )
char    *s;
int     se_flag;
{
  static struct
  {
    int         phns;                   /* number of phonemes in the class */
    int         class;                  /* phoneme class identification number */
    char        *phn[10];               /* phoneme symbols */
  } tab[10] =
  {
    { 6,        Vowels,                 { "a","i","u","e","o","N","","","","" } },
    { 3,        SemiVowels,             { "r","w","y","","","","","","","" } },
    { 3,        Nasals,                 { "m","n","g","","","","","","","" } },
    { 2,        VoicedStops,            { "b","d","","","","","","","","" } },
    { 1,        VoicedAffricates,       { "j","","","","","","","","","" } },
    { 1,        VoicedFricatives,       { "z","","","","","","","","","" } },
    { 3,        VoicelessStops,         { "p","t","k","","","","","","","" } },
    { 2,        VoicelessAffricates,    { "ts","ch","","","","","","","","" } },
    { 4,        VoicelessFricatives,    { "s","sh","h","f","","","","","","" } },
    { 2,        Edges,                  { "Top","End","","","","","","","","" } }
  };

  int           classes = 10;           /* number of classes */
  register int  i, j;
  char          *top(), *tail();

  if( size(s) > 1 )
  {
    if( se_flag == Start_Edge ) return( class(top(s), se_flag) );
    else return( class(tail(s), se_flag) );
  }
  else
  {
    for( i=0; i<classes; i++ )
    {
      for( j=0; j<tab[i].phns; j++ )
        if( strcmp(s, tab[i].phn[j]) == 0 ) return( tab[i].class );
    }
    return( Unknown );
  }
}

/******** detect energy minimum frame ****************************************/
```

```
/*                                                                          */
/*      cep:    [in ] cepstrum                                              */
/*      offset: [in ] unit end frame                                        */
/*      area:   [in ] optimal junction point search area                    */
/*                                                                          */
/**************************************************************************/
min_pow( cep, offset, area )
float           *cep;
int             offset;
SearchArea      area;
{
  float         min = 10000000.0;
  int           min_p;
  register int  i = 0;

  printf( " [ Min. E   ] " );
  for( i=offset+area.stt; i<=offset+area.end; i++ )
  {
    if( *(cep+i*32+1) < min )
    {
      min = *(cep+i*32+1);
      min_p = i;
    }
  }
  return( min_p );
}

/******** detect low frequency spectrum power minimum frame *****************/
/*                                                                          */
/*      cep:    [in ] cepstrum                                              */
/*      offset: [in ] unit end frame                                        */
/*      area:   [in ] optimal junction point search area                    */
/*                                                                          */
/**************************************************************************/
min_low_freq_pow( cep, offset, area )
float           *cep;
int             offset;
SearchArea      area;
{
  float         band_pwr();
  float         min = 10000000.0;
  float         low_pwr;
  int           min_p;
  register int  i = 0;

  printf( " [ Min. L.E ] " );
  for( i=offset+area.stt; i<=offset+area.end; i++ )
  {
    if( ( low_pwr=band_pwr(cep+i*32, 50.0, 100.0) ) < min )
    {
      min = low_pwr;
      min_p = i;
    }
  }
  return( min_p );
}

/******** detect maximum spectral changing frame ****************************/
/*                                                                          */
/*      cep:    [in ] cepstrum                                              */
/*      offset: [in ] unit end frame                                        */
/*      area:   [in ] optimal junction point search area                    */
/*                                                                          */
/**************************************************************************/
max_change( cep, offset, area )
```

```
float           *cep;
int             offset;
SearchArea      area;
{
  float         max = 0.0;
  float         dist;                   /* spectral distance */
  float         cep_dist();
  int           max_p;
  register int  i = 0;

  printf( " [ Max. S.C ] " );
  for( i=offset+area.stt; i<=offset+area.end; i++ )
  {
    if( ( dist=cep_dist(cep+i*32, cep+(i-1)*32) ) > max )
    {
      max = dist;
      max_p = i;
    }
  }
  return( max_p );
}

/******** detect spectral distance minimum frame between units ******************/
/*                                                                              */
/*      cep_p:          [in ] previous cepstrum                                 */
/*      cep:            [in ] cepstrum                                          */
/*      p_offset:       [in ] previous unit end frame                           */
/*      onset:          [in ] unit start frame                                  */
/*      fr:             [in ] optimal point search area in front unit           */
/*      bk:             [in ] optimal point search area in back unit            */
/*                                                                              */
/********************************************************************************/
Junction        min_dist( cep_p, cep, p_offset, onset, fr, bk )
float           *cep_p, *cep;
int             p_offset, onset;
SearchArea      fr, bk;
{
  Junction      junc;
  float         dist;                   /* spectral distance */
  float         cep_dist();
  int           i1, i2, j1, j2, min_i, min_j;
  float         min1 = 100., min2 = 100., min_d = 100.;
  register int  i=0, j=0;

/***** edited by K.Abe '89 Sep.24 ***********************************************/
  if( ( fr.stt < 0 ) && ( fr.end > 0 ) && ( bk.stt < 0 ) && ( bk.end > 0 ) )
  {
    for( i=p_offset+fr.stt; i<=p_offset-1; i++ )
        for( j=onset+bk.stt; j<=onset-1; j++ )
            if( ( dist=cep_dist(cep_p+i*32, cep+j*32) ) < min1 )
            {
              min1 = dist;
              i1 = i;
              j1 = j;
            }
    for( i=p_offset; i<=p_offset+fr.end; i++ )
        for( j=onset; j<=onset+bk.end; j++ )
            if( ( dist=cep_dist(cep_p+i*32, cep+j*32) ) < min2 )
            {
              min2 = dist;
              i2 = i;
              j2 = j;
            }
    if( min1 < min2 )
    {
```

```
      min_d = min1;
      min_i = i1;
      min_j = j1;
    }
    else
    {
      min_d = min2;
      min_i = i2;
      min_j = j2;
    }
  }
  else
       for( i=p_offset+fr.stt; i<=p_offset+fr.end; i++ )
            for( j=onset+bk.stt; j<=onset+bk.end; j++ )
                 if( ( dist=cep_dist(cep_p+i*32, cep+j*32) ) < min_d )
                 {
                   min_d = dist;
                   min_i = i;
                   min_j = j;
                 }
  junc.end = min_i;
  junc.start = min_j;

/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ***********************************/
  printf( "\n  +++ [offset]=%3d ( %3d :%3d )  [onset]=%3d ( %3d :%3d )\n",
          p_offset, fr.stt, fr.end, onset, bk.stt, bk.end );
  printf( "  +++ Min. Dist. [1] = %5.2f ( %3d >> %3d )\n", min_d, min_i, min_j );
/***** [ Abe 2 ] edited by K.Abe '89 Oct. 17 ***********************************/

/*****************************************************************************/
/*  for( i=p_offset+fr.bw; i<=p_offset+fr.fw; i++ )                          */
/*       for( j=onset+bk.bw; j<=onset+bk.fw; j++ )                           */
/*          if( ( dist=cep_dist(cep_p+i*32, cep+j*32) ) < min1 )             */
/*          {                                                                */
/*            min2 = min1;      min1 = dist;                                 */
/*            i2 = i1;  i1 = i;                                              */
/*            j2 = j1;  j1 = j;                                              */
/*          }                                                                */
/*  junc.end = i1;                                                           */
/*  junc.start = j1;                                                         */
/***** edited by K.Abe '89 Sep.24 **********************************************/

  return( junc );
}

/******** extract band power *************************************************/
/*                                                                           */
/*      cep:    [in ] cepstrum                                               */
/*      fl:     [in ] lower limit freq. of pitch freq. search range          */
/*      fu:     [in ] upper limit freq. of pitch freq. search range          */
/*                                                                           */
/*---------------------------------------------------------------------------*/
/*                              Coded by Katsuo Large Abe                    */
/*****************************************************************************/
float   band_pwr( cep, fl, fu )
float   *cep;
float   fl, fu;
{
  double        w12, w21, w212, sum, ww;
  double        bn;                             /* band power */
  double        pi = 3.14159265358979323846;
  register int  i;

  w12 = 2.*pi*(fl+fu)/Samp_Freq;
  w21 = 2.*pi*(fu-fl)/Samp_Freq;
```

```
  w12 /= 2.0;
  w212 = w21/2.;
  for( i=0; i<256; i++ )                cep[i] *= 0.5;
  sum = 0.;
  for( i=1; i<=30; i++ )
  {
    sum += ( *(cep+i) * cos((double)i*w12) * sin((double)i*w212) ) / w21 / (double)i;
  }
  bn = cep[0] + 4.* sum;
  return( (float)bn );
}

/******** cepstrum distance ( including power term ) ***************************/
/*                                                                             */
/*      cep1:   [in ] cepstrum1                                                */
/*      cep2:   [in ] cepstrum2                                                */
/*                                                                             */
/*******************************************************************************/
float   cep_dist( cep1, cep2 )
float   cep1[], cep2[];
{
  float         sum = 0.0;
  float         d_cep;
  register int  i;

  for( i=1; i<32; i++ )
  {
    d_cep = cep1[i] - cep2[i];
    sum += d_cep * d_cep;
  }
  return( sum );
}

/******** get top phoneme ******************************************************/
/*                                                                             */
/*      s:      [in ] phoneme symbol                                           */
/*                                                                             */
/*******************************************************************************/
char    *top( s )
char    *s;
{
  char          *ptr;
  register int  i;

  ptr = (char*)malloc( strlen(s)+1 );
  strcpy( ptr, s );
  for( i=0; i<strlen(ptr); i++ )
  {
    if( *(ptr+i) == ',' )
    {
      *(ptr+i) = '\0';
      break;
    }
  }
  if( *(ptr+i-1) == 'y' ) *(ptr+i-1) = '\0';
  return( ptr );
}

/******** get tail phoneme ******************************************************/
/*                                                                             */
/*      s:      [in ] phoneme symbol                                           */
/*                                                                             */
/*******************************************************************************/
char    *tail( s )
char    *s;
```

```
{
  register int  i;
  char          *ptr;

  for( i=strlen(s)-1; i>=0; i-- )
  {
    ptr = s+i;
    if( *(s+i) == ',' )
    {
      ptr++;
      break;
    }
  }
  if( *(ptr+strlen(ptr)-1) == 'y' ) return( ptr+strlen(ptr)-1 );
  return( ptr );
}

/******** get size phoneme ************************************************/
/*                                                                        */
/*      s:      [in ] phoneme symbol                                      */
/*                                                                        */
/**************************************************************************/
int     size( s )
char    *s;
{
  char  c;
  int   count = 1;

  while( c = *s++ )     if( c == ',' ) count++;
  return( count );
}

/******** write phoneme coincidence around junction ***********************/
/*                                                                        */
/*      junc:   [in ] code for phoneme coincidence around junction       */
/*                                                                        */
/**************************************************************************/
wr_junc( junc )
int     junc;
{
  switch( junc )
  {
    case SamePhoneme:
      printf( " [ SamePhoneme ]\n" );
      break;
    case SameFrontPhoneme:
      printf( " [ SameFrontPhoneme ]\n" );
      break;
    case SameBackPhoneme:
      printf( " [ SameBackPhoneme ]\n" );
      break;
    case SameVowel:
      printf( " [ SameVowel ]\n" );
      break;
    case DifferentPhoneme:
      printf( " [ DifferentPhoneme ]\n" );
      break;
  }
}

static int read_size = 0;
static struct
{
  char *u_en;
  char *u_pre;
```

```
} same_ph_table[256];

checkSameFrontPhoneme( s1, s2 )
char *s1, *s2;
{
  register int i;

  if( read_size == 0 )
       if(( read_size = read_table( "/SYN/Parms/same-ph-tab" )) < 0 )
       {
         fprintf( stderr, "can't read table SamePhonemeTable\n");
         exit(-1);
       }


  for( i = 0; i < read_size; i++ )
  {
    if(( strcmp( s1, same_ph_table[i].u_en ) == 0 )
       &&
       ( strcmp( s2, same_ph_table[i].u_pre ) == 0 ))
         return(1);
  }
  return(0);
}

read_table(file)
char *file;
{
  FILE *fp, *fopen();
  register int i = 0;
  char line[256];
  char en[20], pre[20];

  if(( fp = fopen( file, "r" )) == (FILE*)NULL )
  {
    fprintf( stderr, "file: %s, can't open.\n", file );
    return(-1);
  }

  while( fgets( line, 256, fp ) != 0 )
  {
    if( line[0] == '#' )
         continue;
    sscanf( line, "%s %s", en, pre );

    same_ph_table[i].u_en = (char*)malloc(strlen(en)+1);
    same_ph_table[i].u_pre = (char*)malloc(strlen(pre)+1);

    strcpy( same_ph_table[i].u_en, en );
    strcpy( same_ph_table[i].u_pre, pre );
    i++;
  }
  return(i);
}
```

```
/*
  durset.c

  Duration data generate routine.

  Origianlly coded by K. KIN Takeda.            [Jul 15, 1988] ATR
  >> History >>
  [Takeda 1]
     Shorten inseparable portion's duration     Jul. 18, 1988
  [Takeda 2]
     For the using of new data structures       Jul. 28, 1988
     and adding information write out routine
  [Yamazaki 1]                                  Aug.  7, 1988
     Shorttening/Lengthening ratio is limited between
     0.5 to 1.5.
  [Takeda 3, Aug 23, 1989]
     Processing of inseparable portions are improved.
     A reconstruction was made to adapt to the
     new junction controlling.
  [Takeda 4, Sep 5, 1989]
     Minor revision for bug fix of the [Takeda 3] concerned with
     Inf field information of Phoneme table.
  [Abe 1, Sep 25, 1989]
     Minor revision for bug fix of match_dr_and_ua routine.
  [Takeda 5, Oct 3, 1989]
     Minor revision for bug fix of the [Abe 1]
  [Takeda 6, Oct 26, 1989]
     To match complicated things
*/

#include <stdio.h>
#include <fcntl.h>
#include "durset.h"
#include "/SYN/include/Synthesis.h"

#define max(A,B)     ((A)>(B)?(A):(B))

#define END                     1
#define E                       stderr

/***** edited by K.Abe '89 Sep. 22 ******************************
#define Max_unit                512
#define Max_phoneme             2048                    Max_unit */
/***** edited by K.Abe '89 Sep. 22 ******************************/

#define PhPause                 1
#define PhUsualMatch            2
#define PhFusion                4
#define PhDuplicate             8
#define PhDuplicatedBefore     16

#define MaxLength               1.5
#define MinLength               0.5

typedef struct
          { int stt, end, Top, End; } Margin;
typedef struct
          {
            int unit, phn, size, durtabp[10];
            float ratio;
            unsigned int inf;   /* [Takeda 4] */
          } WorkTable;

typedef struct
          {
```

```
            int size;
            int phrase[4], phoneme[4];
          } Revtab;

char *prog;
int total_unit_size;
float frame_len;

main(argc,argv)
int argc;
char *argv[];
{

/***** edited by K.Abe '89 Sep. 22 ******************************/
  UnitAttribute ua[Max_unit];           /* read from .ua file */
  UnitAttribute oua[Max_unit];          /* read from .oua file */
  UnitAttribute *new[Max_unit];
  DurationData dr[Max_phoneme];         /* read from .DR file */
  Phoneme pn[Max_phoneme];              /* for output .PN file */
  Margin margin[Max_unit];
  WorkTable work[Max_unit];
  Revtab rvtab[Max_unit];
/*************************************************************/
/*  UnitAttribute ua[64];                                    */
/*  UnitAttribute oua[64];                                   */
/*  UnitAttribute *new[64];                                  */
/*  DurationData dr[256];                                    */
/*  Phoneme pn[256];                                         */
/*  Margin margin[64];                                       */
/*  WorkTable work[128];                                     */
/***** edited by K.Abe '89 Sep. 22 ******************************/

  int dr_size;                  /* size of duration file (xxx.DR) */
  int ua_size, oua_size;        /* size of unit */
  int pn_size;                  /* size of pn */
  int phonemes;

  prog = argv[0];

  if(argc==1)
  {
    fprintf(E,"[%s] duration reset.\n", prog );
    fprintf(E,"\tUsage: %s file-name-header.\n", prog );
    exit(0);
  }

  if(( dr_size = ReadDuration( argv[1], dr )) <= 0 )
  {
    fprintf(E,"%s: duration information file read error.\n", prog );
    exit(1);
  }

  /* ----------------- Read unit attribute file-------------------------  */
  if(( ua_size = ReadUnitAttribute( argv[1], ua )) <= 0 )
  {
    fprintf(E,"%s: unit attribute file read error.\n", prog );
    exit(1);
  }

  /* ------------------ Read original unit attribute file ----------------- */
  if(( oua_size = ReadOriginalUnitAttribute( argv[1], oua )) <= 0 )
  {
    fprintf(E,"%s: original unit attribute file read error.\n", prog );
    exit(1);
  }
```

```
  /* ------------------------ Chech size of units ------------------------ */
  if( oua_size != ua_size )
  {
    fprintf( E,
     "%s: unit size of xxx.ua file and xxx.oua file is not coinside.\n",
     prog );
    exit(1);
  }
  /* ------------ Check how much the boundaries were moved ---------------- */
  if(( ua_size = get_margin( margin, oua, ua, oua_size, ua_size, new ))
     < 0 )
  {
    fprintf( E, "%s: error in get_margin\n", prog );
    exit(1);
  }


  /* ---------- Compare .DR file and .ua file ---------------------------- */
  if(( phonemes = match_dr_and_ua( margin,
                                   new,
                                   work,
                                   ua_size,
                                   dr_size, dr )) <= 0 )
  {
    fprintf(E,"%s: .DR and .ua file match error.\n", prog );
    exit(2);
  }

/*
  debug_print( new, work, phonemes, dr );

  if( generate_revtab( phonemes, work, revtab ) <= 0 )
  {
    fprintf( E, "%s\n: revtab generation error.\n", prog );
    exit(2);
  }
*/
  /* ----------- set phoneme duration and supplementary attributes ------- */
  if(( pn_size = set_phoneme_data( work, phonemes,
                                   new, ua_size,
                                   dr, dr_size,
                                   pn )) <= 0 )
  {
    fprintf(E,"%s: phoneme duration set error.\n", prog );
    exit(2);
  }

  /* ---------- Write out xxx.unit file --------------------------------- */
  if( WritePhoneme( argv[1], phonemes, pn ) <= 0 )
  {
    fprintf(E,"%s: Write faild.\n", prog );
    exit(3);
  }

  /* --------- Generate xxx.PN file contents --------------------------- */
  if( create_phn_inf_file( phonemes, argv[1], pn, work ) <= 0 )
  {
    fprintf(E,"%s: Phoneme inf. file generate error.\n", prog );
    exit(4);
  }

  /* --------- Write out xxx.FR file ----------------------------------- */
  if( create_frm_inf_file( phonemes, argv[1], pn ) <= 0 )
  {
```

```
    fprintf(E,"%s: Frame inf. file generate error.\n", prog );
    exit(5);
  }

  /* done */
  exit(0);
}

get_margin( margin, oua, ua, oua_size, ua_size, new )
Margin margin[];
UnitAttribute ua[], oua[], *new[];
int oua_size, ua_size;
/*
  [Aug 23, 1989 Takeda]
   This program calculates how much edges of each unit
   were moved to the BACKward direction
   from original label segmentation, returns the
   value through margin.
*/
{
  register int i, index;
  UnitAttribute *new_ua, *read_label();
  int stt, end;
  int IsTop, IsEnd, IsHeadDup, IsLastDup;
  int TopShouldMoved, EndShouldMoved;

  index = 0;

  for( i = 0; i < oua_size; i++ )
  {
    /* Skip pause segment */
    if( ! strcmp( oua[i].roman, "pau" ))
         continue;

    stt =
    margin[index].stt = Unit_start_frame(oua[i]) - Unit_start_frame(ua[i]);
    end =
    margin[index].end = Unit_end_frame(oua[i]) - Unit_end_frame(ua[i]);

  /* Set Predicates */
    /* Is the unit start from the beggining of the word ? */

/***** edited by K.Abe '89 Oct. 17 ********************************************/
/* [Takeda 7 ] */
    if(( i > 0 && ( ! strcmp( oua[i-1].roman, "pau" )))
       ||
       ( ! strcmp( oua[i].pre -> phn, "Top" )))
         IsTop = 1;
    else
         IsTop = 0;

/* [Takeda 7]
    IsTop = strcmp( oua[i].pre->phn, "Top" ) == 0 ? 1 : 0;
*/
/***** edited by K.Abe '89 Oct. 17 ********************************************/
/*    IsTop = strcmp( oua[i].pre, "Top" ) == 0 ? 1 : 0;                        */
/***** edited by K.Abe '89 Oct. 17 ********************************************/

    /* Is the unit end at the end of the word ? */

/***** edited by K.Abe '89 Oct. 17 ********************************************/
/* [Takeda 7 ] */
    if(( i < oua_size - 1 && ( ! strcmp( oua[i+1].roman, "pau" )))
       ||
       ( ! strcmp( oua[i].fol -> phn, "End" )))
```

```
          IsEnd = 1;
     else
          IsEnd = 0;
/* [Takeda 7]
     IsEnd = strcmp( oua[i].fol->phn, "End" ) == 0 ? 1 : 0;
*/
/***** edited by K.Abe '89 Oct. 17 ***********************************************/
/*     IsEnd = strcmp( oua[i].fol, "End" ) == 0 ? 1 : 0;                          */
/***** edited by K.Abe '89 Oct. 17 ***********************************************/

     /* Is the left context is coinside ? */

/***** edited by K.Abe '89 Oct. 17 ***********************************************
     if(( i > 0 ) && ( strcmp( oua[i].pre->phn, Unit_end_phoneme(oua[i-1]) ) == 0 ))
**** edited by K.Abe '89 Oct. 17 ***********************************************
     if(( i > 0 ) && ( strcmp( oua[i].pre, Unit_end_phoneme(oua[i-1]) ) == 0 ))
***** edited by K.Abe '89 Oct. 17 ***********************************************/
     if(( i > 0 )
        &&
        ((! strcmp( oua[i].pre->phn, Unit_end_phoneme(oua[i-1]) ))
         ||
         quasi_same( oua[i].pre->phn, Unit_end_phoneme( oua[i-1] ))))
          IsHeadDup = 1;
     else
          IsHeadDup = 0;

     /* Is the right context is coinside ? */

/***** edited by K.Abe '89 Oct. 17 ***********************************************
     if(( i < oua_size - 1 )
        && ( strcmp( oua[i].fol->phn, Unit_start_phoneme(oua[i+1]) ) == 0 ))
***** edited by K.Abe '89 Oct. 17 ***********************************************
*     if(( i < oua_size - 1 )                                                      *
*        && ( strcmp( oua[i].fol, Unit_start_phoneme(oua[i+1]) ) == 0 ))
***** edited by K.Abe '89 Oct. 17 ***********************************************/
     if(( i < oua_size - 1 )
        &&
        (( strcmp( oua[i].fol->phn, Unit_start_phoneme(oua[i+1])) == 0 )
         ||
         quasi_same( oua[i].fol->phn, Unit_start_phoneme(oua[i+1]))))
          IsLastDup = 1;
     else
          IsLastDup = 0;

     if( IsTop || (! IsHeadDup ))
          TopShouldMoved = 0;
     else
          TopShouldMoved = 1;

     if( IsEnd || (! IsLastDup))
          EndShouldMoved = 0;
     else
          EndShouldMoved = 1;

     if(( stt > 0 ) && ( end >= 0 ))
     /* Only start edge should be treated */
     {
       if( ! TopShouldMoved )
       {
         margin[index].Top = margin[index].End = 0;
         new_ua = read_label( oua[i].word, oua[i].stt, oua[i].end );
       }
       else
       {
         margin[index].Top = 1; margin[index].End = 0;
```

```
        new_ua = read_label( oua[i].word, oua[i].stt - 1, oua[i].end );
      }
    }
    else if (( stt <= 0 ) && ( end < 0 ))
    /* Only end edge should be treated */
    {
      if( ! EndShouldMoved )
      {
        margin[index].Top = 0; margin[index].End = 0;
        new_ua = read_label( oua[i].word, oua[i].stt, oua[i].end );
      }
      else
      {
        margin[index].Top = 0; margin[index].End = 1;
        new_ua = read_label( oua[i].word, oua[i].stt, oua[i].end + 1 );
      }
    }
    else if(( stt > 0 ) && ( end < 0 ))
    /* Both edge should be treated */
    {
      if(( ! EndShouldMoved ) && ( ! TopShouldMoved ))
      {
        margin[index].Top = 0; margin[index].End = 0;
        new_ua = read_label( oua[i].word, oua[i].stt, oua[i].end );
      }
      else if( TopShouldMoved && ( ! EndShouldMoved ))
      {
        margin[index].Top = 1; margin[index].End = 0;
        new_ua = read_label( oua[i].word, oua[i].stt - 1, oua[i].end );
      }
      else if( EndShouldMoved && ( ! TopShouldMoved ))
      {
        margin[index].Top = 0; margin[index].End = 1;
        new_ua = read_label( oua[i].word, oua[i].stt, oua[i].end + 1 );
      }
      else
      {
        margin[index].Top = 1; margin[index].End = 1;
        new_ua = read_label( oua[i].word, oua[i].stt - 1 , oua[i].end + 1 );
      }
    }
    else
    {
      margin[index].Top = margin[index].End = 0;
      new_ua = read_label( oua[i].word, oua[i].stt, oua[i].end );
    }

    new_ua -> wd_roman = oua[i].wd_roman;
    new_ua -> roman = oua[i].roman;
    new_ua -> word = oua[i].word;
    ResetStartPoint( new_ua, &oua[i], stt, end );
    ResetEndPoint( new_ua, &oua[i], stt, end );
    ResetLength( new_ua, &oua[i], stt, end );

    new[index] = new_ua;
    index++;
  }

  return(index);
}

ResetStartPoint( new, old, stt, end )
UnitAttribute *new, *old;
int stt, end;
{
```

```
/* Reset start frame */
      new -> phn[0]
          -> stt
          =
      new -> phn[0]
          -> eve[0]
          -> stt
          = old -> phn[0] -> stt - stt;
}
ResetEndPoint( new, old, stt, end )
UnitAttribute *new, *old;
int stt, end;
{
/* Reset end frame */
      new -> phn[ new -> phn_size - 1 ]
          -> end
          =
      new -> phn[ new -> phn_size - 1 ]
          -> eve[ new -> phn[ new -> phn_size - 1 ] -> eve_size - 1 ]
          -> end
           = old -> phn[ old -> phn_size - 1] -> end - end;
}
ResetLength( new, old, stt, end )
UnitAttribute *new, *old;
int stt, end;
{
  new -> length = old -> length + ( stt - end );
}

set_phoneme_data( work, work_size,
                  new, ua_size,
                  dr, dr_size,
                  pn )
WorkTable work[];
UnitAttribute *new[];
DurationData dr[];
Phoneme pn[];
int work_size, ua_size, dr_size;
{
  /* [Takeda Aug 23, 1989]
    This progam set phonemic information and target duration onto the
     array pn[], by referring the work[] array */

  register int i, j, k;
  int pn_count = 0;
  int wunit, wpno, wsize, wphn;
  int target_duration, original_duration;
  int od1, od2;

  int FusionDupFlag = 1;
      /* [Takeda 5]
         if this flag is zero, two units are concatenated
         at the medial resion of inseparable phonemes. */

  float ratio;

  for( i = 0; i < work_size; i++ )
  {
    wunit = work[i].unit;
    wpno = work[i].phn;
    wsize = work[i].size;

    target_duration = get_target_duration( dr, &work[i], new );

    /* If the segment is pause */
```

```
    if( strcmp( dr[ work[i].durtabp[0] ].phn, "pau" ) == 0 )
    {
      set_pause_duration( &pn[pn_count++], target_duration );
    }
    /* The boundary to the following unit is located in the segment */
    else
         /* [Takeda 5],
             if the ending phoneme is inseparable, then the duplicate should
             be detected by comparing the last elemental phoneme of preceding
             unit and Either of phoneme involved in the top of
             following phoneme
         if(( i < work_size - 1 ) &&
            ( work[i].durtabp[ wsize - 1 ] == work[ i + 1 ].durtabp[0] ))
                                           ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                           This is not correct */
    {
      FusionDupFlag = 1;
      for( k = 0; k < work[i+1].size; k++ )
           FusionDupFlag *= ( work[i  ].durtabp[ wsize - 1 ] ==
                              work[i+1].durtabp[k]
                              ?
                             0:1);
      if(! FusionDupFlag)
      {
        target_duration = max( get_target_duration( dr, &work[i+1], new ),
                               target_duration );

        /* the duration of the forwarding portion */
        od1 = ( new[ work[i].unit ] -> phn[ work[i].phn ] -> end
                -
                new[ work[i].unit ] -> phn[ work[i].phn ] -> stt );

      /* the duration of the following portion */
        od2 = ( new[ work[i+1].unit ] -> phn[ work[i+1].phn ] -> end
                -
                new[ work[i+1].unit ] -> phn[ work[i+1].phn ] -> stt );

        original_duration = od1 + od2;

        ratio = (float)target_duration /
                (float)original_duration;

        set_duration_by_ratio( &pn[pn_count++], ratio, &work[i], new );
        set_duration_by_ratio( &pn[pn_count++], ratio, &work[i+1], new );
        i++;
      }
      else
      {
        set_duration_by_length( &pn[pn_count++],
                                target_duration,
                                &work[i],
                                new );
      }
    }
  }
  return(pn_count);
}

int UnitPhonemeDuration( p )
Phone *p;
{
  return( p->end - p->stt);
}

get_target_duration( dr, work, new )
```

```
DurationData dr[];
WorkTable *work;
UnitAttribute *new[];
{
  register int i;
  int duration = 0;
  unsigned int type_of_insep, classify_insep();

  if( work -> size == 1 ) return( dr[ work-> durtabp[0] ].len );

  type_of_insep = classify_insep( new[ work -> unit ] ->
                                  phn[ work -> phn ] );

  switch( type_of_insep )
  {
  case InsepDevocalization:
    for( i = 0; i  < work -> size; i++ )
    {
      if( isV( *dr[ work -> durtabp[i]].phn ) != 1 )
      {
        duration += dr[ work -> durtabp[i]].len;
      }
    }
    break;

  default:
    for( i = 0; i < work -> size; i++ )
    {
      duration += dr[ work-> durtabp[i]].len;
    }
    break;
  }
  return(duration);
}


set_duration_by_length( pn, length, work, new )
Phoneme *pn;
int length;
WorkTable *work;
UnitAttribute *new[];
{
  int wuno, wpno;
  int frame_len;

  wuno = work -> unit;
  wpno = work -> phn;

  pn -> inf = work -> inf;
  pn -> phn = (Phone*)malloc(sizeof(Phone));
  if( work->inf == PhDuplicatedBefore )
  {
    pn->phn->phn = (char*)malloc(10);
    strcpy( pn->phn->phn, new[wuno]->pre->phn );
  }
  else
  pn->phn = new[wuno]->phn[wpno];

  pn->unit_no = new[wuno]->word;
  pn->length = length;
  frame_len = UnitPhonemeDuration( new[wuno] -> phn[wpno] );

  /* [Yamazaki 1]

  if ((float)pn->length/frame_len > MaxLength)
```

```
      pn->length = round(MaxLength*frame_len);
    if ((float)pn->length/frame_len < MinLength)
      pn->length = max( round(MinLength*frame_len), 1 );
      */

    return(0);
}


set_pause_duration( pn, length )
Phoneme *pn;
int length;
{
    pn -> inf = PhPause;  /* [Takeda 4] */
    pn -> phn = (Phone*)malloc(sizeof(Phone));
    pn->phn->phn = (char*)malloc(4);
    strcpy( pn->phn->phn, "pau" );
    pn->unit_no = 0;
    pn->length = length;
    return(0);
}

set_duration_by_ratio( pn, ratio, work, new )
Phoneme *pn;
float ratio;
WorkTable *work;
UnitAttribute *new[];
{
    int wuno, wpno;
    int frame_len;

    wuno = work -> unit;
    wpno = work -> phn;
    pn -> phn = (Phone*)malloc(sizeof(Phone));
    pn -> inf = PhDuplicate | work -> inf; /* [Takeda 4] */
    if( work->inf == PhDuplicatedBefore )
    {
        pn->phn->phn = (char*)malloc(10);
        strcpy( pn->phn->phn, new[wuno]->pre->phn );
      }
    else
         pn->phn = new[wuno]->phn[wpno];

    pn->unit_no = new[wuno]->word;
    frame_len = UnitPhonemeDuration( new[wuno] -> phn[wpno] );
    pn->length = max( round((float)frame_len * ratio), 1 );

    /* [Yamazaki 1]
       [Takeda 7],
       Hey Mr. Yamazaki, moast of your early efforts are
       needless by now, .
    if ((float)pn->length/frame_len > MaxLength)
       pn->length = round(MaxLength*frame_len);
    if ((float)pn->length/frame_len < MinLength)
       pn->length = max( round(MinLength*frame_len), 1 );
       */
    return(0);
}


break_down( s, lbl )
char s[], *lbl[];
{
    register int i;
    int num = 0;
```

```
  char buf[32];


  strcpy( buf, "" );
  for( i = 0 ; i < strlen(s); i++ )
  {
    if( s[i] == ',' )
    {
      lbl[num] = (char*)malloc(strlen(buf)+1);
      strcpy( lbl[num], buf );
      strcpy( buf, "" );
      num++;
    }
    else
    {
      sprintf( buf, "%s%c", buf, s[i] );
    }
  }
  lbl[num] = (char*)malloc(strlen(buf)+1);
  strcpy( lbl[num], buf );
  num++;
  return(num);
}


create_phn_inf_file( phns, header, unit, work )
int phns;
char *header;
Phoneme unit[];
WorkTable work[];
{
  FILE *fp, *fopen();
  char file[256];
  register int tab = 0;
  register int i, j;
  int off_set = 0;

  sprintf( file, "%s.PN", header );
  if(( fp = fopen( file, "w" )) == NULL )
  {
    fprintf(stderr,"[Error in create_phn_inf_file] File %s can't open.\n",
            file );
    return(-1);
  }

  for( i = 0; i < phns; i++ )
  {
    fprintf(fp, "%4d %4d %2d %04d %4d %4d %4d %6s %5d",
                i,
                unit[i].length,
                unit[i].inf,
                unit[i].unit_no,
                off_set,
                (off_set + unit[i].length),
                unit[i].phn->end - unit[i].phn->stt,
                unit[i].phn->phn,
                work[tab].size
           );
    for( j = 0; j < work[tab].size; j++ )
         fprintf( fp, " %4d" ,work[tab].durtabp[j] );
    fprintf( fp, "\n" );
    tab++;
    off_set += unit[i].length;
  }
  fclose(fp);
```

```
  return(i);
}

create_frm_inf_file( phns, header, unit )
char *header;
int phns;
Phoneme unit[];
{
  FILE *fp, *fopen();
  float ratio;
  int counter = 0;
  register int i, j, k, l;
  char file[256], event[128], alpone[128];

  sprintf( file, "%s.FR", header );
  if(( fp = fopen( file, "w" )) == NULL )
  {
    fprintf(stderr,"[Error in frm_phn_inf_file] File %s can't open.\n",
            file );
    return(-1);
  }
/* [Takda 5]  The seventh field of the .PN file means
              the size of phoneme corresponding the
              unit element.
*/

  for( i = 0; i < phns; i++ )
  {
    if( strcmp( unit[i].phn->phn, "pau" ) == 0 )
    {
      for( j = 0; j < unit[i].length; j++ )
      {
        fprintf( fp, "%4d %4d %5d %5s %5s %5s\n",
                 counter++,
                 0,
                 0,
                 unit[i].phn->phn,
                 "(nil)",
                 "(nil)"
                );
      }
    }
    else
    {
      ratio = (float)( unit[i].phn->end - unit[i].phn->stt )
            /(float)unit[i].length;
      if( ratio == 0 )  ratio = 1.0 / (float)unit[i].length;

      for( j = 0; j < unit[i].length; j++ )
      {
        l = round((float)j*ratio) + unit[i].phn->stt;

        strcpy( event, "" );
        strcpy( alpone, "" );
        for( k = 0; k < unit[i].phn->eve_size; k++ )
        {
          if( unit[i].phn->eve[k]->stt <= l && unit[i].phn->eve[k]->end >= l )
              strcpy( event,  unit[i].phn->eve[k]->eve );
        }
        for( k = 0; k < unit[i].phn->alp_size; k++ )
        {
          if( unit[i].phn->alp[k]->stt <= l && unit[i].phn->alp[k]->end >= l )
              strcpy( alpone, unit[i].phn->alp[k]->alp );
        }
        fprintf( fp, "%4d %4d %5d %5s %5s %5s\n",
```

```
                    counter++,
                    unit[i].unit_no,
                    l,
                    unit[i].phn->phn,
                    ( strcmp( event, "" ) == 0 ? "(nil)" : event ),
                    ( strcmp( alpone, "" ) == 0 ? "(nil)" : alpone )
                );
        }
      }
    }
    fclose(fp);
    return(i);
}


round(f) float f; { return((int)(f+0.5)); }

printerror( uno, pno, u, fsize, dur )
int uno, pno, fsize;
UnitAttribute u[];
DurationData *dur;
{
    register int j;
    fprintf(E," insep_proc: mismatch !!.\n" );
    fprintf(E,"              Unit# = %2d, Phoneme# = %5d \n", uno, pno );
    fprintf(E," \tunit attribute label = %s\n", u[uno].phn[pno]->phn );
    fprintf(E," \tduration labels are;\n" );
    for( j = 0; j < fsize; j++ )
        fprintf(E,"\t\t%s\n", (dur+j)->phn );
}

UnitAttribute *read_label( word, stt, end )
int word, stt, end;
{
    struct { int stt, end; char *sym; } work[3][256];
    UnitAttribute *unit;
    Phone ph[256];
    Event ev[512];
    Alophone al[128];
    FILE *fp, *open_label_file();

    char *roman;            /* roman description of the word */
    char l[256];            /* read buffer */
    int size[3];            /* size of each layer of label */
    /* [Takeda 1]
        l_stt and l_end are needless !.
    int l_stt, l_end;        label no.
                             associating first and last phoneme of unit */
    int layer = 0;          /* layer no of work array
                                       0; phonemic label layer
                                       1; event label layer
                                       2; alophonic variation layer
                            */
    int label = 0;          /* symbol no */
    unsigned int fusion_flag = 0;

    register int i, j, k;

    unit = (UnitAttribute*)malloc(sizeof(UnitAttribute));
    /* set word no of the unit */

    if( word < 1 || word > 5240 )
    {
      fprintf(E,"read_label: Bad word no (%d).\n", word );
      return((UnitAttribute*)NULL);
```

```
  }
  if(( fp = open_label_file(word)) == NULL )
  {
    fprintf(E,"read_label: label file (%04d) can't open.\n", word );
    return((UnitAttribute*)NULL);
  }

  /* set word and location attribute of unit */
  unit->stt = stt;
  unit->end = end;

  /* read label */
  while( fgets( l, 256, fp ) != 0 )
  {
    char lb[64];          /* read buffer for label symbol */
    float f_stt, f_end; /* read buffer for label start and end */

    if( l[0] == '#' )
    /* read next layer */
    {
      size[layer] = label;
      if( ++layer > 2 ) break;
      label = 0;
    }
    else
    {
      sscanf( l, "%f %s %f", &f_stt, lb, &f_end );
      work[layer][label].stt = f_stt/Ratio;
      work[layer][label].end = f_end/Ratio;
      work[layer][label].sym = (char*)malloc( strlen(lb)+1);
      strcpy( work[layer][label].sym, lb );
      label++;
    }
  }

  /* set Phoneme data */
  for( i = 0; i < size[0]; i++ )
  {
    ph[i].stt = work[0][i].stt;
    ph[i].end = work[0][i].end;
    ph[i].phn = (char*)malloc(strlen(work[0][i].sym)+1);
    strcpy( ph[i].phn, work[0][i].sym );
  }
  /* set Event data */
  for( i = 0; i < size[1]; i++ )
  {
    ev[i].stt = work[1][i].stt;
    ev[i].end = work[1][i].end;
    ev[i].eve = (char*)malloc(strlen(work[1][i].sym)+1);
    strcpy( ev[i].eve, work[1][i].sym );
  }
  /* set Alophone data */
  for( i = 0; i < size[2]; i++ )
  {
    al[i].stt = work[2][i].stt;
    al[i].end = work[2][i].end;
    al[i].alp = (char*)malloc(strlen(work[2][i].sym)+1);
    strcpy( al[i].alp, work[2][i].sym );
  }

  /* [Takeda 1] You don't have to match string and label
                nay more ! */
  /* Match strings and label
  fusion_flag = match( work[0], stt, end, &l_stt, &l_end );
  */
```

```
  /* set pre/fol char */
  /* the index variable l_stt and l_end are changed to be
     stt and end, respectively. */

  stt--; end--;          /* stt and end should be counted from 0 */

/***** edited by K.Abe '89 Oct. 17 ***********************************************/
  if( stt == 0 )         /* if the unit start from initial of word */
  {
    unit->pre = (NeighborPhone*)malloc(sizeof(NeighborPhone));
    unit->pre->phn = (char*)malloc(4);
    strcpy( unit->pre->phn, "Top" );
  }
  else
  {
    unit->pre = (NeighborPhone*)malloc(sizeof(NeighborPhone));
    unit->pre->phn = (char*)malloc(strlen(ph[stt-1].phn)+1);
    strcpy( unit->pre->phn, ph[stt-1].phn );
  }
  if( end == size[0] - 1 )       /* if the unit end at the end of the word */
  {
    unit->fol = (NeighborPhone*)malloc(sizeof(NeighborPhone));
    unit->fol->phn = (char*)malloc(4);
    strcpy( unit->fol->phn, "End" );
  }
  else
  {
    unit->fol = (NeighborPhone*)malloc(sizeof(NeighborPhone));
    unit->fol->phn = (char*)malloc(strlen(ph[end+1].phn)+1);
    strcpy( unit->fol->phn, ph[end+1].phn );
  }
/***** edited by K.Abe '89 Oct. 17 ***********************************************/
/***** edited by K.Abe '89 Oct. 17 ***********************************************/
/*  if( stt == 0 )                                                               */
/*  {                                                                            */
/*    unit->pre = (char*)malloc(4);                                              */
/*    strcpy( unit->pre, "Top" );                                                */
/*  }                                                                            */
/*  else                                                                         */
/*  {                                                                            */
/*    unit->pre = (char*)malloc(strlen(ph[stt-1].phn)+1);                        */
/*    strcpy( unit->pre, ph[stt-1].phn );                                        */
/*  }                                                                            */
/*  if( end == size[0] - 1 )                                                     */
/*  {                                                                            */
/*    unit->fol = (char*)malloc(4);                                              */
/*    strcpy( unit->fol, "End" );                                                */
/*  }                                                                            */
/*  else                                                                         */
/*  {                                                                            */
/*    unit->fol = (char*)malloc(strlen(ph[end+1].phn)+1);                        */
/*    strcpy( unit->fol, ph[end+1].phn );                                        */
/*  }                                                                            */
/***** edited by K.Abe '89 Oct. 17 ***********************************************/

  /* set ph field of unit data */
  for( i = stt; i <= end; i++ )
  {
    unit->phn[i-stt] = (Phone*)malloc(sizeof(Phone));
    unit->phn[i-stt]->stt = ph[i].stt;
    unit->phn[i-stt]->end = ph[i].end;
    unit->phn[i-stt]->phn = (char*)malloc(strlen(ph[i].phn)+1);
    strcpy( unit->phn[i-stt]->phn, ph[i].phn );
  }
```

```
  unit->phn_size = end - stt + 1;

  for( i = 0; i < unit->phn_size; i++ )
  {
    int eve_size = 0;    /* event included in the i-th phoneme */
    int alp_size = 0;    /* alophonic symbols included in the i-th phoneme */

    /* search event that is included in the phoneme */
    for( j = 0; j < size[1]; j++ )
    {
      if( unit->phn[i]->stt <= ev[j].stt &&
          unit->phn[i]->end >= ev[j].end )
      {
        unit->phn[i]->eve[eve_size] = (Event*)malloc(sizeof(Event));
        unit->phn[i]->eve[eve_size]->stt = ev[j].stt;
        unit->phn[i]->eve[eve_size]->end = ev[j].end;
        unit->phn[i]->eve[eve_size]->eve = (char*)malloc(strlen(ev[j].eve)+1);
        strcpy( unit->phn[i]->eve[eve_size]->eve, ev[j].eve );
        eve_size++;
      }
    }
    /* set including event size */
    unit->phn[i]->eve_size = eve_size;

    /* search alophonic symbol that is included in the phoneme */
    for( j = 0; j < size[2]; j++ )
    {
      if( unit->phn[i]->stt <= al[j].stt &&
          unit->phn[i]->end >= al[j].end )
      {
        unit->phn[i]->alp[alp_size] = (Alophone*)malloc(sizeof(Alophone));
        unit->phn[i]->alp[alp_size]->stt = al[j].stt;
        unit->phn[i]->alp[alp_size]->end = al[j].end;
        unit->phn[i]->alp[alp_size]->alp = (char*)malloc(strlen(al[j].alp)+1);
        strcpy( unit->phn[i]->alp[alp_size]->alp, al[j].alp );
        alp_size++;
      }
    }
    /* set included alophonic symbol size */
    unit->phn[i]->alp_size = alp_size;
  }

  /* set unit full length */
  return(unit);
}

FILE *open_label_file(no)
int no;
{
  FILE *fp;
  char f[256];

  sprintf( f, "/MHT/LBL/D%d/MHT_1_%04d.LB", no/1000, no );
  if(( fp = fopen( f, "r" )) == NULL )
  {
    fprintf(E,"open_label_file: can't open %s.\n", f );
    return((FILE*)NULL);
  }
  return(fp);
}

unsigned int classify_insep( label )
Phone *label;
{
  /* This program classifies all inseparable phonemic sequence
```

```
     ( assimilation ) into followin categories.

    - Devocalization
    - NasalCluster
    - Others

  */

  register int i;

  /* Detect devocalization */

  for( i = 0; i < label->alp_size; i++ )
  {
    if( strcmp( label->alp[i]->alp, "dv" ) == 0 )
         return( InsepDevocalization );
  }

  /* Detect NasalCluster */

  for( i = 0; i < label->eve_size; i++ )
  {
    if( index( label->eve[i]->eve, 'N' ) == 1 )
         return( InsepNasalCluster );
  }

  /* Others */

  return( InsepOthers );
}

index( s, c )
char *s, c;
{
  char d;
  while( d = *s++ )
  {
    if( c == d )         return(1);
  }
  return(0);
}
char *Lastphoneme( s )
char *s;
{
  char *ptr;
  register int i;
  for( i = strlen(s) - 1; i >= 0; i-- )
  {
    ptr = s+i;
    if(*(s+i) == ',' )
    {
      ptr++; break;
    }
  }
  return(ptr);
}
int Size(s)
char *s;
{
  char c;
  int count = 1;
  while( c = *s++ ) if ( c == ',' ) count++;
  return(count);
}
debug_print( new, work, size, dr )
```

```
UnitAttribute *new[];
WorkTable work[];
int size;
DurationData dr[];
{
  register int iz, izz;
  for( iz = 0; iz < size; iz++ )
  {
    if( work[iz].unit < 0 )
    {
      printf( "U: -1 P: -1 [pau] -> pause\n" );
      continue;
    }
    printf( "U: %d P: %d [%s] -> ",
           work[iz].unit,
           work[iz].phn,
           new[work[iz].unit]->phn[ work[iz].phn ]->phn );
    for( izz = 0; izz < work[iz].size; izz++ )
         printf("(%d %s)", work[iz].durtabp[izz],
                           dr[work[iz].durtabp[izz]].phn );
    printf("\n");
  }
}
static int read_size = 0;
static struct
{
  char *u_en;
  char *u_pre;
} same_ph_table[256];

quasi_same( s1, s2 )
char *s1, *s2;
{
  register int i;

  if( read_size == 0 )
       if(( read_size = read_table( "/SYN/Parms/same-ph-tab" )) < 0 )
       {
         fprintf( stderr, "can't read table SamePhonemeTable\n");
         exit(-1);
       }


  for( i = 0; i < read_size; i++ )
  {
    if(( strcmp( s1, same_ph_table[i].u_en ) == 0 )
       &&
       ( strcmp( s2, same_ph_table[i].u_pre ) == 0 ))
         return(1);
  }
  return(0);
}

read_table(file)
char *file;
{
  FILE *fp, *fopen();
  register int i = 0;
  char line[256];
  char en[20], pre[20];

  if(( fp = fopen( file, "r" )) == (FILE*)NULL )
  {
    fprintf( stderr, "file: %s, can't open.\n", file );
    return(-1);
```

```
  }

  while( fgets( line, 256, fp ) != 0 )
  {
    if( line[0] == '#' )
         continue;
    sscanf( line, "%s %s", en, pre );

    same_ph_table[i].u_en = (char*)malloc(strlen(en)+1);
    same_ph_table[i].u_pre = (char*)malloc(strlen(pre)+1);

    strcpy( same_ph_table[i].u_en, en );
    strcpy( same_ph_table[i].u_pre, pre );
    i++;
  }
  return(i);
}

match_error( uac, pc, drc, new, dr, sw, inf )
int uac, pc, drc, sw, inf;
UnitAttribute *new[];
DurationData dr[];
{
  int i;
  fprintf(E,"%s: phoneme sequence of dur. and unit mismatch.\n", prog );
  fprintf(E,"phoneme = %s  vs %s = durtab\n",
          new[uac]->phn[pc]->phn, dr[drc].phn );
  fprintf(E, " this mis-match occured \n " );
  switch(sw)
  {
  case 1:
    fprintf(E, "between unit %d and %d\n", uac, (uac-1) );
    fprintf(E, "tryed duration phonemes were\n" );
    for( i = 0; i < inf; i++ )
         fprintf( E, "  [ %s ]", dr[drc - inf + i].phn );
    fprintf(E, "\n");
    break;
  case 2:
    fprintf(E, "Unit: %3d.     DrPhn: %3d.\n", uac, drc );
    fprintf(E, " while tryed to match %dth phn of insep phns\n", inf );
    break;
  case 3:
    fprintf(E, "this occered at Unit: %3d.     DrPhn: %3d.\n", uac, drc );
    break;
  }
  return(-1);
}

match_dr_and_ua( margin,
                 new,
                 work,
                 ua_size,
                 dr_size, dur )
int ua_size, dr_size;
Margin margin[];
UnitAttribute *new[];
DurationData dur[];
WorkTable work[];
/*
  [Takeda Aug 23, 1989]
  This program matches .ua file contents and .DR file contents and generates
  work array in which correspondance of two files are written.
  [Takeda 6]
  I rewrote almost everything, listening to Chisato, CHISAMA Moritaka !!
*/
```

```
{
  int ua;                       /* counter for unit attribute array */
  int p;                        /* counter for phoneme in a unit */
  int size;             /* inseparable phonemic label size if it is */
  int wk = 0;           /* index for work table */


  char *eles[42];
  char *follow();               /* i,y --> y */
  char *current_phn;
  int u, d, w;

  int dr = 0;
  register int i;
                                /* suffix of array pn and dr */

  for ( ua = 0; ua < ua_size; ua++ )
  {
    /* If the beginning edge of the unit was moved to preceding phoneme */

    if( ua )
    {
      if(( ! margin[ua].Top ) &&  ( ! margin[ua-1].End ))
           goto StartMatchHere;
    }
    else
    {
      if( ! margin[ua].Top )
           goto StartMatchHere;
    }

    current_phn = new[ua]->phn[0]->phn;

/*
  If adjasent two units are concatenated in the middle of
  vowels, the vowel would be appear in the unit information
  array, onle one corresponding phoneme is on the duration
  information array, though.

  Thus, in shuch case, the duraiton array counter, "dr" should be
  count down by one, ( or in some case appears below, by more than
  one.)
 */

    if(( ! strcmp( current_phn, dur[dr-1].phn )))
    { dr--; goto StartMatchHere; }
/*
  As for long vowels some special procedures are required.
  such case that      ane + eta <> a n E t a
 */

    if(( ! strcmp( current_phn, "o" )) && (! strcmp( dur[dr-1].phn, "o,u" )))
    { dr--; goto StartMatchHere; }

    if(( ! strcmp( current_phn, "e" )) && (! strcmp( dur[dr-1].phn, "e,i" )))
    { dr--; goto StartMatchHere; }

    if(( ! strcmp( current_phn, "o,u" )) && (! strcmp( dur[dr-1].phn, "o" )))
    { dr--; goto StartMatchHere; }

    if(( ! strcmp( current_phn, "e,i" )) && (! strcmp( dur[dr-1].phn, "e" )))
    { dr--; goto StartMatchHere; }

/*
```

```
  Preceded by "e", "i" can be regarded as "e".
  The same relation can be true for "o" and "u".
 */
    if(( dr > 2 )
       &&
       (( ! strcmp( current_phn, "e" ))
        &&
        ( ! strcmp( dur[dr-1].phn, "i" ))
        &&
        ( ! strcmp( dur[dr-2].phn, "e" )))

        ||

       (( ! strcmp( current_phn, "o" ))
        &&
        ( ! strcmp( dur[dr-1].phn, "u" ))
        &&
        ( ! strcmp( dur[dr-2].phn, "o" ))))
    {
      work[wk].unit = ua;
      work[wk].phn = 0;  /* always zero */
      work[wk].size = 2; /* this unit-phoneme corresponds two dr-phonemes */
      work[wk].durtabp[0] = dr - 2;
      work[wk].durtabp[1] = dr - 1;
      work[wk].inf = PhDuplicate;
      wk++;
      p = 1;
      goto InsideLoop; /* Jump into the loop */
    }

/*
  special proc for inseparable portions
  for such case that i + i,y <> iy
 */

    size = break_down( current_phn, eles );

    for( i = 0; i < size; i++ )
    {
      if( ! strcmp( eles[0], dur[dr - size + i ].phn ))
      {
        dr -= (size-i);
        goto StartMatchHere;
      }
    }
    match_error( ua, 0, dr, new, dur, 1, size );
    return(-1);


  /* start matching. */

  StartMatchHere:
    for( p = 0; p < new[ua]->phn_size; p++ )
    {

    InsideLoop:
      /* The .DR record is pause */
      if(( p == 0) && ( ! strcmp( dur[dr].phn, "pau" )))
      {
        work[wk].unit = -1;
        work[wk].phn = -1;;
        work[wk].size = 1;
        work[wk].durtabp[0] = dr;
        work[wk].inf = PhPause;     /* [Takeda 4] */
        dr++;
```

```
        wk++;
      }

      current_phn = new[ua]->phn[p]->phn;
/*
  The unit-phoneme and duration-phoneme are coinside simplly.
  */
      if(( ! strcmp( current_phn, dur[dr].phn ))
         ||
         (( ! strcmp( dur[dr].phn, "w" ))
           &&
          ( ! strcmp( current_phn, "h" ))))
      {
        work[wk].unit = ua;
        work[wk].phn = p;
        work[wk].size = 1;
        work[wk].durtabp[0] = dr;
        work[wk].inf = PhUsualMatch; /* [Takeda 4] */
        dr++;
        wk++;
        continue;
      }


/*
  match "o,u" and "o", "u"( "e,i" and "e", "i" ) .
 */
      if((( ! strcmp( dur[dr].phn, "o,u" ))
          &&
          ( ! strcmp( current_phn, "o" ))
          &&
          ( ! strcmp( follow( new, ua, p ), "u" )))

         ||

         (( ! strcmp( dur[dr].phn, "e,i" ))
          &&
          ( ! strcmp( current_phn, "e" ))
          &&
          ( ! strcmp( follow( new, ua, p ), "i" ))))
      {
        work[wk].unit = work[wk+1].unit = ua;
        work[wk].phn = work[wk+1].phn = p;
        work[wk].size = work[wk+1].size = 1;
        work[wk].durtabp[0] = work[wk+1].durtabp[0] = dr;
        work[wk].inf = work[wk+1].inf = PhUsualMatch;
        wk += 2;
        p++;
        dr++;
        continue;
      }

/*
  to match an inseparable unit-phoneme-cluster and duration phonemes
*/

      if(( size = break_down( current_phn, eles )) <= 1 )
           goto NextMatch;

      /* the current value of dr should be memorized. */
      d = dr;
      work[wk].size = size;

      for( i = 0; i < size; i++ )
      {
```

```
        if(( ! strcmp( eles[i], dur[dr].phn ))
           ||
           (( ! strcmp( dur[dr].phn, "w" )) && ( ! strcmp( eles[i], "h" ))))
        {
          work[wk].durtabp[i] = dr++;
        }
        else
        {
          dr = d;                /* reset the counted up dr calue */
          work[wk].size = 0;
          goto NextMatch;
        }
      }
      work[wk].unit = ua;
      work[wk].phn = p;
      work[wk].inf = PhFusion; /* [Takeda 4] */
      wk++;
      continue;

    NextMatch:

/*
  Special Condition for unit start and end phonemes
 */

      if( p == 0 )
      {
        u = ua; d = dr; w = wk;
        if( beginingedge( new, &ua, dur, &dr, work, &wk ) > 0 )
             continue;

        ua = u; dr = d; wk = w;
        if((( ! strcmp( current_phn, "o,u" ))
            &&
            ( ! strcmp( dur[dr].phn, "o" )))
           ||
           (( ! strcmp( current_phn, "e,i" ))
            &&
            ( ! strcmp( dur[dr].phn, "e" )))
           ||
           (( ! strcmp( current_phn, "e" ))
            &&
            ( ! strcmp( dur[dr].phn, "e,i" )))
           ||
           (( ! strcmp( current_phn, "o" ))
            &&
            ( ! strcmp( dur[dr].phn, "o,u" ))))

        {
          work[wk].unit = ua;
          work[wk].phn = p;
          work[wk].size = 1;
          work[wk].durtabp[0] = dr;
          work[wk].inf = PhUsualMatch; /* [Takeda 4] */
          dr++;
          wk++;
          continue;
        }

      }

      if( p == new[ua] -> phn_size - 1 )
      {
        u = ua; d = dr; w = wk;
        if( endingedge( new, &ua, p, dur, &dr, work, &wk ) > 0 )
```

```
            continue;

        ua = u; dr = d; wk = w;
        if((( ! strcmp( current_phn, "o,u" ))
            &&
            ( ! strcmp( dur[dr].phn, "o" )))
           ||
           (( ! strcmp( current_phn, "e,i" ))
            &&
            ( ! strcmp( dur[dr].phn, "e" )))
           ||
           (( ! strcmp( current_phn, "e" ))
            &&
            ( ! strcmp( dur[dr].phn, "e,i" )))
           ||
           (( ! strcmp( current_phn, "o" ))
            &&
            ( ! strcmp( dur[dr].phn, "o,u" ))))
        {
          work[wk].unit = ua;
          work[wk].phn = p;
          work[wk].size = 1;
          work[wk].durtabp[0] = dr;
          work[wk].inf = PhUsualMatch; /* [Takeda 4] */
          dr++;
          wk++;
          continue;
        }
      }

/*
 * This must be tentative !!!, KIn Nov 26.
 *
 */
      if( ! strcmp( current_phn, dur[dr+1] ))
      {
        wk--;
        work[wk].size++;
        work[wk].durtabp[ work[wk].size - 1 ] = dr;
        work[wk].inf = PhDuplicate;
        dr ++;
        wk++;
        p--;
        continue;
      }

      match_error( ua, 0, dr, new, dur, 1, size );
      return(-1);

    } /* end of p-loop */

  }   /* End of ua loop */
  /* Maybe the last phoneme is "pause" */
  if( ! strcmp( dur[dr].phn, "pau" ))
  {
    work[wk].unit = -1;
    work[wk].phn = -1;;
    work[wk].size = 1;
    work[wk].durtabp[0] = dr;
    work[wk].inf = PhPause;  /* [Takeda 4] */
    dr++;
    wk++;
  }
  return(wk);
}
```

```
beginingedge( new, ua, dur, dr, work, wk )
UnitAttribute *new[];
DurationData dur[];
WorkTable work[];
int *ua, *dr, *wk;
{
  int size;
  char *eles[10];
  register int i, j;

  if(( size = break_down( new[*ua] -> phn[0] -> phn, eles )) <= 1 )
       goto Error;

  for( i = 0; i < size; i++ )
  {
    if( ! strcmp( eles[i], dur[*dr] ) )
         goto NextLoop;
  }
  goto Error;

 NextLoop:

  for( j = 0 ; i < size; i++, (*dr)++, j++ )
  {
    if( strcmp( eles[i], dur[*dr].phn ))
         goto Error;

    work[*wk].unit = *ua;
    work[*wk].phn = 0;
    work[*wk].size++;
    work[*wk].durtabp[j] = *dr;
  }
  (*wk)++;
  return(1);

 Error:
    return(-1);
}


endingedge( new, ua, p, dur, dr, work, wk )
UnitAttribute *new[];
DurationData dur[];
WorkTable work[];
int *ua, *dr, *wk, p;
{
  int size;
  register int i;
  char *eles[10];


  if(( size = break_down( new[*ua] -> phn[p] -> phn, eles )) <= 1 )
       goto Error;

  i = 0;

  while( ! strcmp( eles[i++], dur[(*dr)++] ))
  {
    work[*wk].unit = *ua;
    work[*wk].phn = p;
    work[*wk].size++;
    work[*wk].durtabp[i-1] = *dr - 1;
```

```
    work[*wk].inf = PhFusion;
  }

  (*dr)--;
  (*wk)++;
  return(1);

 Error:
    return(-1);
}


char *follow( new, ua, p )
UnitAttribute *new[];
int ua, p;
{
  /* I believe in the given arguments are collect value. */

  return(
         ( p >= new[ua]->phn_size - 1 ) ?
         new[ua+1] -> phn[0] -> phn :
         new[ua] -> phn[p+1] -> phn );
}
```

```
/*
  mkcep.c

  make cepstrum file from unit file.
  This program read xxx.unit file as input and create xxx.CP file.
  Main task of this routine is control segmental duration.

  Originally coded by K. Takeda, ATR    [Jul 16, 1988]
  [Takeda 1]                             Jul. 28, 1988
  Modified to use new data structures
  [Yamazaki 1]                           Aug. 9, 1988
  Add following procedures
     1. Individual controlling of each acoustic event  in p, t, k
     2. Adding closure portion if not.
  [Kaiki 1]                             Jan. 16, 1990
  Modified voiceless stop bug fix.
*/
#include <stdio.h>
#include <fcntl.h>
#include <sys/file.h>
#include "/SYN/include/Synthesis.h"

#define E               stderr
#define Order           32
#define Initial_Memory  12800
#define NO_PAUSE        5

/***** edited by K.Abe '89 Oct.03 ************************************************/
/*      'Max_unit' is defined in 'Synthesis.h'                                    */
/***** edited by K.Abe '89 Oct.03 ************************************************/
/***** edited by K.Abe '89 Sep. 22 ***********************************************/
/* #define Max_unit     512                                                       */
/***** edited by K.Abe '89 Sep. 22 ***********************************************/

char *prog;

main(argc,argv)
int argc;
char *argv[];
{

/***** edited by K.Abe '89 Sep. 22 ***********************************************/
  Phoneme unit[Max_unit];
/*********************************************************************************/
/*  Phoneme unit[128];                                                           */
/***** edited by K.Abe '89 Sep. 22 ***********************************************/

  int fd_cep;
  int phns;
  char unit_file[256], cep_file[256], *file_header;

  prog = argv[0];

  if(argc == 1)
  {
    fprintf(E,"\t[%s] make cepstrum file from xxx.unit file.\n", prog );
    fprintf(E,"\t[Usage] %s: file-name-header\n", prog );
    exit(0);
  }

  file_header = argv[1];

  if(( phns = ReadPhoneme( file_header, unit )) <= 0 )
  {
    fprintf(E,"%s: unit_file (%s) can't read.\n", prog, unit_file );
```

```
    exit(1);
  }

  sprintf( cep_file, "%s.CP", file_header );

/***** edited by K.Abe '89 Dec. 01 ********************************************/
/*  if(( fd_cep = open( cep_file, O_WRONLY|O_CREAT, 420 )) < 0 )                */
/***** edited by K.Abe '89 Dec. 01 ********************************************/
  if(( fd_cep = open( cep_file, O_CREAT|O_WRONLY|O_TRUNC, 420 )) < 0 )
  {
    fprintf(E,"%s: cepstrum file (%s) can't create.\n", prog, cep_file );
    exit(1);
  }

  if( create_cep_file( fd_cep, phns, unit ) < 0 )
  {
    fprintf(E,"%s: error in create cep file.\n", prog );
    exit(2);
  }
  exit(0);
}

create_cep_file( fd, phns, unit )
int fd;
int phns;
Phoneme unit[];
{

  float *cep;           /* pointer for original cepstrum data */
  float *new_cep;       /* pointer for duration reset cepstrum data */
  int write_size;       /* size of duration reset cepstrum data */
  register int i;

  cep = (float*)malloc(12800);
  new_cep = (float*)malloc(12800);

  for( i = 0; i < phns; i++ )
  {
    write_size = unit[i].length * Order * sizeof(float);

    /* as for pause, all zero data are used */
    if( strcmp( unit[i].phn->phn, "pau" ) == 0 )
    {
      if( generate_pause( unit[i].length, new_cep ) < 0 )
      {
        fprintf(E,"generate pause error.\n");
        return(-1);
      }
    }
    else
    {
      if( read_cep( unit[i].unit_no, unit[i].phn->stt, unit[i].phn->end,
                    cep ) < 0 )
      {
        fprintf(E,"read cepfile error.\n");
        return(-1);
      }
      if( dur_reset( cep, new_cep, unit[i] ) <= 0 )
      {
        fprintf(E,"duration reset error.\n");
        return(-2);
      }
    }
    if( write( fd, new_cep, write_size ) != write_size )
    {
```

```
      fprintf(E,"cepfile write error in %dth unit.\n", i );
      return(-1);
    }
  }
}


read_cep( word, stt, end, cep )
int word, stt, end;
float *cep;
{
  int fd;
  int read_size;
  int off_set;
  char cep_file[256];


  /* set read size */
  read_size = end - stt + 1;
  read_size *= Order * sizeof(float);

  /* set read off set */
  off_set = stt * Order * sizeof(float);

  sprintf( cep_file, "/MHT/CEP/D%d/MHT_%04d.CEP", word/1000, word );
  if(( fd = open( cep_file, O_RDONLY )) < 0 )
  {
    fprintf(E,"original cep file (%s) can't open.\n", cep_file );
    return(-1);
  }

  /* rewind cep file */
  if( lseek( fd, off_set, L_SET ) != off_set )
  {
    fprintf(E,"original cep file (%s) seek error.\n", cep_file );
    return(-1);
  }
  /* read original cepstrum data */
  if( read( fd, cep, read_size ) != read_size )
  {
    fprintf(E,"original cep file (%s) unexpected eof.\n", cep_file );
    return(-1);
  }
  close(fd);
  return(1);
}


generate_pause( length, cep )
int length;
float *cep;
{
  register int i;

  for( i = 0; i < length*Order; i++ )
  {
    *(cep+i) = 0.0;
  }
}


dur_reset( cep, new_cep, unit )
float *cep;                        /* pointer for cepstrum BEFORE
                                      duration reset */
float *new_cep;          /* pointer for duration reset cep data */
```

```
Phoneme unit;
{

  int target_len;               /* terget duration (in frame) */
  int original_len;             /* original duration ( in frame ) */

  target_len = unit.length;
  original_len = unit.phn->end - unit.phn->stt;
  if( original_len == 0 )       original_len = 1;
  if( strcmp( unit.phn->phn, "p" ) == 0 ||
      strcmp( unit.phn->phn, "t" ) == 0 ||
      strcmp( unit.phn->phn, "k" ) == 0  )
/* [ Yamazaki 1 ]
   if  PTK then execute nonlinear interpol.
*/
  {
    if( non_linear_interpolate( original_len,
                                target_len,
                                cep,
                                new_cep,
                                unit ) < 0 )
    {
      fprintf(E,"error in linear interpolation.\n");
      return(-1);
    }
  }
  else
  {
    if(linear_interpolate( original_len, target_len, cep, new_cep) < 0)
    {
      fprintf(E,"error in linear interpolation.\n");
      return(-1);
    }
  }
  return(1);
}

linear_interpolate( l_x, l_y, x, y )
int l_x;        /* length of array before inter polation */
int l_y;        /* length of array after inter polation */
float *x;       /* array of [0,31] before inter polation */
float *y;       /* array of [0,31] after inter polation */
{
  float ratio;  /* shortening or lengthening ratio
                   if more than 1 -----> lengthening
                   if less than 1 -----> shortening */
  int x_suf, y_suf;     /* suffics of array x and y */
  register int i;

  ratio = (float)l_y/(float)l_x;

  if( ratio < 1.0 )
  {
    for( i = 0; i < l_x; i++ )
    {
      x_suf = i*Order;
      y_suf = round((float)i*ratio)*Order;
      array_copy( Order, y + y_suf, x + x_suf );
    }
  }
  else
  {
    for( i = 0; i < l_y; i++ )
    {
      x_suf = round((float)i/ratio)*Order;
```

```
      y_suf = i*Order;
      array_copy( Order, y + y_suf, x + x_suf );
    }
  }
}

/*
  Non-linear interpolation of PTK frame data
  Coded by T.Yamazaki, Shizuoka Univ.
  Aug. 9, 1988 at ATR
*/
non_linear_interpolate( l_x, l_y, x, y, unit)
int l_x;        /* length of array before inter polation */
int l_y;        /* length of array after inter polation */
float *x;       /* array of [0,31] before inter polation */
float *y;       /* array of [0,31] after inter polation */
Phoneme unit;
{
  float ratio;  /* shortening or lengthening ratio
                   if more than 1 -----> lengthening
                   if less than 1 -----> shortening */
  float ratio2;
  float mod_coef[2];
  int x_suf, y_suf;     /* suffics of array x and y */
  register int i;
  int diff;             /* l_x-l_y or l_y-l_x */
  int burst_point;
  char label_symbol;
  int closure_len, steady_len, total_frame_len;
  int event;            /* event size */
  int temp;

  ratio = (float)l_y/(float)l_x;

  if ((event = unit.phn->eve_size) == 0)
    fprintf(E,"Oh, my GOD!!!\n");

  switch(*(unit.phn->phn))
  {
/*
  mod_coef[0] represents lengthen/shortening ratio of closure portion
  mod_coef[1] represents lengthen/shortening ratio of sterady portion
*/
  case 'p':
    mod_coef[0] = 0.9;
    mod_coef[1] = 0.1;
    break;
  case 't':
    mod_coef[0] = 0.85;
    mod_coef[1] = 0.15;
    break;
  case 'k':
    mod_coef[0] = 0.7;
    mod_coef[1] = 0.3;
    break;
  default:
    fprintf(E,"Wuppp!!!\n" );
    break;
  }


  if (event > 1)
       /* case PTK consists of closure and steady portions */
  {
    closure_len = unit.phn->eve[0]->end - unit.phn->eve[0]->stt;
```

```
    steady_len =  unit.phn->eve[1]->end - unit.phn->eve[1]->stt;
    total_frame_len = closure_len + steady_len;

    if( ratio < 1.0 )
    {
      diff = l_x - l_y;
      for(i = 0; i < l_y; i++) {    /******** 21 DEC 88 Sagisaka ******/
/****************************** DELETE THIS PORTION 21 DEC 88 Sagisaka ****
      ratio2 = ((float)closure_len - diff * mod_coef[0])/(float)closure_len;
      for(i = 0; i < l_y; i++)
      {
        if (i < closure_len - round(diff * mod_coef[0]))
        {
          *************** Closure portion ***********
          x_suf = (int)((float)i/ratio2)*Order;
          y_suf = i*Order;
        }
        else if (i == closure_len - round(diff * mod_coef[0]))
        {
          *************** Burst point ***************
          x_suf = closure_len*Order;
          y_suf = i*Order;
          temp = i;
        }
        else
        {
          x_suf = (closure_len + i - temp)*Order;
          y_suf = i*Order;
        }
*** The upper portion is substituted by the lower one ** 21 DEC 88 ******/
          x_suf = ( i+diff )*Order;
          y_suf = i*Order;
/****************************************** 21 DEC 88 Sagisaka ******/
/***** append by N.Kaiki '90 Jan. 16 **********************************/
      generate_pause( diff, x );
/*****************************************************************/
      array_copy( Order, y + y_suf, x + x_suf);
      }
    }
    else
    {
      diff = l_y - l_x;

      burst_point = round(closure_len + diff * mod_coef[0]);

      ratio2 = ((float)burst_point)/(float)closure_len;

      for(i = 0; i < l_y; i++)
      {
        if (i <= burst_point -1)
        {
          x_suf = round((float)i/ratio2)*Order;
          y_suf = i*Order;
          array_copy( Order, y + y_suf, x + x_suf);
        }
        else if (i == closure_len + round(diff * mod_coef[0]))
        {
          x_suf = closure_len*Order;
          y_suf = i*Order;
          array_copy( Order, y + y_suf, x + x_suf);
          temp = i;
        }
        else
        {
          x_suf = (closure_len + i - temp)*Order;
```

```
          if ((closure_len + i - temp) > (total_frame_len - 1))
               x_suf = (total_frame_len - 1)*Order;
          y_suf = i*Order;
          array_copy( Order, y + y_suf, x + x_suf);
        }
      }
    }
  }
  else if(event == 1)        /* Event = only steady */
  {

    closure_len = ((diff = l_y - l_x) > NO_PAUSE ? diff : NO_PAUSE);
    steady_len  = unit.phn->eve[0]->end - unit.phn->eve[0]->stt;
    total_frame_len = closure_len + steady_len;

    if (diff < 0)            /* shortening */
    {
/************************************* 21 DEC 88 Sagisaka ***
      generate_pause( NO_PAUSE, y);
      ratio2 = (float)(l_y - NO_PAUSE)/(float)l_x;
      for (i = NO_PAUSE; i < l_y; i++)
      {
        x_suf = (int)((float)(i - NO_PAUSE)/ratio2)*Order;
***** Upper portion is substituted by the lower portion  *******/

      for (i = 0; i < l_y; i++)
      {        x_suf = (i-diff)*Order;
/************************************* 21 DEC 88 Sagisaka ***/
        y_suf = i*Order;
        array_copy( Order, y + y_suf, x + x_suf);
      }
    }
    else                     /* lengthening */
    {
      generate_pause( diff, y );
      for (i = diff; i < l_y; i++ )
      {
        x_suf = ( i - diff ) * Order;
        y_suf = i*Order;
        array_copy( Order, y + y_suf, x + x_suf );
      }
    }
  }
}


round(f)                 /* rounding ( shi-sha go-nyuu) */
float f;
{
  return((int)(f+0.5));
}

array_copy( size, a, b )
int size;
float a[], b[];
{
  register int i;
  for( i = 0; i < size; i++ )
  {
    a[i] = b[i];
  }
}
```

```
/*****************************************************************************/
/*                                                                           */
/*      Modify Power Parameters ( mdpower.c )                                */
/*                                                                           */
/*      This program read xxx.unit and xxx.CP file as input                  */
/*      and create xxx.PW file.                                              */
/*      Main task of this routine is control power pattern                   */
/*                                                                           */
/*---------------------------------------------------------------------------*/
/*                                                                           */
/*      Originally coded by K. Abe, ATR                 Jul. 11, 1989        */
/*                                                                           */
/*****************************************************************************/
#include <stdio.h>
#include <fcntl.h>
#include <sys/file.h>
#include "/SYN/include/Synthesis.h"
#include "mdpower.h"

#define E               stderr
#define Ord             32

/***** edited by K.Abe '89 Oct.03 ************************************************/
/*      'Max_unit','Max_unit_length' are defined in 'Synthesis.h'                */
/***** edited by K.Abe '89 Oct.03 ************************************************/
/***** edited by K.Abe '89 Sep.24 ************************************************/
/* #define Max_unit             512                                              */
/* #define Max_unit_length      128                                              */
/***** edited by K.Abe '89 Sep.24 ************************************************/

#define End_Edge                0
#define Start_Edge              1

char    *prog;

main( argc, argv )
int     argc;
char    *argv[];
{

/***** edited by K.Abe '89 Sep.24 ************************************************/
  Phoneme       u[Max_unit];            /* unit data */
/*****************************************************************************/
/*  Phoneme     u[128];                                                          */
/***** edited by K.Abe '89 Sep.24 ************************************************/

  int           fd_cep,                 /* file descriptor of cepstrum file */
                fd_pwr,                 /* file descriptor of power file */
                phns;                   /* number of phonemes */
  char          u_file[256],            /* unit filename */
                cep_file[256],          /* cepstrum filename */
                pwr_file[256],          /* power filename */
                *header;

  prog = argv[0];

/*------- check arguement consistency ---------------------------------------*/
  if( argc == 1 )
  {
    fprintf( E, "\t[%s] modify power pattern from xxx.unit and xxx.CP file.\n",
      prog );
    fprintf( E, "\t[ USAGE ] %s filename header\n", prog );
    exit( 0 );
  }
```

```
  header = argv[1];

/*------- read phoneme information ------------------------------------------*/
  if( ( phns=ReadPhoneme(header, u) ) <= 0 )
  {
    fprintf( E, "%s: u_file (%s) can't read.\n", prog, u_file );
    exit( 1 );
  }

/*------- open cepstrum file ------------------------------------------------*/
  sprintf( cep_file, "%s.CP", header );
  if( ( fd_cep=open(cep_file, O_RDONLY) ) < 0 )
  {
    fprintf( E, "%s: cepstrum file (%s) can't open.\n", prog, cep_file );
    exit( 1 );
  }

/*------- create power file -------------------------------------------------*/
  sprintf( pwr_file, "%s.PW", header );
  if( ( fd_pwr=open(pwr_file, O_WRONLY|O_CREAT, 0644) ) < 0 )
  {
    fprintf( E, "%s: power file (%s) can't open.\n", prog, pwr_file );
    exit( 1 );
  }

/*------- create power file -------------------------------------------------*/
  if( crt_pwr_file(fd_cep, fd_pwr, phns, u) < 0 )
  {
    fprintf( E, "%s: error in create power file.\n", prog );
    exit( 2 );
  }

/*------- ending ------------------------------------------------------------*/
  exit( 0 );
}


/******** create power file ***************************************************/
/*                                                                            */
/*      fd_cep: [i/o] cepstrum file descriptor                                 */
/*      fd_pwr: [i/o] power file descriptor                                   */
/*      phns:   [in ] number of phonemes                                      */
/*      u:      [in ] phoneme unit                                            */
/*                                                                            */
/******************************************************************************/
crt_pwr_file( fd_cep, fd_pwr, phns, u )
int     fd_cep, fd_pwr, phns;
Phoneme u[];
{

/***** edited by K.Abe '89 Sep. 24 ********************************************/
  float         power[Max_unit][Max_unit_length];       /* original power */
  float         org_power[Max_unit_length];             /* original power */
  float         mod_power[Max_unit_length];             /* modified power */
/******************************************************************************/
/*  float               power[128][256];                                      */
/*  float               org_power[256];                                       */
/*  float               mod_power[256];                                       */
/***** edited by K.Abe '89 Sep. 24 ********************************************/

  float         *cep;                   /* pointer for original cepstrum data */
  float         base_cep0;              /* base cepstrum[0] */
  float         rate;
  float         rate_int;
  float         power_int;              /* interpolated power */
```

```
  float         d_power;                /* power difference */
  int           rsize;                  /* size of cepstrum data */
  int           wsize;                  /* size of power data */
  int           code;                   /* code of phoneme kinds */
  int           p_code;                 /* code of previous phoneme kinds */
  register int  i, j;
  static int    cnt;

  cep = (float*)malloc( 12800 );

  base_cep0 = 4.25;

  rate = 0.5;

  for( i=0; i<phns; i++ )
  {
    rsize = u[i].length * Ord * sizeof(float);

/***** edited by K.Abe '89 Sep. 24 ********************************************/
    if( u[i].length >= Max_unit_length )
    {
      printf( " *** Too long unit length = %d > (%d) ***\n",
             u[i].length, Max_unit_length );
      exit();
    }
/***** edited by K.Abe '89 Sep. 24 ********************************************/

/*------- read cepstrum ------------------------------------------------------*/
    if( read(fd_cep, cep, rsize) != rsize )
    {
      fprintf( E, "cepstrum file read error in %dth unit.\n", i );
      return( -1 );
    }

    code = class( u[i].phn->phn, Start_Edge );

/*-------------------------------------------------------------------------------*/
/*    wr_class( code );                                                          */
/*-------------------------------------------------------------------------------*/

/*------- modify cepstrum[0] -------------------------------------------------*/
    for( j=0; j<u[i].length; j++ )
    {
      mod_power[j] = rate * *(cep+32*j+1);
      org_power[j] = *(cep+32*j+1);

/*-------------------------------------------------------------------------------*/
/*      d_cep0 = *(cep+32*j+1) - p_cep0;                                         */
/*      mod_cep0 = base_cep0 + rate*d_cep0;                                      */
/*      base_cep0 = mod_cep0;                                                    */
/*      p_cep0 = *(cep+32*j+1);                                                  */
/*      mod_power[j] = mod_cep0;                                                 */
/*-------------------------------------------------------------------------------*/

    }

/*------- write power --------------------------------------------------------*/
    switch( code )
    {
      case Vowels:
      case SemiVowels:
      case Nasals:
      case VoicedAffricates:
      case VoicedFricatives:
```

```
/*-------------------------------------------------------------------------------*/
/*      if( p_code != Vowels ) base_cep0 = org_power[0] - mod_power[0];          */
/*-------------------------------------------------------------------------------*/

/*        printf( " [ %4d ]", i );
        printf( " original=%6.2f   modify=%6.2f   base=%6.2f\n",
          org_power[0], mod_power[0], base_cep0 );                               */
        for( j=0; j<u[i].length; j++ )
        {
          power[i][j] = base_cep0 + mod_power[j];
          if( power[i][j] >= org_power[j] ) power[i][j] = org_power[j];
        }
        break;
      case Unknown:
      case VoicedStops:
      case VoicelessStops:
      case VoicelessAffricates:
      case VoicelessFricatives:
      default:
        for( j=0; j<u[i].length; j++ )
        {
          power[i][j] = org_power[j];
        }
        break;
    }
    p_code = code;
  }

/*------- interpolate power between units ---------------------------------------*/
  for( i=0; i<phns; i++ )
  {

/*-------------------------------------------------------------------------------*/
/*    printf( "***** %2d ***** ", i );                                           */
/*    printf( " [word no.] %4d (%4s) [original] %3d [target] %3d\n",             */
/*      u[i].unit_no, u[i].phn->phn, u[i].phn->end-u[i].phn->stt, u[i].length );*/
/*-------------------------------------------------------------------------------*/

    if( u[i].unit_no != u[i+1].unit_no )
    {

/*-------------------------------------------------------------------------------*/
/*      printf( "\n" );                                                          */
/*-------------------------------------------------------------------------------*/

      power_int = ( power[i][u[i].length-1] + power[i+1][0] ) / 2.;

/*-------------------------------------------------------------------------------*/
/*      printf( "  %6.2f\n  %6.2f\n  %6.2f\n  %6.2f\n  %6.2f\n",                */
/*      power[i][u[i].length-5], power[i][u[i].length-4],                        */
/*      power[i][u[i].length-3], power[i][u[i].length-2],                        */
/*      power[i][u[i].length-1] );                                               */
/*      printf( " [ interpolate power ] %6.2f\n", power_int );                   */
/*      printf( "  %6.2f\n  %6.2f\n  %6.2f\n\n",                                */
/*      power[i+1][0], power[i+1][1], power[i+1][2] );                           */
/*-------------------------------------------------------------------------------*/

/*
      d_power = ( power[i+1][0] - power[i][u[i].length-5] ) / 5.;
      power[i][u[i].length-4] = power[i][u[i].length-5] + d_power;
      power[i][u[i].length-3] = power[i][u[i].length-5] + 2.*d_power;
      power[i][u[i].length-2] = power[i][u[i].length-5] + 3.*d_power;
      power[i][u[i].length-1] = power[i][u[i].length-5] + 4.*d_power;
*/
```

```
/*-------------------------------------------------------------------------------*/
/*      printf( "  %6.2f\n  %6.2f\n  %6.2f\n  %6.2f\n  %6.2f\n",                 */
/*      power[i][u[i].length-5], power[i][u[i].length-4],                        */
/*      power[i][u[i].length-3], power[i][u[i].length-2],                        */
/*      power[i][u[i].length-1] );                                               */
/*      printf( " [ interpolate power ] %6.2f\n", power_int );                   */
/*      printf( "  %6.2f\n  %6.2f\n  %6.2f\n\n",                                 */
/*      power[i+1][0], power[i+1][1], power[i+1][2] );                           */
/*-------------------------------------------------------------------------------*/

    }
  }

/*------- write power ---------------------------------------------------------*/
  for( i=0; i<phns; i++ )
  {
    wsize = u[i].length * sizeof(float);
    if( write(fd_pwr, power[i], wsize) != wsize )
    {
      fprintf( E, "power file write error in %dth unit.\n", i );
      return( -1 );
    }
  }
}

/******** classify phoneme ***************************************************/
/*                                                                           */
/*      s:              [in ] phoneme symbol                                 */
/*      se_flag:        [in ]                                                */
/*                                                                           */
/*****************************************************************************/
class( s, se_flag )
char    *s;
int     se_flag;
{
  static struct
  {
    int         phns;                   /* number of phonemes in the class */
    int         class;                  /* phoneme class identification number */
    char        *phn[10];               /* phoneme symbols */
  } tab[10] =
  {
    { 6,        Vowels,                 { "a","i","u","e","o","N","","","","" } },
    { 3,        SemiVowels,             { "r","w","y","","","","","","","" } },
    { 2,        Nasals,                 { "m","n","","","","","","","","" } },
    { 3,        VoicedStops,            { "b","d","g","","","","","","","" } },
    { 1,        VoicedAffricates,       { "j","","","","","","","","","" } },
    { 1,        VoicedFricatives,       { "z","","","","","","","","","" } },
    { 3,        VoicelessStops,         { "p","t","k","","","","","","","" } },
    { 2,        VoicelessAffricates,    { "ts","ch","","","","","","","","" } },
    { 4,        VoicelessFricatives,    { "s","sh","h","f","","","","","","" } },
    { 2,        Edges,                  { "Top","End","","","","","","","","" } }
  };

  int           classes = 10;           /* number of classes */
  register int  i, j;
  char          *top(), *tail();

  if( size(s) > 1 )
  {
    if( se_flag == Start_Edge ) return( class(top(s), se_flag) );
    else return( class(tail(s), se_flag) );
  }
  else
  {
```

```
    for( i=0; i<classes; i++ )
    {
      for( j=0; j<tab[i].phns; j++ )
        if( strcmp(s, tab[i].phn[j]) == 0 ) return( tab[i].class );
    }
    return( Unknown );
  }
}

/******** write phoneme class code ***********************************************/
/*                                                                               */
/*      code:   [in ] phoneme class code                                         */
/*                                                                               */
/*********************************************************************************/
wr_class( code )
int     code;
{
  switch( code )
  {
    case Unknown:
      printf( "[ Unknown             ]\n" );
      break;
    case VoicelessStops:
      printf( "[ VoicelessStops      ]\n" );
      break;
    case VoicelessAffricates:
      printf( "[ VoicelessAffricates ]\n" );
      break;
    case Nasals:
      printf( "[ Nasals              ]\n" );
      break;
    case Vowels:
      printf( "[ Vowels              ]\n" );
      break;
    case VoicedAffricates:
      printf( "[ VoicedAffricates    ]\n" );
      break;
    case VoicedFricatives:
      printf( "[ VoicedFricatives    ]\n" );
      break;
    case SemiVowels:
      printf( "[ SemiVowels          ]\n" );
      break;
    case VoicelessFricatives:
      printf( "[ VoicelessFricatives ]\n" );
      break;
    case VoicedStops:
      printf( "[ VoicedStops         ]\n" );
      break;
    case Edges:
      printf( "[ Edges               ]\n" );
      break;
  }
}

/******** get top phoneme ********************************************************/
/*                                                                               */
/*      s:      [in ] phoneme symbol                                             */
/*                                                                               */
/*********************************************************************************/
char    *top( s )
char    *s;
{
  char          *ptr;
  register int  i;
```

```
  ptr = (char*)malloc( strlen(s)+1 );
  strcpy( ptr, s );
  for( i=0; i<strlen(ptr); i++ )
  {
    if( *(ptr+i) == ',' )
    {
      *(ptr+i) = '\0';
      break;
    }
  }
  if( *(ptr+i-1) == 'y' ) *(ptr+i-1) = '\0';
  return( ptr );
}

/******** get tail phoneme ************************************************/
/*                                                                        */
/*      s:      [in ] phoneme symbol                                      */
/*                                                                        */
/**************************************************************************/
char    *tail( s )
char    *s;
{
  register int  i;
  char          *ptr;

  for( i=strlen(s)-1; i>=0; i-- )
  {
    ptr = s+i;
    if( *(s+i) == ',' )
    {
      ptr++;
      break;
    }
  }
  if( *(ptr+strlen(ptr)-1) == 'y' ) return( ptr+strlen(ptr)-1 );
  return( ptr );
}

/******** get size phoneme ************************************************/
/*                                                                        */
/*      s:      [in ] phoneme symbol                                      */
/*                                                                        */
/**************************************************************************/
int     size( s )
char    *s;
{
  char  c;
  int   count = 1;

  while( c = *s++ )     if( c == ',' ) count++;
  return( count );
}
```

```
/*****************************************************************************/
/*                                                                           */
/*      Synthesis Program in Cepstrum Vocoder ( lma_pwr.c )                  */
/*                                                                           */
/*              - synthesize speech from *.CP, *.PT and *.PW file -          */
/*              - using LMA filter -                                         */
/*                                                                           */
/*      Compiled by:                                                         */
/*              cc -o lma lma.c -lm                                          */
/*                                                                           */
/*      [ USAGE ]                                                            */
/*              lma_pwr FILE                                                 */
/*                                                                           */
/*        Input:                                                             */
/*              FILE:   input filename                                       */
/*                                                                           */
/*---------------------------------------------------------------------------*/
/*      Version 1.0     coded by k.abe                  4/13/88              */
/*              1.1     edited by k.abe                 4/22/88              */
/*              1.2     edited by k.abe                 8/08/88              */
/*              1.3     edited by k.abe                 5/30/89              */
/*              1.3     edited by k.abe                 7/11/89              */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>

int     nframe;                 /* frame number */

main( argc, argv )
int     argc;
char    *argv[];
{
  float cep[64];                /* current frame cepstrum */
  float cep_n[64];              /* next frame cepstrum */
  char  F_PT_CP[128];           /* pitch(f)-cepstrum(f) filename */
  char  F_PT[128];              /* pitch(i) filename */
  char  F_PW[128];              /* power(f) filename */
  char  F_WV[128];              /* synthesized speech wave filename */
  short out_sp[128];            /* synthesized output speech */

  float pwr;                    /* current frame power */
  float pwr_n;                  /* next frame power */
  float rnd();                  /* random noise generation routine */
  int   fd_pit_cep;             /* pitch(f)-cepstrum(f) file descriptor */
  int   fd_pit;                 /* pitch(i) file descriptor */
  int   fd_pwr;                 /* power(f) file descriptor */
  int   fd_syn;                 /* synthesized speech file descriptor */
  int   ptr;                    /* pulse pointer */
  int   opit;                   /* current frame original pitch period */
  int   opit_n;                 /* next frame original pitch period */
  int   pit;                    /* current frame pitch period */
  int   pit_n;                  /* next frame pitch period */

  float shift_cep0 = 0.5;       /* cepstrum[0] shift value */
  int   sframe = 60;            /* shift frame = 5ms */
  int   ord_cep = 30;           /* cepstrum order */

  register int  i;

  static double filt_io[128];   /* LMA filter input/output */
  static double filt_buf[1024]; /* LMA filter buffer */

  rnd( 9 );
```

```
/*------- check arguement consistency ----------------------------------------*/
  if( argc != 2 )
  {
    printf( "[ USAGE ] lma FILE\n" );
    exit();
  }

  strcpy( F_PT_CP, argv[1] );
  strcat( F_PT_CP, ".CP" );              /* pitch(f)-cepstrum(f) filename */
  strcpy( F_PT, argv[1] );
  strcat( F_PT, ".PT" );                 /* pitch(i) filename */
  strcpy( F_PW, argv[1] );
  strcat( F_PW, ".PW" );                 /* power(f) filename */
  strcpy( F_WV, argv[1] );
  strcat( F_WV, ".WV" );                 /* synthesized speech filename */

/*------- open files ---------------------------------------------------------*/
  if( ( fd_pit_cep=open(F_PT_CP, 0) ) == -1 ) open_err( F_PT_CP );
  if( ( fd_pit=open(F_PT, 0) ) == -1 ) open_err( F_PT );
  if( ( fd_pwr=open(F_PW, 0) ) == -1 ) open_err( F_PW );
  fd_syn = creat( F_WV, 0644 );

/*------- initialize counters ------------------------------------------------*/
  nframe = 0;
  ptr = 0;

/*------- read cepstrum data ( current frame ) -------------------------------*/
  if( read(fd_pit_cep, &opit, 4) == 0 ) eof( F_PT_CP );
  if( read(fd_pit_cep, cep, 4*ord_cep+4) == 0 ) eof( F_PT_CP );
  if( read(fd_pit, &pit, 2) == 0 ) eof( F_PT );
  if( read(fd_pwr, &pwr, 4) == 0 ) eof( F_PW );
  cep[0] = pwr;
  cep[0] -= shift_cep0;

/*------- read cepstrum data ( next frame ) ----------------------------------*/
  while( ++nframe > 0 )
  {
    if( read(fd_pit_cep, &opit_n, 4) == 0 ) eof( F_PT_CP );
    if( read(fd_pit_cep, cep_n, 4*ord_cep+4) == 0 ) eof( F_PT_CP );
    if( read(fd_pit, &pit_n, 2) == 0 ) eof( F_PT );
    if( read(fd_pwr, &pwr_n, 4) == 0 ) eof( F_PW );
    cep_n[0] = pwr_n;
    cep_n[0] -= shift_cep0;

/*------- generate filter exciting function ----------------------------------*/
    plsg( filt_io, &ptr, sframe, pit );

/*------- digital filtering --------------------------------------------------*/
    digfi( filt_io, filt_buf, cep, cep_n, ord_cep, sframe );

/*------- write speech data --------------------------------------------------*/
    for( i=0; i<sframe; i++ ) out_sp[i] = (short)filt_io[i];
    write( fd_syn, out_sp, 2*sframe );

/*------- shift data ---------------------------------------------------------*/
    f_copy( cep, cep_n, ord_cep+1 );
    pit = pit_n;
    pwr = pwr_n;
  }
}


/******** clear double array ******************************************************/
/*                                                                                */
/*      x:      [out] cleared double array                                        */
```

```
/*      n:      [in ] array length                                              */
/*                                                                              */
/********************************************************************************/
d_zero( x, n )
double  x[];
int     n;
{
  int   i;

  for( i=0; i<n; i++ )          x[i] = 0.;
}

/******** copy x from y ( floating ) ******************************************/
/*                                                                              */
/*      x:      [out] copied float array                                        */
/*      y:      [in ] source float array                                        */
/*      n:      [in ] copy length                                               */
/*                                                                              */
/********************************************************************************/
f_copy( x, y, n )
float   x[], y[];
int     n;
{
  int   i;

  for( i=0; i<n; i++ )          x[i] = y[i];
}

/******** generate pulse train ************************************************/
/*                                                                              */
/*      filt_io:        [out] filter exciting function                          */
/*      ptr:            [i/o] pulse pointer                                     */
/*      sframe:         [in ] frame shift                                       */
/*      pit:            [in ] pitch period                                      */
/*                                                                              */
/********************************************************************************/
plsg( filt_io, ptr, sframe, pit )
double  filt_io[];
int     *ptr, sframe, pit;
{
  int           i;
  static double pulse_amp = 1.0;        /* pulse amplitude */
  static double noise_amp = 0.125;      /* noise amplitude */

  d_zero( filt_io, sframe );

/*------- generate random noise -------------------------------------------------*/
  if( pit == 0 )
  {
    for( i=(*ptr); i<sframe; i++ )
    {
      filt_io[i] = 2.*rnd(1) - 1.;
      if( filt_io[i] >= 0.0 ) filt_io[i] = noise_amp;
      else if( filt_io[i] < 0.0 ) filt_io[i] = -noise_amp;
    }
    *ptr = 0;
    return;
  }

/*------- generate pulse train --------------------------------------------------*/
  while( *ptr < sframe )
  {
    filt_io[*ptr] = pulse_amp;
    *ptr += pit;
  }
```

```
  *ptr -= sframe;
}

/******** digital filter ************************************************/
/*                                                                      */
/*      filt_io:        [in ] filter exciting function                  */
/*                      [out] synthesized speech                        */
/*      filt_buf:       [i/o] filter buffer                             */
/*      cep:            [in ] current frame cepstrum                    */
/*      cep_n:          [in ] next frame cepstrum                       */
/*      ord_cep:        [in ] cepstrum order                            */
/*      sframe:         [in ] frame shift                               */
/*                                                                      */
/************************************************************************/
digfi( filt_io, filt_buf, cep, cep_n, ord_cep, sframe )
double  filt_io[], filt_buf[];
float   cep[], cep_n[];
int     ord_cep, sframe;
{
  double        cep_diff[64];   /* difference between cep_n & cep */
  double        cep_itpl[64];   /* interpolated cepstrum */
  int           i, j, lv;
  static int    itvl_itpl = 15; /* interpolation interval */

  if( ord_cep < 0 ) return;
  lv = sframe / itvl_itpl;

  for( i=0; i<=ord_cep; i++ )
       cep_diff[i] = ((double)cep_n[i] - (double)cep[i])/(double)lv;

  for( j=1; j<=lv; j++ )
  {
    for( i=0; i<=ord_cep; i++ )
         cep_itpl[i] = (double)cep[i] + cep_diff[i]*(double)j;
    for( i=(j-1)*itvl_itpl; i<j*itvl_itpl; i++ )
    {
      lma( &(filt_io[i]), filt_buf, cep_itpl, ord_cep );
      if( fabs(filt_io[i]) > 32767. )
           printf( "\n [ %5d - %2d ] cep[0]=%4.1f !!! overflow !!! x=%6.0f\n",
                  nframe, i, cep_itpl[0], filt_io[i] );
    }
  }
}

/******** log magnitude approximate filter ******************************/
/*                                                                      */
/*      x:              [i/o] input/output of LMA filter                */
/*      filt_buf:       [i/o] filter buffer                             */
/*      cep:            [in ] cepstrum                                  */
/*      ord_cep:        [in ] cepstrum order                            */
/*                                                                      */
/************************************************************************/
lma( x, filt_buf, cep, ord_cep )
double  *x, filt_buf[], cep[];
int     ord_cep;
{
  double        w0, x0, wn1[3];
  int           i, i0, j, k, ms, ms0;
  static double fk1 = 0.999071235 / 2.;
  static double fk2[3] = { 0., 0.999870613/2., 0.994034994/2. };
  static int    l1[64] = { 1, 3, 3, 2, 2, 2, 2, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
                        1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                        1, 1, 1, 1 };

  *x *= exp( cep[0] );

  ms = 0;

  for( i0=1; i0<=ord_cep; i0++ )
       for( i=1; i<=ll[i0]; i++ )
       {
         x0 = *x;
         ms0 = ms;
         for( j=1; j<=2; j++ )
         {
           if( i0 != 1 )
                for( k=1; k<i0; k++ )
                {
                  w0 = filt_buf[++ms];
                  filt_buf[ms] = *x;
                  *x = w0;
                }
           w0 = filt_buf[++ms];
           filt_buf[ms] = *x;
           *x = w0 * cep[i0];
           if( ll[i0] == 1 ) wn1[j] = *x * fk2[j];
           else if( j == 1 ) wn1[j] = *x * fk2[j] / (double)ll[i0];
           else wn1[j] = *x * fk2[j] / ((double)ll[i0] * (double)ll[i0]);
           if( fabs(wn1[j]) <= 1.0e-30 ) wn1[j] = 0.;
         }
         filt_buf[ms0+1] = filt_buf[ms0+1] + wn1[1] - wn1[2];
         *x = x0 + 2.*wn1[1];
       }
}

/******** generate random noise ************************************************/
/*                                                                             */
/*      x:      [i/o] input/output of random routine                           */
/*                                                                             */
/*******************************************************************************/
float   rnd( x )
short   x;
{
  static short  ix=1, init_on=0;

  if( ((x % 2)!=0) && (init_on==0) )
  {
    ix = x;
    init_on = 1;
  }                     /* set init flag */

  ix *=  899;
  if( ix < 0 ) ix = ix + 32767 + 1;

  return( (float)ix/32768.0 );
}

/******** open error ***********************************************************/
/*                                                                             */
/*      file:   [in ] filename which doesn't exist                             */
/*                                                                             */
/*******************************************************************************/
open_err( file )
char    file[];
{
  printf( " * Open Error ! %s\n", file );
```

```
  exit();
}

/******** End Of File ************************************************************/
/*                                                                               */
/*      file:   [in ] filename which is on End Of File                           */
/*                                                                               */
/*******************************************************************************/
eof( file )
char    file[];
{
  printf( " * End Of File ! %s [ %d ]\n", file, nframe );
  exit();
}
```

Appendix2

ライブラリプログラム

GetDuration
ReadDuration
GetEntity
GetFrameOutLine
ReadFrameOutLine
GetLabel
ReadLabel
GetPhoneme
ReadPhoneme
GetPhonemeOutLine
ReadPhonemeOutLine
GetPhrase
ReadPhrase
GetUnitAttribute
ReadUnitAttribute
PutPhoneme
WritePhoneme
PutUnitAttribute
WriteUnitAttribute

```
#include "/usr4/takeda/SYN/include/Synthesis.h"
GetDuration( fp, dur )
FILE *fp;
DurationData dur[];
{
  char ph[64];
  float dr;
  register int i = 0;
  while( fscanf( fp, "%s %f", ph, &dr ) != EOF )
  {
    dur[i].phn = (char*)malloc(strlen(ph)+1);
    strcpy( dur[i].phn, ph );
    dur[i].len = (int)(dr/Ratio);
    i++;
  }
  return(i);
}
#include "/usr4/takeda/SYN/include/Synthesis.h"
GetEntity( fp, ent )
FILE *fp;
Entity ent[];
{
  register int i = 0;
  int st, en, wd;
  char rm[128], wdr[128];

  while( fscanf( fp, "%d %d %d %s %s", &wd, &st, &en, rm, wdr ) != EOF )
  {
    ent[i].word = wd;
    ent[i].stt = st;
    ent[i].end = en;
    ent[i].roman = (char*)malloc(strlen(rm)+1);
    strcpy( ent[i].roman, rm );
    ent[i].wd_roman = (char*)malloc(strlen(wdr)+1);
    strcpy( ent[i].wd_roman, wdr );
    i++;
  }
  return(i);
}
#include "/usr4/takeda/SYN/include/Synthesis.h"
GetFrameOutLine( fp, frame )
FILE *fp;
FrameOutLine frame[];
{
  int no, wd, fm;
  char phn[62], eve[62], alp[64];
  register int i = 0;

  while( fscanf( fp, "%d %d %d %s %s %s", &no, &wd, &fm, phn, eve, alp )
        != EOF )
  {
    frame[i].no = no;
    frame[i].word = wd;
    frame[i].frm = fm;
    frame[i].phn = (char*)malloc(strlen(phn)+1);
    strcpy( frame[i].phn, phn );
    frame[i].eve = (char*)malloc(strlen(eve)+1);
    strcpy( frame[i].eve, eve );
    frame[i].alp = (char*)malloc(strlen(alp)+1);
    strcpy( frame[i].alp, alp );
    i++;
  }
  return(i);
}
#include "/SYN/include/Synthesis.h"
```

```
GetLabel( fp, phn )
FILE *fp;
Phone phn[];
{

  Event ev[256];
  Alophone al[256];

  struct { int stt, end; char *sym; } work[3][256];

  char l[256];
  int size[3];          /* size of each layer of label */
  int layer = 0;        /* layer no of work array
                                0; phonemic label layer
                                1; event label layer
                                2; alophonic variation layer
                        */

  register int i, j, k, label = 0;

  while( fgets( l, 256, fp ) != 0 )
  {
    char lb[64];        /* read buffer for label symbol */
    float f_stt, f_end; /* read buffer for label start and end */

    if( l[0] == '#' )
    /* read next layer */
    {
      size[layer] = label;
      if( ++layer > 2 ) break;
      label = 0;
    }
    else
    {
      sscanf( l, "%f %s %f", &f_stt, lb, &f_end );
      work[layer][label].stt = f_stt/Ratio;
      work[layer][label].end = f_end/Ratio;
      work[layer][label].sym = (char*)malloc( strlen(lb)+1);
      strcpy( work[layer][label].sym, lb );
      label++;
    }
  }

  /* set Phoneme data */
  for( i = 0; i < size[0]; i++ )
  {
    phn[i].stt = work[0][i].stt;
    phn[i].end = work[0][i].end;
    phn[i].phn = (char*)malloc(strlen(work[0][i].sym)+1);
    strcpy( phn[i].phn, work[0][i].sym );
  }
  /* set Event data */
  for( i = 0; i < size[1]; i++ )
  {
    ev[i].stt = work[1][i].stt;
    ev[i].end = work[1][i].end;
    ev[i].eve = (char*)malloc(strlen(work[1][i].sym)+1);
    strcpy( ev[i].eve, work[1][i].sym );
  }
  /* set Alophone data */
  for( i = 0; i < size[2]; i++ )
  {
    al[i].stt = work[2][i].stt;
    al[i].end = work[2][i].end;
    al[i].alp = (char*)malloc(strlen(work[2][i].sym)+1);
```

```
    strcpy( al[i].alp, work[2][i].sym );
  }

  for( i = 0; i < size[0]; i++ )
  {
    int eve_size = 0;   /* event included in the i-th phoneme */
    int alp_size = 0;   /* alophonic symbols included in the i-th phoneme */

    /* search event that is included in the phoneme */
    for( j = 0; j < size[1]; j++ )
    {
      if( phn[i].stt <= ev[j].stt &&
          phn[i].end >= ev[j].end )
      {
        phn[i].eve[eve_size] = (Event*)malloc(sizeof(Event));
        phn[i].eve[eve_size]->stt = ev[j].stt;
        phn[i].eve[eve_size]->end = ev[j].end;
        phn[i].eve[eve_size]->eve = (char*)malloc(strlen(ev[j].eve)+1);
        strcpy( phn[i].eve[eve_size]->eve, ev[j].eve );
        eve_size++;
      }
    }
    /* set including event size */
    phn[i].eve_size = eve_size;

    /* search alophonic symbol that is included in the phoneme */
    for( j = 0; j < size[2]; j++ )
    {
      if( phn[i].stt <= al[j].stt &&
          phn[i].end >= al[j].end )
      {
        phn[i].alp[alp_size] = (Alophone*)malloc(sizeof(Alophone));
        phn[i].alp[alp_size]->stt = al[j].stt;
        phn[i].alp[alp_size]->end = al[j].end;
        phn[i].alp[alp_size]->alp = (char*)malloc(strlen(al[j].alp)+1);
        strcpy( phn[i].alp[alp_size]->alp, al[j].alp );
        alp_size++;
      }
    }
    /* set included alophonic symbol size */
    phn[i].alp_size = alp_size;
  }

  /* set unit full length */
  return(size[0]);
}

#include "/usr4/takeda/SYN/include/Synthesis.h"
GetPhoneme( p, u )
FILE *p;
Phoneme u[];
{
  char fl[32], lb[63];
  int no, len, inf, unit, st, en, asz, esz;
  register int i = 0, j;
  for(;;)
  {
    if( fscanf( p, "%s %d %d %d %d", fl, &no, &len, &inf, &unit ) == EOF )
    {
      return(i);
    }
    if( fl[0] != '#' )
    {
      fprintf(stderr,"read_unit: file format error(record no = %d)\n", i );
      return(-1);
```

```
    }
    else
    {
      u[i].length = len;
      u[i].inf = inf;
      u[i].unit_no = unit;
    }
    if( fscanf(p,"%s %d %s %d %d %d", fl, &st, lb, &en, &esz, &asz ) == EOF ||
        fl[0] != 'p' )
    {
      fprintf(stderr,"read_unit: file format error(record no =%d)\n", i );
      return(-1);
    }
    else
    {
      u[i].phn = (Phone*)malloc(sizeof(Phone));
      u[i].phn->stt = st;
      u[i].phn->phn = (char*)malloc(strlen(lb)+1);
      strcpy( u[i].phn->phn, lb );
      u[i].phn->end = en;
      u[i].phn->eve_size = esz;
      u[i].phn->alp_size = asz;

      /* read event enformations */
      for( j = 0; j < esz; j++ )
      {
        if( fscanf(p,"%s %d %s %d", fl, &st, lb, &en ) == EOF ||
            fl[0] != 'e' )
        {
          fprintf(stderr,"read_unit: file format error(record no = %d)\n", i );
          return(-1);
        }
        else
        {
          u[i].phn->eve[j] = (Event*)malloc(sizeof(Event));
          u[i].phn->eve[j]->stt = st;
          u[i].phn->eve[j]->eve = (char*)malloc(strlen(lb)+1);
          strcpy( u[i].phn->eve[j]->eve, lb );
          u[i].phn->eve[j]->end = en;
        }
      }
      /* read alponic informations */
      for( j = 0; j < asz; j++ )
      {
        if( fscanf(p,"%s %d %s %d", fl, &st, lb, &en ) == EOF ||
            fl[0] != 'a' )
        {
          fprintf(stderr,"read_unit: file format error(record no = %d)\n", i );
          return(-1);
        }
        else
        {
          u[i].phn->alp[j] = (Alophone*)malloc(sizeof(Alophone));
          u[i].phn->alp[j]->stt = st;
          u[i].phn->alp[j]->alp = (char*)malloc(strlen(lb)+1);
          strcpy( u[i].phn->alp[j]->alp, lb );
          u[i].phn->alp[j]->end = en;
        }
      }
    }
    i++;
  }
}
#include "/SYN/include/Synthesis.h"
GetPhonemeOutLine( fp, ph )
```

```
FILE *fp;
PhonemeOutLine ph[];
{
  register int i = 0, j = 0;
  char line[256];
  int ags;
  int ac;
  char *av[24];

  while( fgets( line, 256, fp ) != 0 )
  {
    ac = ( line[0] == ' ' ? 1:0);
    ags = bd( line, av );
    ph[i].no = atoi( av[ac++] );
    ph[i].length = atoi( av[ac++] );
    ph[i].inf = atoi( av[ac++] );
    ph[i].word = atoi( av[ac++] );
    ph[i].stt = atoi( av[ac++] );
    ph[i].end = atoi( av[ac++] );
    ph[i].o_length = atoi( av[ac++] );
    ph[i].phn = av[ac++];
    if( ags <= ac )
    {
      fprintf( stderr, "The .PN file is old format with only 8 fields.\n" );
      i++;
      continue;
    }
    ph[i].rp_size = atoi(av[ac++]);
    for( j = 0; j < ph[i].rp_size; j++ )
         ph[i].rp[j] = atoi( av[ac++] );
    i++;
  }
  return(i);
}
bd( line, av )
char line[], *av[];
{
  register int i;
  int num = 0;
  char buf[32];


  strcpy( buf, "" );
  for( i = 0 ; i < strlen(line) && line[i] != '\n'; i++ )
  {
    if( line[i] == ' ' || line[i] == '\t' )
    {
      av[num] = (char*)malloc(strlen(buf)+1);
      strcpy( av[num], buf );
      strcpy( buf, "" );
      num++;
      if( line[i] == ' ' || line[i] == ',' || line[i] == '\t' )
      {
        while( line[i] == ' ' || line[i] == ',' || line[i] == '\t' ) i++;
        --i;
      }
    }
    else
    {
      sprintf( buf, "%s%c", buf, line[i] );
    }
  }
  av[num] = (char*)malloc(strlen(buf)+1);
  strcpy( av[num], buf );
  num++;
```

```
  return(num);
}

#include "/usr4/takeda/SYN/include/Synthesis.h"
GetPhrase(fp,phrase)
FILE *fp;
Phrase phrase[];
{
/*
  [Takeda May, 1989]
   A new data field "pt" is added
   [Takeda Sep 4, 1989]
   A new field "boundary is added"
 */
  register int i;
  int ac;
  char *av[7];
  char line[256];

  i = 0;
  while( fgets( line, 256, fp ) != 0 )
  {
    if(( ac = break_down( line, av )) < 6 )
    {
      fprintf( stderr, "GetPhrase: data format error\n" );
      return(-1);
    }
    phrase[i].phrase_id = atoi( av[0] );
    phrase[i].string = (char *)malloc(strlen(av[1])+1);
    strcpy( phrase[i].string, av[1] );
    phrase[i].mora = atoi( av[2] );
    phrase[i].accent = atoi( av[3] );
    phrase[i].pause = atoi( av[4] );
    phrase[i].pitch = atoi( av[5] );
    if( ac == 6 )
    {
      fprintf( stderr,
      "Warning: 6-field .PH file. 0 is assumed for the last fiedl!!\n" );
      phrase[i].boundary = 0;
    }
    else
    {
      phrase[i].boundary = atoi( av[6] );
    }
    i++;
  }
  return(i);
}


/* This program parses the input char-array and
   breaks its into strings at spaces or tab.
   The elemental strings will be stored in av
   and the number of elements will be returned.
*/

break_down( line, av )
char line[], *av[];
{
  register int i;
  int num = 0;
  char buf[32];


  strcpy( buf, "" );
```

```
  for( i = 0 ; i < strlen(line) && line[i] != '\n'; i++ )
  {
    if( line[i] == ' ' || line[i] == ',' || line[i] == '\t' )
    {
      av[num] = (char*)malloc(strlen(buf)+1);
      strcpy( av[num], buf );
      strcpy( buf, "" );
      num++;
      if( line[i] == ' ' || line[i] == ',' || line[i] == '\t' )
      {
        while( line[i] == ' ' || line[i] == ',' || line[i] == '\t' ) i++;
        --i;
      }
    }
    else
    {
      sprintf( buf, "%s%c", buf, line[i] );
    }
  }
  av[num] = (char*)malloc(strlen(buf)+1);
  strcpy( av[num], buf );
  num++;
  return(num);
}

#include "/SYN/include/Synthesis.h"
GetUnitAttribute(fp,u)
FILE *fp;
UnitAttribute u[];
{
  register int i = 0, j, k;
  int wd, st, en, nm, ev_sz, al_sz;
  char ph[64], ev[64], al[64], pr[64], fo[64], rm[128], fs[5];

/***** edited by K.Abe '89 Oct. 17 ********************************************/
  int stt, end;
/***** edited by K.Abe '89 Oct. 17 ********************************************/

  for(;;)
  {
    if( fscanf( fp, "%s %s %d %d %d", fs, rm, &wd, &st, &en ) == EOF )
    {
      return(i);
    }
    if( fs[0] != '#' )
    {
      fprintf(stderr,"[Error in GetUnitAtteibute] File Bad format.\n");
      return(-1);
    }
    u[i].roman = (char*)malloc(strlen(rm)+1);
    strcpy(u[i].roman, rm );
    u[i].word = wd;
    u[i].stt = st;
    u[i].end = en;

/***** edited by K.Abe '89 Oct. 17 ********************************************/
    if( fscanf(fp, "%s %d %s %d %s %d %d %s %d",
        fs, &nm, rm, &st, pr, &en, &stt, fo, &end ) == EOF || fs[0] != '*' )
/***** edited by K.Abe '89 Oct. 17 ********************************************/
/*    if( fscanf(fp, "%s %d %s %s %s", fs, &nm, rm, pr, fo ) == EOF             */
/*        || fs[0] != '*' )                                                     */
/***** edited by K.Abe '89 Oct. 17 ********************************************/

    {
      fprintf(stderr,"[Error in GetUnitAtteibute] File Bad format.\n");
```

```
      return(-1);
    }
    u[i].wd_roman = (char*)malloc(strlen(rm)+1);
    strcpy( u[i].wd_roman, rm );

/***** edited by K.Abe '89 Oct. 17 ********************************************/
    u[i].pre = (NeighborPhone*)malloc(sizeof(NeighborPhone));
    u[i].pre->stt = st;
    u[i].pre->phn = (char*)malloc(strlen(pr)+1);
    strcpy( u[i].pre->phn, pr );
    u[i].pre->end = en;
    u[i].fol = (NeighborPhone*)malloc(sizeof(NeighborPhone));
    u[i].fol->stt = stt;
    u[i].fol->phn = (char*)malloc(strlen(fo)+1);
    strcpy( u[i].fol->phn, fo );
    u[i].fol->end = end;
/***** edited by K.Abe '89 Oct. 17 ********************************************/
/*    u[i].pre = (char*)malloc(strlen(pr)+1);                                  *
/*    strcpy( u[i].pre, pr );                                                  *
/*    u[i].fol = (char*)malloc(strlen(fo)+1);                                  *
/*    strcpy( u[i].fol, fo );                                                  *
/***** edited by K.Abe '89 Oct. 17 ********************************************/

    u[i].phn_size = nm;

    for( j = 0; j < u[i].phn_size; j++ )
    {
      if( fscanf(fp, "%s %d %s %d %d %d", fs, &st, ph, &en, &ev_sz, &al_sz )
         == EOF
         || fs[0] != 'p' )
      {
        fprintf(stderr,"[Error in GetUnitAtteibute] File Bad format.\n");
        return(-1);
      }
      u[i].phn[j] = (Phone*)malloc(sizeof(Phone));
      u[i].phn[j]->stt = st;
      u[i].phn[j]->phn = (char*)malloc(strlen(ph)+1);
      strcpy( u[i].phn[j]->phn, ph );
      u[i].phn[j]->end = en;
      u[i].phn[j]->eve_size = ev_sz;
      u[i].phn[j]->alp_size = al_sz;

      for( k = 0; k < u[i].phn[j]->eve_size; k++ )
      {
        if( fscanf(fp, "%s %d %s %d", fs, &st, ev, &en ) == EOF
           || fs[0] != 'e' )
        {
          fprintf(stderr,"[Error in GetUnitAtteibute] File Bad format.\n");
          return(-1);
        }
        u[i].phn[j]->eve[k] = (Event*)malloc(sizeof(Event));
        u[i].phn[j]->eve[k]->stt = st;
        u[i].phn[j]->eve[k]->eve = (char*)malloc(strlen(ev)+1);
        strcpy( u[i].phn[j]->eve[k]->eve, ev );
        u[i].phn[j]->eve[k]->end = en;
      }

      for( k = 0; k < u[i].phn[j]->alp_size; k++ )
      {
        if( fscanf(fp, "%s %d %s %d", fs, &st, al, &en ) == EOF
           || fs[0] != 'a' )
        {
          fprintf(stderr,"[Error in GetUnitAtteibute] File Bad format.\n");
          return(-1);
        }
```

```
        getc(fp);

        u[i].phn[j]->alp[k] = (Alophone*)malloc(sizeof(Alophone));
        u[i].phn[j]->alp[k]->stt = st;
        u[i].phn[j]->alp[k]->alp = (char*)malloc(strlen(al)+1);
        strcpy( u[i].phn[j]->alp[k]->alp, al );
        u[i].phn[j]->alp[k]->end = en;
      }
    }
    i++;
  }
}
#include "/usr4/takeda/SYN/include/Synthesis.h"
PutPhoneme( p, tab, no )
FILE *p;
Phoneme tab;
int no;
{
  register int i;

  fprintf(p,"# %4d %3d %3d %4d\n", no, tab.length, tab.inf,
                                   tab.unit_no );
  fprintf(p,"p %4d %s %4d %2d %2d\n", tab.phn->stt,
                                tab.phn->phn,
                                tab.phn->end,
                                (tab.phn==NULL?0:tab.phn->eve_size),
                                (tab.phn==NULL?0:tab.phn->alp_size)
        );
  for( i = 0; i < (tab.phn==NULL?0:tab.phn->eve_size); i++ )
  {
    fprintf(p,"e %4d %s %4d\n", tab.phn->eve[i]->stt,
                                tab.phn->eve[i]->eve,
                                tab.phn->eve[i]->end );
  }
  for( i = 0; i < (tab.phn==NULL?0:tab.phn->alp_size); i++ )
  {
    fprintf(p,"a %4d %s %4d\n", tab.phn->alp[i]->stt,
                                tab.phn->alp[i]->alp,
                                tab.phn->alp[i]->end );
  }
}
#include "/SYN/include/Synthesis.h"
PutUnitAttribute(fp, u)
FILE *fp;
UnitAttribute *u;
{
  register int i, j;

  fprintf(fp, "# %s %d %d %d\n", u->roman, u->word, u->stt, u->end );

/***** edited by K.Abe '89 Oct. 17 *********************************************/
  fprintf(fp,"* %d %s %d %s %d %d %s %d \n", u->phn_size, u->wd_roman,
                                            u->pre->stt, u->pre->phn, u->pre->end,
                                            u->fol->stt, u->fol->phn, u->fol->end );
/***** edited by K.Abe '89 Oct. 17 *********************************************/
/*  fprintf(fp,"* %d %s %s %s \n", u->phn_size, u->wd_roman, u->pre, u->fol );
/***** edited by K.Abe '89 Oct. 17 *********************************************/

  for( i = 0; i < u->phn_size; i++ )
  {
    fprintf(fp, "p %d %s %d %d %d\n", u->phn[i]->stt, u->phn[i]->phn,
                                      u->phn[i]->end, u->phn[i]->eve_size,
                                      u->phn[i]->alp_size );
    for( j = 0; j < u->phn[i]->eve_size; j++ )
    {
```

```
      fprintf(fp, "e %d %s %d\n", u->phn[i]->eve[j]->stt,
                                  u->phn[i]->eve[j]->eve,
                                  u->phn[i]->eve[j]->end );
    }
    for( j = 0; j < u->phn[i]->alp_size; j++ )
    {
      fprintf(fp, "a %d %s %d\n", u->phn[i]->alp[j]->stt,
                                  u->phn[i]->alp[j]->alp,
                                  u->phn[i]->alp[j]->end );
    }
  }
}


#include "/usr4/takeda/SYN/include/Synthesis.h"
ReadDuration( header, dur )
char *header;
DurationData dur[];
{
  FILE *fp, *fopen();
  char file[256];
  int size;
  sprintf( file, "%s.DR", header );
  if(( fp = fopen( file, "r" )) == NULL )
  {
    fprintf(stderr,"[Error in ReadDuration] File: %s can't open.\n", file );
    return(-1);
  }
  size = GetDuration(fp, dur);
  fclose(fp);
  return(size );
}
#include "/usr4/takeda/SYN/include/Synthesis.h"
ReadFrameOutLine( header, fr )
char *header;
FrameOutLine fr[];
{
  FILE *fp, *fopen();
  char file[256];
  int size;

  sprintf( file, "%s.FR", header );
  if(( fp = fopen( file, "r" )) == NULL )
  {
    fprintf(stderr,"[Error in FrameOutLine] File: %s can't open.\n", file );
    return(-1);
  }
  size = GetFrameOutLine(fp,fr);
  fclose(fp);
  return(size);
}
#include "/SYN/include/Synthesis.h"
#define E stderr
ReadLabel( word, phn )
int word;
Phone phn[];
{
  FILE *fp, *fopen();
  int size;
  char file_name[256];

  if( word < 0 || word > 5240 )
  {
    fprintf( E, "ReadLabel: Invalid File No (%d)\n", word );
    return(-1);
```

```
  }

  sprintf( file_name, "/MHT/LBL/D%d/MHT_1_%04d.LB",
                      word/1000, word );

  if(( fp = fopen( file_name, "r" )) == NULL )
  {
    fprintf( E, "ReadLabel: File open error (%s) \n", file_name );
    return(-1);
  }

  if(( size = GetLabel( fp, phn )) < 0 )
  {
    fprintf( E, "ReadLabel: Get Label error \n" );
    return(size);
  }

  return(size);
}
#include "/usr4/takeda/SYN/include/Synthesis.h"
ReadOriginalUnitAttribute( header, ua )
char *header;
UnitAttribute ua[];
{
  FILE *fp, *fopen();
  char file[256];
  int size;

  sprintf( file, "%s.oua", header );
  if(( fp = fopen( file, "r" )) == NULL )
  {
    fprintf( stderr,
             "[Error in ReadOriginalUnitAttribute] File %s can't open.\n",
             file );
    return(-1);
  }
  size = GetUnitAttribute(fp,ua);
  fclose(fp);
  return(size);
}
#include "/usr4/takeda/SYN/include/Synthesis.h"
ReadPhoneme( header, unit )
char *header;
Phoneme unit[];
{
  FILE *fp, *fopen();
  char file[256];
  int size;

  sprintf( file, "%s.unit", header );
  if(( fp = fopen( file, "r" )) == NULL )
  {
    fprintf(stderr,"[Error in ReadPhoneme] File: %s can't open.\n", file );
    return(-1);
  }
  size = GetPhoneme( fp, unit );
  fclose(fp);
  return(size);
}



#include "/usr4/takeda/SYN/include/Synthesis.h"
ReadPhonemeOutLine(header, ph)
char *header;
```

```
PhonemeOutLine ph[];
{
  FILE *fp, *fopen();
  char file[256];
  int size;

  sprintf( file, "%s.PN", header );
  if(( fp = fopen( file, "r" )) == NULL )
  {
    fprintf(stderr,"[Error in ReadPhonemeOutLine] File: %s can't open.\n",
            file );
    return(-1);
  }
  size = GetPhonemeOutLine(fp, ph);
  fclose(fp);
  return(size);
}


#include "/usr4/takeda/SYN/include/Synthesis.h"
ReadPhrase(header,phrase)
char *header;
Phrase phrase[];
{
  FILE *fp, *fopen();
  int no = 0;
  char file[256];

  sprintf( file, "%s.PH", header );
  if(( fp = fopen( file, "r" )) == NULL )
  {
    fprintf(stderr,"[Error in ReadPhrase]; File: %s not exist.\n", file );
    return(-1);
  }
  no = GetPhrase(fp,phrase);
  fclose(fp);
  return(no);
}
#include "/usr4/takeda/SYN/include/Synthesis.h"
ReadUnitAttribute( header, ua )
char *header;
UnitAttribute ua[];
{
  FILE *fp, *fopen();
  char file[256];
  int size;

  sprintf( file, "%s.ua", header );
  if(( fp = fopen( file, "r" )) == NULL )
  {
    fprintf(stderr,"[Error in ReadUnitAttribute] File %s can't open.\n",
                   file );
    return(-1);
  }
  size = GetUnitAttribute(fp,ua);
  fclose(fp);
  return(size);
}
#include "/usr4/takeda/SYN/include/Synthesis.h"
WritePhoneme( header, size, tab )
char *header;
int size;
Phoneme tab[];
{
  FILE *fp, *fopen();
```

```
  char file[256];
  register int i;

  sprintf( file, "%s.unit", header );
  if(( fp = fopen( file, "w" )) == NULL )
  {
    fprintf(stderr,"[Error in WritePhoneme] File %s can't open.\n", file );
    return(-1);
  }
  for( i = 0; i < size; i++ )
  {
    PutPhoneme( fp, tab[i], i );
  }
  fclose(fp);
  return(1);
}
#include <stdio.h>
#include "/usr4/takeda/SYN/include.h"
write_unit( p, size, tab )
FILE *p;
int size;
Unit tab[];
{
  register int i, j;
  for( i = 0; i < size; i++ )
  {
    fprintf(p,"# %4d %3d %3d %4d\n", i, tab[i].length, tab[i].inf,
                                     tab[i].unit_no );
    fprintf(p,"p %4d %s %4d %2d %2d\n", tab[i].phn->stt,
                                tab[i].phn->phn,
                                tab[i].phn->end,
                                (tab[i].phn==NULL?0:tab[i].phn->eve_size),
                                (tab[i].phn==NULL?0:tab[i].phn->alp_size)
          );
    for( j = 0; j < (tab[i].phn==NULL?0:tab[i].phn->eve_size); j++ )
    {
      fprintf(p,"e %4d %s %4d\n", tab[i].phn->eve[j]->stt,
                                  tab[i].phn->eve[j]->eve,
                                  tab[i].phn->eve[j]->end );
    }
    for( j = 0; j < (tab[i].phn==NULL?0:tab[i].phn->alp_size); j++ )
    {
      fprintf(p,"a %4d %s %4d\n", tab[i].phn->alp[j]->stt,
                                  tab[i].phn->alp[j]->alp,
                                  tab[i].phn->alp[j]->end );
    }
  }
}
#include "/usr4/takeda/SYN/include/Synthesis.h"
WriteUnitAttribute( header, size, u )
char *header;
int size;
UnitAttribute u[];
{
  FILE *fp, *fopen();
  char file[256];
  register int i;

  sprintf( file, "%s.ua", header );
  if(( fp = fopen( file, "w" )) == NULL )
  {
    fprintf(stderr,"[Error in WriteUnitAttribute] File: %s can't create.\n",
                   file );
  }
  for( i = 0; i < size; i++ )
```

```
  {
    PutUnitAttribute(fp,u[i]);
  }
  fclose(fp);
}
#include "/usr4/takeda/SYN/include/Synthesis.h"
char *Unit_end_phoneme(u)
UnitAttribute u;
{
  if( ! strcmp( u.roman, "pau" ))
       return( "nil" );
  return(u.phn[u.phn_size-1]->phn);
}
char *Unit_start_phoneme(u)
UnitAttribute u;
{
  if( ! strcmp( u.roman, "pau" ))
       return("nil");
  return(u.phn[0]->phn);
}
int Unit_end_frame(u)
UnitAttribute u;
{
  return(u.phn[u.phn_size-1]->end);
}
int Unit_start_frame(u)
UnitAttribute u;
{
  return(u.phn[0]->stt);
}
Unit_length(u)
UnitAttribute u;
{
  return( Unit_end_frame(u)-Unit_start_frame(u));
}
```

Appendix3

単位選択部

unit-select.art
run-us.lisp
definitions.lisp
tables.lisp
string-letter.lisp
criterion.lisp
break-string.lisp
pre-selection.lisp
label-functions.lisp
costing;functions.lisp
dictionary-manipulation.lisp
new-get-lattice.lisp
art-tools.lisp
art-functions.lisp
call-vax.lisp
communicate.lisp

```
;;; -*- mode: lisp; Package: ART-USER; base: 10  -*-
;;
;;


;;
;; Convert Package for ART
;;
(defun convert-string-to-phonetic-property-list (string)
  (zl-user::convert-string-to-phonetic-property-list string))

(defun get-lattice-new (string)
  (zl-user::get-lattice-new string))


;;
;; Mesellious
;;

(defun sort-sequence-candidates (list)
  "This function sorts seuqnec candidates
   according to the costs."
  (sort list '(lambda (x y) (< (car x)
                               (car y)))))

(defun load-sequence (sequence)
  "This function load elemental information
   about sequence."
  (let* ((sequen (cadr sequence)))
    (list (car sequence)
          (loop for seq in sequen
                collect
                  (list (zl-user::unit-flavor-unit-string seq)
                        (zl-user::unit-flavor-unit-stt seq)
                        (zl-user::unit-flavor-unit-end seq)
                        (zl-user::unit-flavor-start-criterion seq)
                        (zl-user::unit-flavor-end-criterion seq))))))

;;
;; For the rough search
;;
(defun swap-long-and-short (string switch)
  "This funciton convert long vowels to short ones,
   if they are (or it is) located at initial or final of
   a speech segment."
  (let* ((ph-st (zl-user::string-to-phoneme-list string))
         (init-st (car ph-st))
         (init-st-swp (swap-case init-st))
         (final-st (car (reverse ph-st)))
         (final-st-swp (swap-case final-st))
         (body-st-list (cdr (reverse (cdr ph-st))))
         (body-st (loop with res
                        for st in body-st-list
                        do
                    (setf res (cl:concatenate 'string st res))
                        finally
                          (return res))))
    (cl:concatenate 'string
                    (if (or (= switch 1) (= switch 3))
                        init-st-swp
                        init-st)
                    body-st
                    (if (or (= switch 2) (= switch 3))
                        final-st-swp
                        final-st))))
;    (cond ((= switch 1)
;          (cl:concatenate 'string
;                          init-st-swp
;                          body-st
;                          final-st))
;;                  (cons ''string
```



```
;;                          (cons init-st
;;                                (cdr ph-st))))))
;         ((= switch 2)
;          (cl:concatenate 'string
;                          init-st
;                          body-st
;                          final-st-swp))
;                          (reverse (cdr (reverse ph-st)))
;                          (list final-st))))))
;         ((= switch 3)
;          (eval
;            (cons 'cl:concatenate
;                  (cons ''string
;                        (cl:concatenate
;                          'list
;                          (reverse (cdr (reverse ph-st)))
;                          (list final-st))))))
;         )))


(defun swap-case (string)
  "This function convert the string to the
   string which consists of lower case chars.
   CAUTION !! This function assumes input as
   a string of ONE character, such as 'A and 'z"
  (cond ((string= string "A") "a")
        ((string= string "I") "i")
        ((string= string "U") "u")
        ((string= string "E") "e")
        ((string= string "O") "o")
        ((string= string "a") "A")
        ((string= string "i") "I")
        ((string= string "u") "U")
        ((string= string "e") "E")
        ((string= string "o") "O")
        (T string)))

(defun examine-dictionary (unit-prop-list)
  "This function examines dictionary if the speech segment
   is in the dictioanry or not."
  (let* ((null-context (zl-user::inquire-phonetic-properties "#"))
         (string (car unit-prop-list))
         (stt (zl-user::intp (second unit-prop-list)))
         (end (zl-user::intp (third unit-prop-list)))
         (sswitch (cond ((and stt end) -1)
                        ((and (not stt) end) -2)
                        ((and stt (not end)) -3)
                        (T -4))))
    (format t "~% Searching ~a with condition ~a."
            string sswitch)

    (or (cl-user::search-entry-new string
                                   null-context
                                   null-context
                                   sswitch)
        (if (not stt)
            (cl-user::search-entry-new (swap-long-and-short string 1)
                                       null-context
                                       null-context
                                       sswitch))
        (if (not end)
            (cl-user::search-entry-new (swap-long-and-short string 2)
                                       null-context
                                       null-context
                                       sswitch))
        (if (and (not end) (not stt))
            (cl-user::search-entry-new (swap-long-and-short string 3)
                                       null-context
                                       null-context
                                       sswitch))
```

```
143          )))
144
145
146   (defun search-dictionary (unit-prop-list pp-list
147                                           left-context
148                                           right-context)
149     "This function searches the dictionary and
150      returns the condidate speech segments."
151     (let* ((string (car unit-prop-list))
152            (null-context
153              (zl-user::inquire-phonetic-properties "#"))
154            (stt (second unit-prop-list))
155            (end (third unit-prop-list))
156            (sttp (zl-user::intp stt))
157            (endp (zl-user::intp end))
158            (left-context
159              ; To examine intra-phrase context
160              (cond ((zerop (floor stt))
161                     (or left-context
162                         null-context))
163                    (T (nth (1- (floor stt)) pp-list))))
164   ;          (if (zerop (floor stt))
165   ;              null-context
166   ;              (nth (1- (floor stt)) pp-list)))
167            (right-context
168              (or (nth (ceiling end) pp-list)
169                  right-context
170                  null-context))
171            ; To examine intra-phrase context
172   ;          (or (nth (ceiling end) pp-list)
173   ;              null-context))
174            (sswitch (cond ((and sttp endp) 1)
175                           ((and (not sttp) endp) 2)
176                           ((and sttp (not endp)) 3)
177                           (T 4))))
178
179       ; THis is for debug
180       (format t "Search string = ~a, left context = (~a) right context (~a) ~%"   ;
181                 string
182                 (zl-user::phonetic-property-flavor-symbol left-context)
183                 (zl-user::phonetic-property-flavor-symbol right-context))
184
185       (sort
186             (or (cl-user::search-entry-new string
187                                            left-context
188                                            right-context
189                                            sswitch)
190                 (if (not sttp)
191                     (cl-user::search-entry-new (swap-long-and-short string 1)
192                                                left-context
193                                                right-context
194                                                sswitch))
195                 (if (not endp)
196                     (cl-user::search-entry-new (swap-long-and-short string 2)
197                                                left-context
198                                                right-context
199                                                sswitch))
200                 (if (and (not sttp) (not endp))
201                     (cl-user::search-entry-new (swap-long-and-short string 2)
202                                                left-context
203                                                right-context
204                                                sswitch)))
205             '(lambda (x y) (< (seventh x)
206                               (seventh y))))))
207
208
209   ;;
210   ;;
211   ;;
212   (defun pitch-cost (target sub entry)
213     (zl-user::pitch-cost target sub entry))
```



```
214
215   (defun get-total-cost (cost-list)
216     "This function calculates total cost."
217     (let* ((context (car cost-list))
218            (pitch (cadr cost-list))
219            (ext (caddr cost-list)))
220       (+ (* context 100)
221          pitch
222          ext)))
223
224   ;;
225   (defun time ()
226     (zl-user::time))
227
228
229   (defun test (x y)
230     (+ x y))
231   ;;;;;;;;;;;;
232
233   (defun get-word (no)
234     (if (zerop no)
235         "pau"
236         (zl-user::get-word no)))
237
238
239   (defun convert-to-ph-count-stt (stt stt-word)
240         (1+ (ceiling stt-word)))
241
242   ;  (let* ((wordp (zl-user::intp stt-word))
243   ;         (sttp (zl-user::intp stt)))
244   ;    (cond ((and wordp sttp)
245   ;           (1+ (floor stt-word)))
246   ;          ((and wordp (not sttp))
247   ;           (+ (floor stt-word) 2))
248   ;          (T (1+ (ceiling stt-word))))))
249
250   (defun convert-to-ph-count-end (end end-word)
251         (if (zl-user::intp end)
252             (floor end-word)
253             (ceiling end-word)))
254
255   (defun get-the-string (word stt end)
256     (if (zerop word)
257         "pau"
258         (let* ((label (zl-user::get-label word))
259                (ph-trans (mapcar 'car (cdr label)))
260                (ph (mapcar 'car ph-trans))
261                (p (subseq ph (max (1- stt) 0) end))
262                (com1 (cons ''string p))
263                (com2 (cons 'cl:concatenate com1)))
264           (eval com2))))
265
266   ;;;
267   ;;;  For the rule retry-with-smaller-units
268   ;;;
269
270
271   ;;
272   ;;
273   ;;   These functions are for progs before version 2.0
274   ;;
275   ;;(defun primitivep (unit-inf)
276   ;;  "This func. examine if the string
277   ;;   can be devided by the current break-points."
278   ;;  (let* ((string (car unit-inf))
279   ;;         (stt-p (second unit-inf))
280   ;;         (end-p (third unit-inf))
281   ;;         (c-b (list (- stt-p (floor stt-p))
282   ;;                    (- end-p (floor stt-p))
283   ;;                    (if (zl-user::intp stt-p) (+ 0.5 stt-p))
284   ;;                    (if (zl-user::intp end-p) (- 0.5 end-p)))))
```

```
;;    (not
;;      (loop for cr in zl-user::*nuss-criterion-priority*
;;          for ac = (zl-user::activate-criterion cr)
;;          for brks = (scl:send ac :search-break-point string)
;;          thereis  (set-difference brks c-b)))))

;;(defun make-smaller-unit (sequence-inf index score string-pp)
;; "This function redevide specified unit."
;;  (let* ((the-unit (nth index sequence-inf))
;;        (string (car the-unit))
;;        (stt (second the-unit))
;;        (off-set (floor stt))
;;        (end (third the-unit))
;;        (stt-cr (fourth the-unit))
;;        (end-cr (fifth the-unit))
;;        (stt-brkp (zl:make-instance
;;                    'zl-user::break-point-flavor
;;                    :location stt
;;                    :criterion stt-cr))
;;        (end-brkp (zl::make-instance
;;                    'zl-user::break-point-flavor
;;                    :location end
;;                    :criterion end-cr))
;;        (brkp (car (zl-user::get-break-point
;;                     string '("voiceless fricative criterion"
;;                              "voiced stop criterion"
;;                              "voiced fricative criterion"
;;                              "VC boundary criterion")))))
;;    (if brkp
;;      (let* ((brkp-new (zl:make-instance
;;                         'zl-user::break-point-flavor
;;                         :location (+ off-set (zl-user::break-point-flavor-location
 brkp))
;;                         :criterion (zl-user::break-point-flavor-criterion brkp)
;;                         :score (zl-user::break-point-flavor-score brkp)))
;;             (new-score (+ score (zl-user::break-point-flavor-score brkp)))
;;             (new1 (make-unit-flavor stt-brkp brkp-new string-pp))
;;             (new2 (make-unit-flavor brkp-new end-brkp string-pp)))
;;        (loop with res
;;              for unit in sequence-inf
;;              for i from 0
;;              do
;;          (if (= i index)
;;              (progn
;;                (setq res (cons new1 res))
;;                (setq res (cons new2 res)))
;;              (setq res (cons unit res)))
;;              finally (return (list new-score (reverse res)))))
;;      nil)))


(defun make-unit-flavor (stt end pp-string)
  "This function construct unit flavor from
   the given stt and end break point flavor"
  (let* ((stt-in-int (floor (zl-user::break-point-flavor-location stt)))
         (end-in-int (ceiling (zl-user::break-point-flavor-location end))))
  (list (zl-user::inquire-string (subseq pp-string stt-in-int end-in-int))
        (zl-user::break-point-flavor-location stt)
        (zl-user::break-point-flavor-location end)
        (zl-user::break-point-flavor-criterion stt)
        (zl-user::break-point-flavor-criterion end))))

(defun first-nth (n list)
  "This function returns the first nth sub-list
   of the given list"
  (loop with result
        for x in list
        for i from 1 to n
        do
    (setf result (reverse (cons x (reverse result))))
```



```
    finally (return result)))



;;
;;  For the v2
;;

(defun get-new-unit (no unit-list)
  "This function devide unit string more precisely."
  (let* ((string (car unit-list))
         (off-set (floor (second unit-list)))
         (new-cr (nth no zl-user::*nuss-criterion-priority*))
         (new-units (zl-user::get-lattice-new string new-cr)))
    (zl:loop for unit-prop in new-units
             for score = (car unit-prop)
             for unit = (second unit-prop)
      collect
       (list score
             (zl:loop for un in unit
                collect
                  (list (zl-user::unit-flavor-unit-string un)
                        (+ (zl-user::unit-flavor-unit-stt un) off-set)
                        (+ (zl-user::unit-flavor-unit-end un) off-set)
                        (zl-user::unit-flavor-start-criterion un)
                        (zl-user::unit-flavor-end-criterion un)
                        (1+ no)))))))

;; Nov
(defun get-criterion-no (unit)
  (let* ((stt-cr (fourth unit))
         (end-cr (fifth unit))
         (prt-list zl-user::*nuss-criterion-priority*))
    (or
      (loop for index from (length prt-list) by -1
            for cr in (reverse prt-list)
            do
        (if (or (string= cr stt-cr)
                (string= cr end-cr))
            (return index)))
      0)))



;;;
;;;
;;;
(defun get-pp-from-str (str)
  (if (not str)
      nil
      (zl-user::convert-string-to-phonetic-property-list str)))
;;
;;
(defun make-smaller-unit (sequence-inf index score string-pp criterion-no)
  "This function redevide specified unit."
  (let* ((the-unit (nth index sequence-inf))
         (string (car the-unit))
         (stt (second the-unit))
         (off-set (floor stt))
         (end (third the-unit))
         (stt-cr (fourth the-unit))
         (end-cr (fifth the-unit))
         (stt-brkp (zl:make-instance
                     'zl-user::break-point-flavor
                     :location stt
                     :criterion stt-cr))
         (end-brkp (zl::make-instance
                     'zl-user::break-point-flavor
                     :location end
                     :criterion end-cr))
         (brkp (center (zl-user::get-break-point
```

```
426                          string (list (nth (- criterion-no 1)
427                                            zl-user::*nuss-criterion-priority*))))))
428        (if brkp
429            (let* ((brkp-new (zl:make-instance
430                              'zl-user::break-point-flavor
431                              :location (+ off-set (zl-user::break-point-flavor-location
    brkp))
432                              :criterion (zl-user::break-point-flavor-criterion brkp)
433                              :score (zl-user::break-point-flavor-score brkp)))
434                   (new-score (+ score (zl-user::break-point-flavor-score brkp)))
435                   (new1 (make-unit-flavor stt-brkp brkp-new string-pp))
436                   (new2 (make-unit-flavor brkp-new end-brkp string-pp)))
437              (loop with res
438                    for unit in sequence-inf
439                    for i from 0
440                    do
441                (if (= i index)
442                    (progn
443                      (setq res (cons new1 res))
444                      (setq res (cons new2 res)))
445                    (setq res (cons unit res)))
446                    finally (return (list new-score (reverse res)))))
447            nil)))
448
449
450    (defun center (list)
451      (let* ((middle (// (float (length list)) 2.0)))
452        (nth (floor middle) list)))
453
454    (defvar *criterion-no* 1)
455    (defun count-up-criterion ()
456      "This rule count up criterion no."
457      (if (> *criterion-no*
458             (length zl-user::*nuss-criterion-priority*))
459          nil
460          (setf *criterion-no* (1+ *criterion-no*))))
461
462    (defun get-current-criterion ()
463      *criterion-no*)
464
465    (defun reset-criterion ()
466      (setf *criterion-no* 1))
467
468    (defun primitivep (cc unit-inf)
469      "This func. examine if the string
470       can be devided by the current break-points."
471      (let* ((string (car unit-inf))
472             (stt-p (second unit-inf))
473             (end-p (third unit-inf))
474             (c-b (list (- stt-p (floor stt-p))
475                        (- end-p (floor stt-p))
476                        (if (zl-user::intp stt-p) (+ 0.5 stt-p))
477                        (if (zl-user::intp end-p) (- 0.5 end-p)))))
478        (not
479          (loop for cr in (first-nth cc zl-user::*nuss-criterion-priority*)
480                for ac = (zl-user::activate-criterion cr)
481                for brks = (scl:send ac :search-break-point string)
482                thereis  (set-difference brks c-b)))))
483
484
```

```
1     ;;; -*- mode: lisp; package: zetalisp-user; base: 10 -*-
2     ;;;
3
4     ;; Text length counter used in the rule: how-long
5
6     (defun how-long (string)
7       "This function returns the length of the input text sequence"
8       (let* ((string-list (convert-string-to-phonetic-property-list string)))
9         (length string-list)))
10
11    ;;
12    (defun duplicate-vowel-nasal (string)
13      "This function is for the entry-less unit."
14      (let* ((pp-list (if (listp string)
15                          string
16                          (convert-string-to-phonetic-property-list string))))
17        (loop for index from 1
18              for pp in pp-list
19              if (or (phonetic-property-flavor-vowel? pp)
20                     (string= (phonetic-property-flavor-manner pp) "nasal"))
21                collect (list (inquire-string (sublist pp-list
22                                                       0 index))
23                              (inquire-string (sublist
24                                               pp-list
25                                               (max (1- index) 0) (length pp-list)))))))
```

```
;;; -*- mode: lisp; package: zetalisp-user; base: 10 -*-
;;;
;;; Main Programs
;;;

;;
;; Method: search-break-point
;;

(defmethod (:search-break-point break-point-criterion-flavor) (sequence)
  (let* ((phonetic-property-list (if (listp sequence)
                                     sequence
                                     (convert-string-to-phonetic-property-list sequence))))
    (loop for prev = (inquire-phonetic-properties "#")
                  then curr
          for curr in phonetic-property-list
          for position from 0
          if (funcall predicate prev curr)
            collect (if boundary-location
                        (- position 0.5)   ; Takeda Oct 13, 1989
                        position))))       ; To Adapt center concatenation

;;
;;
;;      May 10,  Function break string
;;

(defun get-substring-pairs (string criterion)
    "This function calls generic function search-break-point and
     return all binary divisions obtained by the rule"
    (let* ((break-positions (send criterion :search-break-point string))
           (phonemic-sequence)
           (original-string))
      (cond ((listp string)
             (setq phonemic-sequence (loop for pp in string
                                           for st = (phonetic-property-flavor-symbol pp)
                                           collect (delete-camma st)))
             (setq original-string (list-to-string phonemic-sequence)))
            ((stringp string)
             (setq phonemic-sequence (string-to-phoneme-list string))
             (setq original-string string))
            (t (format 't "[Error in get-substring pairs]~%")
               (format 't "Invalid data type of string.~%")
               (setq break-positions nil)))
      (if break-positions
          (loop for break-position in break-positions
                collect (divide-string phonemic-sequence break-position))
          original-string)))

(defun get-substrings (string criterion)
    "This function calls generic function search-break-point and
     return all substrings obtained by the rule"
    (let* ((break-positions (reverse (send criterion :search-break-point string)))
           (phonemic-sequence)
           (original-string))
      (cond ((listp string)
             (setq phonemic-sequence (loop for pp in string
                                           for st = (phonetic-property-flavor-symbol pp)
                                           collect (delete-camma st)))
             (setq original-string (list-to-string phonemic-sequence)))
            ((stringp string)
             (setq phonemic-sequence (string-to-phoneme-list string))
             (setq original-string string))
            (t (format 't "[Error in get-substring pairs]~%")
               (format 't "Invalid data type of string.~%")
               (setq break-positions nil)))
      (if break-positions
          (break-string phonemic-sequence break-positions)
```



```
          (list original-string))))
```

```
;;; -*- mode: lisp; package: zetalisp-user; base: 10 -*-
;;;
;;;
;;;


; [ Concatenation Cost ] xoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxo

;;
;; Concatenation
;;

(defvar *concatenation-compromising-cost* 10)
(defun concatenation-cost (sub-str-seq)
  "This function calcurates the penalty cost of the useage from the standpoint
   of unit concatenation, for each unit candidate."
  (let* ((stt-criterion (fourth sub-str-seq))
         (end-criterion (fifth sub-str-seq)))
    (loop with cr-list = (cl:concatenate 'list
                                         *nuss-criterion-priority*
                                         (list "VC boundary criterion"))
;                                (cons "vowel center criterion"
;                                      *nuss-criterion-priority*))
          for criterion in cr-list
          for index from 1
                        ; [Kin]
                        ; vowel center criterion's cost is most heavy
                        ;
          if (string= stt-criterion criterion) sum index into cost
          if (string= end-criterion criterion) sum index into cost
          finally
            (return cost))))

; [ Context Simulaity Cost ] xoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxo
x

;;
;; Context similarity
;;
;;
;; This function would be called at first
;;
; ----------------------------------
(defun context-similarity-cost (text-pp-list stt-location end-location word-no stt en
d)
  "This is new version dabe--.
   CAUTION !!!!
   The locating indexes are tricky !!
   The stt index indicates a location wich counting the sequence from 0.
   However, the end index was counted from 1 !!!!!!."
  ; [Takeda, july 18]
  ; A new version. Stt and End index of the speech portion are counted up
  ; normally, means start with 0.
  ; For example, the sub-string EFG is specified as stt = 5 end = 7 in the following
  ; example.
  ;   a b c d E F G h i
  ;   1 2 3 4 5 6 7 8 9
  ;
  ; I hate nth function !! Why don't you start with 1 !!!!
  ;
  (let* ((text-right-pp (nth end-location text-pp-list))
         (text-left-pp (if (= 1 stt-location)
                           nil
                           (nth (- stt-location 2) text-pp-list)))
         (label (loop for lb in (read-label (get-label word-no))
                      collect (send lb :convert-label-into-phonetic-property)))
         (entry-left-pp (if (= 1 stt)
                            nil
                            (nth (- stt 2) label)))
         (entry-right-pp (nth end label))
         (entry-stt-pp (nth (1- stt) label))
```



```
         (entry-end-pp (nth (1- end) label))
         (left-cost (left-context-cal entry-left-pp text-left-pp))
         (right-cost (right-context-cal entry-right-pp text-right-pp)))
    (if (or (string= (phonetic-property-flavor-manner entry-stt-pp) "semi-vowel")
            (phonetic-property-flavor-vowel? entry-stt-pp))
        (setq left-cost (* left-cost 2)))
    (if (or (string= (phonetic-property-flavor-manner entry-end-pp) "semi-vowel")
            (phonetic-property-flavor-vowel? entry-end-pp))
        (setq right-cost (* right-cost 2)))
    (+ right-cost left-cost)))

(defun left-context-cal (pp-a pp-b)
  "Please optimaize this function !!."
  (cond ((or (not pp-a) (not pp-b))
         ;
         ; Boundary property check
         ;
         (if (eq pp-a pp-b)
             0                                  ; both pp are nil
             *cont-sim-cost-edge-unmatch*))     ; either of pp is nil
        ;
        ; CV property check
        ;
        ((not (eq (phonetic-property-flavor-vowel? pp-a)
                  (phonetic-property-flavor-vowel? pp-b)))
         *cont-sim-cost-cv-unmatch*)
        ;
        ; both of pp are vowels
        ;
        ((and (phonetic-property-flavor-vowel? pp-a)
              (not (string= (phonetic-property-flavor-symbol pp-a)
                            (phonetic-property-flavor-symbol pp-b))))
         *cont-sim-cost-place-unmatch*)
        ;
        ; both of pp are consonats
        ;
        (t (let ((cost 0))
             ;
             ; UV property check
             ;
             (if (not (eq (phonetic-property-flavor-voiced? pp-a)
                          (phonetic-property-flavor-voiced? pp-b)))
                 (incf cost *cont-sim-cost-uv-unmatch*))
             ;
             ; Place of articulation check
             ;
             (if (not (string= (phonetic-property-flavor-place pp-a)
                               (phonetic-property-flavor-place pp-b)))
                 (incf cost *cont-sim-cost-place-unmatch*))
             ;
             ; Manner of articulation check
             ;
             (if (not (string= (phonetic-property-flavor-manner pp-a)
                               (phonetic-property-flavor-manner pp-b)))
                 (incf cost *cont-sim-cost-manner-unmatch*))
             cost))))


(defun right-context-cal (pp-a pp-b)
  "Please change this function as you like !!."
  (cond ((or (not pp-a) (not pp-b))
         ;
         ; either or both of pp is nil, which means boundary
         ;
         (if (eq pp-a pp-b)
             0                                  ; both pp are nil
             *cont-sim-cost-edge-unmatch*))     ; either of pp is nil
        (t (let ((cost 0))
             ;
             ; VC property check
             ;
```

```
141                 (if (not (eq (phonetic-property-flavor-vowel? pp-a)
142                              (phonetic-property-flavor-vowel? pp-b)))
143                     (incf cost *cont-sim-cost-cv-unmatch*))
144                 ;
145                 ; UV property check
146                 ;
147                 (if (not (eq (phonetic-property-flavor-voiced? pp-a)
148                              (phonetic-property-flavor-voiced? pp-b)))
149                     (incf cost *cont-sim-cost-uv-unmatch*))
150                 ;
151                 ; Place of articulation check
152                 ;
153                 (if (not (string= (phonetic-property-flavor-place pp-a)
154                                   (phonetic-property-flavor-place pp-b)))
155                     (incf cost *cont-sim-cost-place-unmatch*))
156                 ;
157                 ; Manner of articulation check
158                 ;
159                 (if (not (string= (phonetic-property-flavor-manner pp-a)
160                                   (phonetic-property-flavor-manner pp-b)))
161                     (incf cost *cont-sim-cost-manner-unmatch*))
162                 cost))))
163
164   (defvar *cont-sim-cost-edge-unmatch* 32)
165   (defvar *cont-sim-cost-cv-unmatch* 16)
166   (defvar *cont-sim-cost-uv-unmatch* 8)
167   (defvar *cont-sim-cost-place-unmatch* 4)
168   (defvar *cont-sim-cost-manner-unmatch* 2)
169
170
171   ; [ Pitch Difference Cost ] xoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxo
172
173   ;;
174   ;; Function Pitch-cost
175   ;;
176
177   (defun pitch-cost (target-pitch sub-string entry)
178     "This function calcurates the cost of pitch similarity."
179     (let* ((word-no (fourth entry))
180            (stt (second entry))
181            (end (third entry))
182            (word-stt (floor stt))
183            (word-end (ceiling end))
184            (label (get-label word-no))
185            (stt-in-time (car (label-phonetic-segmentation label (- word-stt 1))))
186            (end-in-time (second (label-phonetic-segmentation label word-end)))
187            (stt-in-frame (convert-time-to-frame stt-in-time))
188            (end-in-frame (convert-time-to-frame end-in-time))
189            (target-pitch-pattern (encode-pitch-pattern
190                                   (get-target-pitch-contour target-pitch sub-string)))
191            (original-pitch-pattern (encode-pitch-pattern
192                                     (cl-user::get-pitch word-no
193                                                         stt-in-frame
194                                                         end-in-frame))))
195       (loop for tp in target-pitch-pattern
196             for op in original-pitch-pattern
197             sum (diff tp op))))
198
199   (defun intp (f)
200     (zerop (- f (floor f))))
201
202   (defun encode-pitch-pattern (point-pitch)
203     "This function encodes the given pitch array into a three element vector."
204     (let* ((pitch-list (loop for p in point-pitch
205                              if (not (zerop p)) collect p))
206            (mean-p (if pitch-list
207                        (// (eval (cons '+ pitch-list)) (length pitch-list))
208                        nil))
209            (real-pitch-list (if pitch-list
210                                 (loop with range = (* mean-p 2)
211                                       for p in pitch-list
```



```
212                                       if (< p range) collect p)
213                                 '(0 0 0)))
214            (stt-pp (car real-pitch-list))
215            (end-pp (car (last real-pitch-list)))
216            (max-pp (eval (cons 'max real-pitch-list)))
217            (cen-pp (if (or (= stt-pp max-pp) (= max-pp end-pp))
218                        (nth (floor (length real-pitch-list) 2) real-pitch-list)
219                        max-pp)))
220       (list stt-pp cen-pp end-pp)))
221
222   ;
223   ; This function is also defined in sp:/usr4/takeda/MIT/isp/synthesis/lable-functions.
      lisp
224   ;(defun label-phonetic-segmentation (label phn-no)
225   ;  "This function returns the start and end time of the phonetic label period."
226   ;  (let* ((the-phn (nth (max 0 (- phn-no 1)) (cdr label)))
227   ;         (stt-time (second (car the-phn)))
228   ;         (end-time (third (car the-phn))))
229   ;    (list stt-time end-time)))
230
231   (defvar *nuss-analysis-period* 5)
232   (defun convert-time-to-frame (time)
233     "Convert absolute offset in time to in frame."
234     (floor time *nuss-analysis-period*))
235
236
237   (defun diff (a b)
238     (abs (- a b)))
239
240   (defun get-target-pitch-contour (target-pitch substring)
241     "This function get terget pitch contour of a speech portion.
242      substring is expected to be a list of (string stt end)"
243     (let* ((string (first substring))
244            (stt-mora (convert-phn-to-mora string (second substring)))
245            (end-mora (convert-phn-to-mora string (third substring))))
246       (sublist target-pitch (max (- stt-mora 1) 0)
247                             (min (+ end-mora 1) (length target-pitch)))))
248
249   (defun convert-phn-to-mora (string loc)
250     "This function converts the location unit to mora from phn."
251     (let* ((p-list (convert-string-to-phonetic-property-list string)))
252       (loop with count = 1
253             for pp in (sublist p-list 0 (- loc 1))
254             do
255         (cond ((phonetic-property-flavor-inseparable pp)
256                (incf count 2))
257               ((phonetic-property-flavor-vowel? pp)
258                (incf count)))
259             finally
260               (return count))))
261
262
263   ;; ------------------------------------------------------------------------
264   ;;;
265   ;;; Duration costing functions
266   ;;;
267   ;(defun pre-processing-for-duraiton-costing (targert-duration phrase-length text-pp)
268   ;  "This function pre-processes the target-duration list."
269   ;  (loop for target-inf in target-duration
270   ;        for target-phn = (car target-inf)
271   ;        for target-dur = (cadr target-inf)
272   ;        if (not (string= target-phn "pau"))
273   ;        do
274   ;    (
275   ;(defun duration-cost (target-duration substring entry)
276   ;  "This function costs the duration difference."
277   ;  (let* ((
```

```
;;; -*- mode: lisp; package: zetalisp-user; base: 10 -*-
;;;
;;; File-name Rule-Management.lisp [~/Lisp/synthesis]
;;;

;;
;; variables: *nuss-number-of-criterions*
;;            *nuss-criterion-table*
;;
(defvar *nuss-max-criterion-number* 30)
(defvar *nuss-number-of-criterions* 0) ; How many criterions are there
(defvar *nuss-criterion-table* (make-array *nuss-max-criterion-number*))
                                       ; Table of criterion


;;
;; Flavor of the criterion expression
;;
(defflavor break-point-criterion-flavor
        ((score)
         (boundary-location)
         (name "example")
         (description "Description about the criterion")
         (predicate '(lambda ())))
        ()
  (:writable-instance-variables)
  :initable-instance-variables)

;;
;; Function: init-nuss
;;
(defun init-criterion (&optional (criterion-size 30))
  "This function intializes the system parameters."
  (setq *nuss-number-of-criterions* 0)
  (setq *nuss-criterion-table* (make-array criterion-size :initial-value nil)))

;;
;; Function: show-criterion-status
;;
(defun show-criterion-status ()
  (format 't "~% ~d criterions are registrated as follows~%" *nuss-number-of-criterions*)
  (loop for criterion being the array-element of *nuss-criterion-table*
        for name = (first criterion)
        for i from 0 below *nuss-number-of-criterions*
        do
    (format 't "   ~a~%" name)))

;;

;; Function: activate-criterion
;;
(defun activate-criterion (criterion-name)
  "This function activate a criterion and make instance of the break-point-criterion-flavor"
  (loop for i from 0 below *nuss-number-of-criterions*
        for criterion-list being the array-element of *nuss-criterion-table*
        for name = (first criterion-list)
        do
    (if (string= criterion-name name)
        (return (make-instance 'break-point-criterion-flavor
                               :name name
                               :description (second criterion-list)
                               :predicate (third criterion-list)
                               :boundary-location (fourth criterion-list)
                               :score (fifth criterion-list))))
    finally
      (format 't "~% Error (active-criterion) The criterion ~a can't find~%" criterion-name)
      (return nil)))
;;
```



```
;; Macro: defcriterion
;;
(defmacro defcriterion (criterion-name descriptions predicate boundary-location score
)
  `(loop for criterion being the array-element of *nuss-criterion-table*
         for new-criterion = (list ,criterion-name
                                   ,descriptions
                                   ,predicate
                                   ,boundary-location
                                   ,score)
         for criterion-no from 0 below *nuss-number-of-criterions*
         do
     (if (string= ,criterion-name (first criterion))
        (progn
          (setf (aref *nuss-criterion-table* criterion-no) new-criterion)
          (return new-criterion)))
         finally
     (setf (aref *nuss-criterion-table* *nuss-number-of-criterions*) new-criterion)
     (incf *nuss-number-of-criterions*)
     (return new-criterion)))

;;
;; Criterions
;;
(defcriterion "voiceless stop criterion"
        "Break before voiceless stop"
  '(lambda (before after)
     (and
       (not (string= (phonetic-property-flavor-symbol before) "#"))
       (not (phonetic-property-flavor-voiced? after))
       (or (string= (phonetic-property-flavor-manner after) "stop")
           (string= (phonetic-property-flavor-manner after) "affricate"))))
  nil
  5)

(defcriterion "voiceless fricative criterion"
              "Break before voiceless fricative"
  '(lambda (before after)
     (and
       (not (string= (phonetic-property-flavor-symbol before) "#"))
       (not (phonetic-property-flavor-voiced? after))
       (string= (phonetic-property-flavor-manner after) "fricative")))
  nil
  15)

(defcriterion "voiced fricative criterion"
              "Break before voiced fricative"
  '(lambda (before after)
     (and
       (not (string= (phonetic-property-flavor-symbol before) "#"))
       (phonetic-property-flavor-voiced? after)
       (string= (phonetic-property-flavor-manner after) "fricative")))
  nil
  35)

(defcriterion "voiced stop criterion"
              "Break before voiced stop"
  '(lambda (before after)
     (and
       (not (string= (phonetic-property-flavor-symbol before) "#"))
       (phonetic-property-flavor-voiced? after)
       (string= (phonetic-property-flavor-manner after) "fricative")))
  nil
  40)


(defcriterion "VC boundary criterion"
              "Break at VC boundary, except for syllabic nasal and semi-vowel"
  '(lambda (before after)
     (and
       (not (string= (phonetic-property-flavor-symbol before) "#"))
```

```
139          (phonetic-property-flavor-vowel? before)
140          (not (equal (phonetic-property-flavor-manner after) "semi-vowel"))
141          (not (phonetic-property-flavor-vowel? after))))
142     nil
143     50)
144
145   (defcriterion "vowel center criterion"
146                 "Break at vowel center"
147     '(lambda (before after)
148        (and (phonetic-property-flavor-vowel? before)
149             ;
150             ; [KIN 7, Nev, 1989]
151             ; the vowel center criterion never applied for word, or phrase,
152             ; final vowel
153             ; To recover the original delete next list
154             (not (string= (phonetic-property-flavor-symbol after) "#"))
155             (or (not (phonetic-property-flavor-inseparable before))
156                 (or (string= (phonetic-property-flavor-inseparable before)
157                              "long vowel")
158                     (vowel-or-Np
159                       (cl:char (phonetic-property-flavor-top before) 0))))))
160     t
161     20)
162
163   ;;
164   ;; Criterions-predicate manipulation
165   ;;
166   (defun criterions-predicate-and (&rest criterions)
167     (loop for criterion in criterions
168           for predicate = (break-point-criterion-flavor-predicate criterion)
169           for predicate-list = (list 'funcall predicate 'befoer 'after)
170           collect  predicate-list into predicates
171           finally
172             (return (cons 'and predicates))))
173
174
175   (defun criterions-predicate-or (&rest criterions)
176     (loop for criterion in criterions
177           for predicate = (break-point-criterion-flavor-predicate criterion)
178           for predicate-list = (list 'funcall predicate 'befoer 'after)
179           collect  predicate-list into predicates
180           finally
181             (return (cons 'or predicates))))
182
183   ;;
184   ;; Complex criterions
185   ;;
186   ;
187   ;(defcriterion "voiceless stop or fricative"
188   ;             "break before voiceless stop or fricative"
189   ;    '(lambda (before after)
190   ;      (or  (and (not (phonetic-property-flavor-voiced? after))
191   ;           (or (string= (phonetic-property-flavor-manner after) "stop")
192   ;               (string= (phonetic-property-flavor-manner after) "affricate")))
193   ;           (and (not (phonetic-property-flavor-voiced? after))
194   ;                (string= (phonetic-property-flavor-manner after) "fricative"))))
195   ;    15)
196   ;
197   ;
198   ;(defcriterion "voiceless or voiced fricative"
199   ;             "break before voiceless stop or fricative or voiced fricative"
200   ;    '(lambda (before after)
201   ;      (or  (and (not (phonetic-property-flavor-voiced? after))
202   ;                (or (string= (phonetic-property-flavor-manner after) "stop")
203   ;                    (string= (phonetic-property-flavor-manner after) "affricate")))
204   ;           (and (not (phonetic-property-flavor-voiced? after))
205   ;                (string= (phonetic-property-flavor-manner after) "fricative"))
206   ;           (and (phonetic-property-flavor-voiced? after)
207   ;                (string= (phonetic-property-flavor-manner after) "fricative"))))
208   ;    15)
```

```
1     ;;; -*- mode: lisp; package: zetalisp-user; base: 10 -*-
2     ;;;
3     ;;; Definitions
4     ;;;
5     ;;;  Flavors
6     ;;;
7
8     (defflavor phonetic-property-flavor
9             ((symbol "?")
10             (vowel? NIL)
11             (voiced? NIL)
12             (manner "?")
13             (place "?")
14             (palatalized? NIL)
15             (gemination NIL)
16             (inseparable)
17             ; instance variable top is added for
18             ; sue dictionary generation
19             (top))
20          ()
21      :initable-instance-variables
22      (:writable-instance-variables))
23
24    (defflavor acoustic-property-flavor
25            ((duration)
26             (fundmental-freqency)
27             (power))
28          ()
29      :initable-instance-variables
30      (:writable-instance-variables))
31
32    (defflavor transcription-flavor
33            ((phonetic-transcription)  ; This is the fundational element
34             (event-transcription)     ; This should be a list !!
35             (alophone-transcription)
36             (insep-transcription)
37             (vowel-center-transcription))
38          ()
39      :initable-instance-variables
40      (:writable-instance-variables))
41
42    (defflavor acoustic-phonetic-property-flvor
43            ()
44          (phonetic-property-flavor
45           acoustic-property-flavor
46           transcription-flavor)
47      :initable-instance-variables
48      (:writable-instance-variables))
49
50    ;;
51    ;; Function for phonetic-property-flavor
52    ;;
53    (defmethod (pp-equal phonetic-property-flavor) (a)
54      "This function examines if the given two phonetic properties are the same."
55      (and (eq (phonetic-property-flavor-vowel? a)
56               (phonetic-property-flavor-vowel? self))
57           (eq (phonetic-property-flavor-voiced? a)
58               (phonetic-property-flavor-voiced? self))
59           (string= (phonetic-property-flavor-manner a)
60                    (phonetic-property-flavor-manner self))
61           (string= (phonetic-property-flavor-place a)
62                    (phonetic-property-flavor-place self))
63           (eq (phonetic-property-flavor-palatalized? a)
64               (phonetic-property-flavor-palatalized? self))
65           (eq (phonetic-property-flavor-gemination a)
66               (phonetic-property-flavor-gemination self))
67           (string= (phonetic-property-flavor-inseparable a)
68                    (phonetic-property-flavor-inseparable self))))
```

```
;;; -*- mode: lisp; Package: zetalisp-user; base: 10  -*-

(defvar *nuss-dictionary-size* 5240)

(defun load-label-file (file-name)
  "THis function read label information file
   and expand it onto a list."
  (with-open-file
    (stream file-name)
    (tv:noting-progress ("Generating label dictionary")
      (loop with table = (make-array  *nuss-dictionary-size*)
            for index from 0 below *nuss-dictionary-size*
            for word = (cl:read stream nil nil)
            while word
            do
        (tv:note-progress index *nuss-dictionary-size*)
        (setf (aref table index) word)
            finally
              (return table)))))

(defun read-label (word)
  "This function convert label into transcription
   flavor object."
  (loop with trans = (cdr word)
        for content in trans
        for phonetic-trans = (caar content)
        for event = (mapcar 'car (second content))
        for alophone = (car (third content))
        for insep = (car (fourth content))
        for vowel-center = (second (fifth content))
        collect
        (make-instance 'transcription-flavor
                       :phonetic-transcription phonetic-trans
                       :event-transcription event
                       :alophone-transcription alophone
                       :insep-transcription insep
                       :vowel-center-transcription vowel-center)))

(defmethod (:convert-label-to-phonetic-property-list
            transcription-flavor) ()
  "this is the method for converting label
   into phonetic property flavor."
  (let* ((symbol phonetic-transcription))
    (if (loop for char being the array-element of symbol
              thereis (equal char #\,))
        ;
        ; May occour some assimilation
        ;
        (cond ((string= alophone-transcription "dv")
               (this-is-devoiced-vowel-cluster symbol))
               ; ther is dv transcription

              ((loop for char being the array-element of symbol
                     always (or (vowelp char) (equal char #\,)))
               (this-is-vowel-cluster symbol))
              ; Consists of only vowels

              ((loop for char being the array-element of symbol
                     thereis (equal char #\N))
               (this-is-nasal-cluster symbol))
              ; nasal cluster

              ((loop for char being the array-element of symbol
                     thereis (voicedp char))
               (this-is-voiced-cluster symbol))
              ; Somewhat voiced cluster

              (t
               (this-is-voiceless-cluster symbol)))
        ;
        ; Normal symbol
        ;
        (inquire-phonetic-properties symbol))))

(defun this-is-voiceless-cluster (sym)
  (let* ((key-phoneme
          (loop for ph in (string-to-phoneme-list (delete-comma sym))
                for pp-of-ph = (inquire-phonetic-properties ph)
                if (not (phonetic-property-flavor-voiced? pp-of-ph))
                  do (return ph))))
    (make-inseparable-phonetic-property key-phoneme
                                        sym
                                        "voiceless cluster")))

(defun this-is-voiced-cluster (sym)
  (let* ((key-phoneme
          (loop for ph in (string-to-phoneme-list (delete-comma sym))
                for pp-of-ph = (inquire-phonetic-properties ph)
                if (phonetic-property-flavor-voiced? pp-of-ph)
                  do (return ph))))
    (make-inseparable-phonetic-property key-phoneme
                                        sym
                                        "nasal cluster")))

(defun this-is-nasal-cluster (sym)
  (make-inseparable-phonetic-property "N"
                                      sym
                                      "nasal cluster"))

(defun this-is-vowel-cluster (sym)
  (let* ((key-phoneme
          (loop for ph in (string-to-phoneme-list (delete-comma sym))
                if (or (string= "a" ph)
                       (string= "i" ph)
                       (string= "u" ph)
                       (string= "e" ph)
                       (string= "o" ph))
                  do (return ph))))
    (if (or (string= sym "o,u")
            (string= sym "e,i")
            (string= sym "a,a")
            (string= sym "i,i")
            (string= sym "u,u")
            (string= sym "e,e")
            (string= sym "o,o"))
        (make-inseparable-phonetic-property key-phoneme
                                            (cond ((string= sym "o,u") "O")
                                                  ((string= sym "e,i") "E")
                                                  (T sym))
                                            "long vowel")
        (make-inseparable-phonetic-property key-phoneme
                                            sym
                                            "vowel cluster"))))

(defun this-is-devoiced-vowel-cluster (sym)
  (let* ((key-phoneme
          (loop for ph in (string-to-phoneme-list (delete-comma sym))
                if (and (not (string= "a" ph))
                        (not (string= "i" ph))
                        (not (string= "u" ph))
                        (not (string= "e" ph))
                        (not (string= "o" ph)))
                  do
                  (return ph))))
    (make-inseparable-phonetic-property key-phoneme
                                        sym
                                        "devoiced vowel cluster")))
(defun delete-comma (string)
  (let* ((char-list
          (loop for c being the array-element of string
                if (not (char= #\, c))
                  collect c)))
```

```
143       (cl:concatenate 'string char-list)))
144
145
146   (defun make-inseparable-phonetic-property (key-ph
147                                              sym
148                                              insep)
149     (let* ((pp (inquire-phonetic-properties key-ph)))
150       (make-instance 'phonetic-property-flavor
151                      :symbol sym
152                      :vowel? (phonetic-property-flavor-vowel? pp)
153                      :voiced? (phonetic-property-flavor-voiced? pp)
154                      :manner (phonetic-property-flavor-manner pp)
155                      :place (phonetic-property-flavor-place pp)
156                      :palatalized? (phonetic-property-flavor-palatalized? pp)
157                      :gemination (phonetic-property-flavor-gemination pp)
158                      :inseparable insep
159                      :top (top-phoneme sym)
160                      )))
161
162
163   (defun voicedp (c)
164     (loop with voiced = "bdgzjmnywrN"
165           for v being the array-element of voiced
166           thereis (equal c v)))
167
168   (defun top-phoneme (seq)
169     (cl:concatenate
170       'string
171       (loop for c being the array-element of seq
172             while (not (equal c #\,))
173             collect c)))
174
175
176   ;;;
177   ;;;
178   ;;;
179
180   (defun make-pp-table (label-table)
181     (tv:noting-progress ("generating phonetic property table")
182       (loop with result-array = (make-array *nuss-dictionary-size*)
183             for entry being the array-element of label-table
184             for label = (read-label entry)
185             for index from 0
186             do
187         (tv:note-progress index *nuss-dictionary-size*)
188         (setf (aref result-array index)
189               (loop for phoneme in label
190                     collect
191                       (send
192                         phoneme
193                         :convert-label-to-phonetic-property-list)))
194         finally
195           (return result-array))))
196
197   ;;
198   ;; Get phonetic property of label
199   ;;
200   (defun get-pp-from-label (no)
201     (let* ((transcriptions
202               (read-label (get-label no))))
203       (loop for lb in transcriptions
204             for pp-lb =
205                 (send lb :convert-label-to-phonetic-property-list)
206             collect pp-lb)))
207
208   ;;
209   ;;
210   ;; Get-label
211   ;;
212
213   (defun get-label (no)
```



```
214     "This function gives label transcripiton of the word"
215     (or (aref *nuss-label-table* (1- no))))
216
217
218   ;;
219   ;; Hey Mr. Kaiki, THis is the new version of
220   ;; Expand database into SUE !!!!
221   ;; You know, I found Bad Bug, in SUE that force me to write this again
222   ;;  November 17, 1989
223   ;;
224   (defun expand-db-into-SUE (file)
225     "This function generates SUE dictionary from label table
226      which is loaded on the memory as *nuss-lable-table*."
227     (with-open-file
228       (stream file :direction :output)
229   ;     (loop for index from 1947 below 1948
230       (loop for index from 1 below *nuss-dictionary-size*
231             for word = (get-pp-from-label index)
232             for subs = (get-arbiterary-sub-strings word)
233             do
234         (loop for sue in subs
235               for string = (unit-flavor-unit-string sue)
236               for stt = (unit-flavor-unit-stt sue)
237               for end = (unit-flavor-unit-end sue)
238               for stcr = (code-criterion
239                            (unit-flavor-start-criterion sue))
240               for encr = (code-criterion
241                            (unit-flavor-end-criterion sue))
242               for rc = (unit-flavor-right-context sue)
243               for lc = (unit-flavor-left-context sue)
244               do
245           (format stream "~a ~a ~a ~a ~a ~a ~a ~a ~a ~a ~a ~a ~a~%"
246                   string stt end index stcr encr
247                   (phonetic-property-flavor-symbol lc)
248                   (if (phonetic-property-flavor-voiced? lc) 1 0)
249                   (if (phonetic-property-flavor-palatalized? lc) 1 0)
250                   (if (phonetic-property-flavor-gemination lc) 1 0)
251                   (phonetic-property-flavor-symbol rc)
252                   (if (phonetic-property-flavor-voiced? rc) 1 0)
253                   (if (phonetic-property-flavor-palatalized? rc) 1 0)
254                   (if (phonetic-property-flavor-gemination rc) 1 0))))))
255
256   (defun code-criterion (name)
257     (if (or (string= name "initial")
258             (string= name "final"))
259         0
260     (loop for cr in *nuss-criterion-priority*
261           for no from 1
262           do
263       (if (string= cr name)
264           (return no))
265       finally (return -1))))
266
267   (defun get-arbiterary-sub-strings (string)
268     "This function breaks down the given phonetic-sequence or
269      string into a list of arbiterary phonemic sequences aplaying
270      the break point criteria as the order of
271      *nuss-criterion-priority-for-sue-generation."
272     (let* ((pp (if (listp string)
273                    string
274                    (convert-string-to-phonetic-property-list string)))
275            (st (if (listp string)
276                    (delete-comma (inquire-string string))
277                    string))
278            (break-points
279              (get-break-point-for-sue
280                pp
281                *nuss-criterion-priority-for-sue-generation*
282                ))
283            ; The break-points shlod be sorted in left-to-right order
284            (stt (make-instance 'break-point-flavor
```

```
285                                 :location 0
286                                 :criterion "initial"))
287           (end (make-instance 'break-point-flavor
288                               :location (length pp)
289                               :criterion "final"))
290           (stt-list
291             (cons stt
292                   (loop for pre = stt then brkp
293                         for brkp in break-points
294                         for lc = (break-point-flavor-location brkp)
295                         for pre-lc = (break-point-flavor-location pre)
296                         if (or (intp lc)
297                                (not (= (floor lc) pre-lc)))
298                           collect brkp)))
299           (end-list
300             (reverse
301               (cons end
302                     (loop for fol = end then brkp
303                           for brkp in (reverse break-points)
304                           for lc = (break-point-flavor-location brkp)
305                           for fol-lc = (break-point-flavor-location fol)
306                           if (or (intp lc)
307                                  (not (= (ceiling lc) fol-lc)))
308                             collect brkp))))
309
310           (nul-context (inquire-phonetic-properties "#")))
311       (loop with result-list
312             for stt in stt-list
313             for stt-loc = (break-point-flavor-location stt)
314             for stt-int = (floor stt-loc)
315             do
316         (loop for end in end-list
317               for end-loc = (break-point-flavor-location end)
318               for end-int = (ceiling end-loc)
319               for unst = (inquire-string (sublist pp stt-int end-int))
320               if (< stt-loc end-loc)
321                 do
322           (setq result-list
323                 (cons (make-instance
324                         'unit-flavor
325                         :input-string st
326                         :unit-string unst
327                         :unit-stt stt-loc
328                         :unit-end end-loc
329                         :start-criterion
330                         (break-point-flavor-criterion stt)
331                         :end-criterion
332                         (break-point-flavor-criterion end)
333                         :right-context
334                         (or (nth end-int pp) nul-context)
335                         :left-context
336                         (if (< stt-int 1)
337                             nul-context
338                             (nth (- stt-int 1) pp))
339
340                         )
341                         result-list)))
342               finally (return result-list))))
343
344
345   (defun get-break-point-for-sue (string criterion-list)
346     "This function generates break-point-list, the contents of which is
347      (break-point location, corresponding criteria)"
348     (loop with result
349           with loc-list
350           with list = (get-break-point-inf string criterion-list)
351           for break-point-inf in list
352           do
353       (loop with break-points = (second break-point-inf)
354             for brp in break-points
355             for loc = (break-point-flavor-location brp)
```



```
356             for check-loc = loc;(if (intp loc) loc (ceiling loc))
357             if (not (member check-loc loc-list))
358               ; If the break point has never used before
359               do
360                 (setf loc-list (cons loc loc-list))
361                 (setf result (cons brp result)))
362       finally (return
363                 (sort result
364                       '(lambda (x y)
365                          (< (break-point-flavor-location x)
366                             (break-point-flavor-location y)))))))
367
```

```
1     ;;; -*- mode: lisp; package: zetalisp-user; base: 10 -*-
2     ;;;
3     ;;; File-name label-functions.lisp [~/Lisp/synthesis]
4     ;;;   Kin Takeda  ATR,
5     ;;;
6     ;;;    In this file, the tools for manipulating label trancriptions are stored.
7     ;;;
8     ;;;                 June     19      1989 at ATR
9
10    (defun label-phonetic-trans-list (label)
11      "list up phonetic transcription of label."
12      (loop for phn in (cdr label)
13            collect (car phn)))
14
15    (defun label-phonetic-symbol-list (label)
16      "list up phonetic symbol of label."
17      (loop for phn in (cdr label)
18            for symbol = (first (car phn))
19            collect symbol))
20
21    (defun label-event-trans-list (label)
22      "list up event transcriptions of label."
23      (loop with result-list
24            for phn in (cdr label)
25            for eve-trans = (second phn)
26            do
27        (loop for eve in eve-trans
28              do
29          (setq result-list (cons eve result-list)))
30        finally
31          (return (reverse result-list))))
32
33    (defun label-event-symbol-list (label)
34      "list up event symbol of label."
35      (loop with result-list
36            for phn in (cdr label)
37            for eve-trans = (second phn)
38            do
39        (loop for eve in eve-trans
40              for eve-symbol = (car eve)
41              do
42          (setq result-list (cons eve-symbol result-list)))
43        finally
44          (return (reverse result-list))))
45
46    (defun label-phonetic-segmentation (label phn-no)
47      "This function returns the start and end time of the phonetic label period."
48      (let* ((the-phn (nth (max 0 (- phn-no 1)) (cdr label)))
49             (stt-time (second (car the-phn)))
50             (end-time (third (car the-phn))))
51        (list stt-time end-time)))
52
53
54    (defun get-word (no)
55      (eval (cons 'cl:concatenate (cons ''string (mapcar 'caar (cdr (get-label no)))))))
```

```
;;; -*- mode: lisp; package: zetalisp-user; base: 10 -*-
;;;
;;;
;;;   Pre-selection of synthesis units
;;;
;;;      In this file, some programs for pre-selection of the optimal units
;;;      are written. Roughly speaking, these programs needs the label table,
;;;      phonetic-property table and sue-dictionary.

(defun append-on (list atom)
  (if list
      (loop for x in list
            collect (cons atom x) into res
            finally (return (cl:concatenate 'list
                                            (cons (list atom) res)
                                            list)))
      (list (list atom))))

(defun get-all-sublist (list)
  (loop with result
        for x in list
        do
    (setf result (append-on result x))
        finally
          (return (cons nil result))))

;;
;; A bew version of get-lattice whcih is used for generates unit sequence candidates.
;; Oct. 13 1989, by K.K.Takeda
;;

(defun get-lattice-new (string criterion)
  "This function returns a list of the possible unit sequences according to
   the given break-point-criterion.
   Each unit sequence is expressed as a list of (score #<UNINT-FLAVOR>s)."
  (let* ((input-string
            (if (listp string)
                (inquire-string string)
                string))
         (pp-list (get-pp string))
         (length-of-pp (length pp-list))
         (start-break-point
           (make-instance 'break-point-flavor
                          :location 0
                          :criterion "initial"
                          :score 0
                          :context "initial"))
         (end-break-point
           (make-instance 'break-point-flavor
                          :location length-of-pp
                          :criterion "final"
                          :score 0
                          :context "final"))
         (break-point-list
           (send (activate-criterion criterion)
                 :break-string string))
         (combination
           (get-all-sublist break-point-list)))
    (loop for index from 0
          for x in combination
          for seq = (cl:concatenate 'list
                                    (list start-break-point)
                                    x
                                    (list end-break-point))
          if x
            collect
              (get-units-from-break-points seq
                                           pp-list
                                           input-string))))

(defun get-lattice-incrimentally (string criterion)
```



```
  "This function returns a list of the possible unit sequences according to
   the order of the given break-point-criteria.
   Each unit sequence is expressed as a list of (score #<UNINT-FLAVOR>s)."
  (let* ((input-string
            (if (listp string)
                (inquire-string string)
                string))
         (pp-list
           (if (listp string)
               string
               (convert-string-to-phonetic-property-list string)))
         (length-of-pp (length pp-list))
         (start-break-point
           (make-instance 'break-point-flavor
                          :location 0
                          :criterion "initial"
                          :score 0
                          :context "initial"))
         (end-break-point
           (make-instance 'break-point-flavor
                          :location length-of-pp
                          :criterion "final"
                          :score 0
                          :context "final"))
         (break-point-list
           (get-break-point pp-list (list criterion)))
         (combination
           (get-all-sublist break-point-list)))
    (loop for index from 0
          for x in combination
          for seq = (cl:concatenate 'list
                                    (list start-break-point)
                                    x
                                    (list end-break-point))
          collect
            (get-units-from-break-points seq
                                         pp-list
                                         input-string))))

;;
(defun get-units-from-break-points (brps pp-list input-string)
  (let* ((word-boundary (inquire-phonetic-properties "#"))
         (srt-brps (sort brps
                         '(lambda (x y)
                            (< (break-point-flavor-location x)
                               (break-point-flavor-location y))))))
    (loop for stt in (reverse (cdr (reverse srt-brps)))
          for stt-loc = (break-point-flavor-location stt)
          for stt-cr = (break-point-flavor-criterion stt)
          for end in (cdr srt-brps)
          for end-loc = (break-point-flavor-location end)
          for end-cr = (break-point-flavor-criterion end)
          for stt-in-int = (if (cl:integerp stt-loc) stt-loc (floor stt-loc))
          for end-in-int = (if (cl:integerp end-loc) end-loc (floor (1+ end-loc)))
          for l-c = (if (< 0 stt-in-int) (nth (1- stt-in-int) pp-list) word-boundary)
          ; My be this is bug
          ; for r-c = (or (nth (1+ end-in-int) pp-list) word-boundary)
          for r-c = (or (nth end-in-int pp-list) word-boundary)
          for cost = (break-point-flavor-score stt)
          if (not (= stt-loc end-loc))
          collect (make-instance 'unit-flavor
                                 :input-string input-string
                                 :unit-string
                                 (inquire-string
                                   (sublist pp-list stt-in-int end-in-int))
                                 :unit-stt stt-loc
                                 :unit-end end-loc
                                 :start-criterion stt-cr
                                 :end-criterion end-cr
                                 :right-context r-c
                                 :left-context l-c) into units
```

```
143                 if (not (= stt-loc end-loc))
144                   sum cost into score
145   ;             do                                ;
146   ;          (format t "STT = ~a END = ~a~%" stt-loc end-loc)
147   ;          For debug
148                 finally (return (list score units)))))
149
150
151   ;;
152   ;; Get-Substirng is slightly different from get-lattice-new
153   ;;
154   (defun get-substring-new (string &optional (criterion-list *nuss-criterion-priority*)
      )
155     "This function returns a list of the possible unit sequences according to
156      the order of the given break-point-criteria.
157      Each unit sequence is expressed as a list of (score #<UNINT-FLAVOR>s)."
158     (let* ((input-string
159               (if (listp string)
160                   (inquire-string string)
161                   string))
162            (pp-list
163              (if (listp string)
164                  string
165                  (convert-string-to-phonetic-property-list string)))
166            (length-of-pp (length pp-list))
167            (start-break-point
168              (make-instance 'break-point-flavor
169                             :location 0
170                             :criterion "initial"
171                             :score 0
172                             :context "initial"))
173            (end-break-point
174              (make-instance 'break-point-flavor
175                             :location length-of-pp
176                             :criterion "final"
177                             :score 0
178                             :context "final"))
179            (word-boundary (inquire-phonetic-properties "#"))
180            (break-point-list
181              (get-break-point pp-list criterion-list))
182            (sorted-break-points
183              (cl:concatenate 'list
184                              (list start-break-point)
185                              (sort
186                                (cl:delete-duplicates
187                                  break-point-list
188                                  :key #'break-point-flavor-location
189                                  :replace t)
190                                '(lambda (x y) (< (break-point-flavor-location x)
191                                                  (break-point-flavor-location y))))
192                              (list end-break-point))))
193       (loop with units
194             for stt in sorted-break-points
195             for stt-loc = (break-point-flavor-location stt)
196             for stt-cr = (break-point-flavor-criterion stt)
197             for stt-in-int = (if (cl:integerp stt-loc) stt-loc (floor stt-loc))
198             for l-c = (if (< 0 stt-in-int) (nth (1- stt-in-int) pp-list) word-boundary)
199             do
200               (loop for end in sorted-break-points
201                     for end-loc = (break-point-flavor-location end)
202                     for end-cr = (break-point-flavor-criterion end)
203                     for end-in-int = (if (cl:integerp end-loc) end-loc (floor (1+ end-l
    oc)))
204                     for r-c = (or (nth end-in-int pp-list) word-boundary)
205                     if (< stt-loc end-loc)
206                       do
207                       (setf units
208                             (cons
209                               (make-instance
210                                 'unit-flavor
211                                 :input-string input-string
```



```
212                                 :unit-string
213                                 (inquire-string
214                                   (sublist pp-list stt-in-int end-in-int))
215                                 :unit-stt stt-loc
216                                 :unit-end end-loc
217                                 :start-criterion stt-cr
218                                 :end-criterion end-cr
219                                 :right-context r-c
220                                 :left-context l-c) units)))
221                     finally (return units))))
222
223   ;;
224   ;; BREAK-POINT-FLAVOR
225   ;;
226   (defflavor break-point-flavor
227           ((location)
228            (score)
229            (criterion)
230            (context))
231           ()
232     (:writable-instance-variables)
233     :initable-instance-variables)
234
235   (defun get-pp (sequence)
236     (if (listp sequence) sequence
237         (zl-user::convert-string-to-phonetic-property-list
238           sequence)))
239
240   (defmethod (:break-string break-point-criterion-flavor) (sequence)
241     (let* ((phonetic-property-list (get-pp sequence)))
242       (loop for prev = (inquire-phonetic-properties "#")
243                     then curr
244             for curr in phonetic-property-list
245             for position from 0
246             if (funcall predicate prev curr)
247               collect
248                 (make-instance
249                   'break-point-flavor
250                   :location (if boundary-location
251                                 (- position 0.5)
252                                 position)
253                   :score score
254                   :criterion name
255                   :context (if boundary-location
256                                (list (phonetic-property-flavor-symbol prev))
257                                (list (phonetic-property-flavor-symbol prev)
258                                      (phonetic-property-flavor-symbol curr)))))))
259
260   (defun get-break-point-inf (string criterion-list)
261     "This function creates Break-Point-Information-Table, which containes
262      the information of break-point as follows.
263
264        Name of criterion            priority  list of break points
265      ((voiceless stop criterion      1       (1 3 5 10))
266       (vowel center criterion        2       (bra bra ......))
267       .. etc. )"
268     (loop for criterion in criterion-list
269           for act-crn = (activate-criterion criterion)
270           for break-points = (send act-crn :break-string string)
271           collect (list (break-point-criterion-flavor-score act-crn) break-points)))
272
273   (defun get-break-point (string criterion-list)
274     "This function generates break-point-list, the contents of which is
275      (break-point location, corresponding criteria)"
276     (loop with result
277           with loc-list
278           with list = (get-break-point-inf string criterion-list)
279           with sorted-list = (sort list '(lambda (x y)
280                                            (< (car x)
281                                               (car y))))
282           for break-point-inf in sorted-list
```

```
283           do
284       (loop with break-points = (second break-point-inf)
285             for brp in break-points
286             for loc = (break-point-flavor-location brp)
287             for check-loc = (if (intp loc) loc (ceiling loc))
288             if (not (member check-loc loc-list))
289               ; If the break point has never used before
290               do
291                 (setf loc-list (cons loc loc-list))
292                 (setf result (cons brp result)))
293       finally (return result)))
294
295   ;;
296   ;; FOR THE USEAGE OF ART
297   ;;
298   (defun load-sequence (sequence)
299     "This would be called by art through the acu::load-sequnce."
300     (list (car sequence)
301     (loop for unit in (second sequence)
302           collect
303           (let* ((end-criterion (unit-flavor-end-criterion unit))
304            (start-criterion (unit-flavor-start-criterion unit))
305            (string (unit-flavor-unit-string unit))
306            (start (unit-flavor-unit-stt unit))
307            (end (unit-flavor-unit-end unit)))
308             (list string start end start-criterion end-criterion)))))
```

```
;;; -*- mode: lisp; package: zetalisp-user; base: 10 -*-
;;;
;;;
;;;   Pre-selection of synthesis units
;;;
;;;      In this file, some programs for pre-selection of the optimal units
;;;      are written. Roughly speaking, these programs needs the label table,
;;;      phonetic-property table and sue-dictionary.

;;
;; The priority for criterion aplying should be defined,
;; as *nuss-criterion-priority*
(defvar *nuss-criterion-priority*
        '("voiceless stop criterion"
          "voiceless fricative criterion"
          "vowel center criterion"
          "voiced fricative criterion"
          "voiced stop criterion"
          "VC boundary criterion"))

(defvar *nuss-criterion-priority-for-sue-generation*
; This criterion order is putting the VOWEL CENTER
; criterion to be applied finally, especially for
; generating sue dictionary."
        '("voiceless stop criterion"
          "voiceless fricative criterion"
          "voiced fricative criterion"
          "voiced stop criterion"
          "VC boundary criterion"
          "vowel center criterion"))

(defflavor entry-flavor
        ((word-no)
         (word-stt)
         (word-end)
         (string))
        ()
  (:writable-instance-variables)
  :initable-instance-variables)

(defflavor unit-flavor
        ((input-string)
         (unit-string)
         (unit-stt)
         (unit-end)
         (start-criterion)
         (end-criterion)
         (right-context)
         (left-context)
         (corresponding-entries))
        ()
  (:writable-instance-variables)
  :initable-instance-variables)

;;
;; Function get-lattice
;;
;;
(defun get-lattice (string &optional (criterion-list *nuss-criterion-priority*))
  "This function returns a list which consits of unit-flavor."
  (let* ((result-list)
         (phonetic-property-list
           (if (listp string)
               string
               (convert-string-to-phonetic-property-list string)))
         (vowel-break-points
           (if (member "vowel center criterion" criterion-list)
               (get-vowel-break-points string)
               (make-array 1 :initial-value nil)))
         (normal-criterion-list
           (remove "vowel center criterion" criterion-list))
```



```
         (normal-break-points
           (get-break-points string normal-criterion-list))
         (normal-break-points-list
           (loop for break-point being the array-element of normal-break-points
                 for index from 0
                 if break-point collect index))
         (vowel-break-points-list
           (loop for break-point being the array-element of vowel-break-points
                 for index from 0
                 if break-point collect index)))
    ;;
    ;; Begining and Ending criteria are both normal
    (setf result-list (cl:concatenate 'list
                                      (make-lattice
                                        normal-break-points-list
                                        normal-break-points-list
                                        phonetic-property-list
                                        normal-break-points
                                        normal-break-points)
                                      result-list))
    ;;
    ;; Begining criterion is normal Ending is vowel
    (setf result-list (cl:concatenate 'list
                                      (make-lattice
                                        normal-break-points-list
                                        vowel-break-points-list
                                        phonetic-property-list
                                        normal-break-points
                                        vowel-break-points)
                                      result-list))
    ;;
    ;; Begining criterion is vowel Ending is normal
    (setf result-list (cl:concatenate 'list
                                      (make-lattice
                                        vowel-break-points-list
                                        normal-break-points-list
                                        phonetic-property-list
                                        vowel-break-points
                                        normal-break-points)
                                      result-list))
     ;;
    ;; Begining and Ending criteria are both vowel
    (setf result-list (cl:concatenate 'list
                                      (make-lattice
                                        vowel-break-points-list
                                        vowel-break-points-list
                                        phonetic-property-list
                                   vowel-break-points
                                        vowel-break-points)
                                      result-list))
    (cl:remove-duplicates
      result-list
      :test '(lambda (a b) (unit-flavor-quasi-eql a b)))))

(defun make-lattice (start-break-points
                     end-break-points
                     pp-list
                     start-bp-array
                     end-bp-array)
  (loop with result-list
        for stt in start-break-points
        do
    (loop for end in end-break-points
          for unit-string = (inquire-string
                              (sublist pp-list stt end))
          do
      (if (< stt end)
          (setq result-list (cons
                              (make-instance 'unit-flavor
                                :input-string (inquire-string pp-list)
                                :unit-string unit-string
```

```
                                    :unit-stt (1+ stt)
                                    :unit-end end
                                    :start-criterion (aref start-bp-array stt)
                                    :end-criterion (aref end-bp-array end))
                                  result-list))))
        finally
          (return result-list)))

(defun unit-flavor-quasi-eql (a b)
  "This method compare two units and if both original text and location
   in the text are the same, returns T"
  (and (string= (unit-flavor-input-string a)
                (unit-flavor-input-string b))
       (string= (unit-flavor-unit-string a)
                (unit-flavor-unit-string b))
       (string= (unit-flavor-start-criterion a)
                (unit-flavor-start-criterion b))
       (string= (unit-flavor-end-criterion a)
                (unit-flavor-end-criterion b))
       (= (unit-flavor-unit-stt a)
          (unit-flavor-unit-stt b))
       (= (unit-flavor-unit-end a)
          (unit-flavor-unit-end b))))

;;
;; By this function, we can obtain a sub-string lattice
;; which consists of duplicated vowel.
;;
;;
;;
;; An inter face of ART
;;
(defun load-lattice-element (sue)
  "This function generates elements of lattice's list"
  (let* ((end-criterion (unit-flavor-end-criterion sue))
         (start-criterion (unit-flavor-start-criterion sue))
         (string (unit-flavor-unit-string sue))
         (start (unit-flavor-unit-stt sue))
         (end (unit-flavor-unit-end sue)))
    (list string start end start-criterion end-criterion)))


;(defun get-break-points-array (string criterion-list)
;  "This function retuens an array which shows how the input string should be
;   break down."
;  (loop with phonetic-property-list
;          = (if (listp string)
;                string
;                (convert-string-to-phonetic-property-list string))
;        with break-point-array = (make-array (+ (length phonetic-property-list) 1)
;                                             :initial-value nil)
;        for criterion in criterion-list
;        for activated-criterion = (activate-criterion criterion)
;        for break-points
;            = (send activated-criterion :search-break-point phonetic-property-list)
;        do
;    (loop for break-point in break-points
;          do
;      (if (aref break-point-array break-point)
;          ()
;          (setf (aref break-point-array break-point) criterion)))
;    finally
;      (return break-point-array)))

(defun get-break-points-array (string criterion-list)
  "This function retuens an array which shows how the input string should be
   break down."
  (loop with phonetic-property-list
          = (if (listp string)
                string
                (convert-string-to-phonetic-property-list string))
```



```
        with break-point-array = (make-array (+ (length phonetic-property-list) 1)
                                             :initial-value nil)
        for criterion in criterion-list
        for activated-criterion = (activate-criterion criterion)
        for break-points
            = (send activated-criterion :search-break-point phonetic-property-list)
        do
    (loop for break-point in break-points
          do
      (if (aref break-point-array break-point)
          ()
          (setf (aref break-point-array break-point) criterion)))
    finally
      (return break-point-array)))

(defun get-break-points (string criterion-list)
  "This function retuens an array which shows how the input string should be
   break down."
  (loop with phonetic-property-list
          = (if (listp string)
                string
                (convert-string-to-phonetic-property-list string))
        with break-point-array = (make-array (+ (length phonetic-property-list) 1)
                                             :initial-value nil)
        for criterion in criterion-list
        for activated-criterion = (activate-criterion criterion)
        for break-points
            = (send activated-criterion :search-break-point phonetic-property-list)
        initially
          (setf (aref break-point-array 0)
                "initial")
          (setf (aref break-point-array (length phonetic-property-list))
                "final")

        do
    (loop for break-point in break-points
          do
      (if (aref break-point-array break-point)
          ()
          (setf (aref break-point-array break-point) criterion)))
    finally
      (return break-point-array)))

(defun get-vowel-break-points (string)
  "This function retuens an array which shows how the input string should be
   break down."
  (loop with phonetic-property-list
          = (if (listp string)
                string
                (convert-string-to-phonetic-property-list string))
        with break-point-array = (make-array (+ (length phonetic-property-list) 1)
                                             :initial-value nil)
        with activated-criterion = (activate-criterion "vowel center criterion")
        with break-points
            = (send activated-criterion :search-break-point phonetic-property-list)
        for break-point in break-points
        initially
          (setf (aref break-point-array 0)
                "initial")
          (setf (aref break-point-array (length phonetic-property-list))
                "final")
        do
    (if (aref break-point-array break-point)
        ()
        (setf (aref break-point-array break-point) "vowel center criterion"))
        finally
          (return break-point-array)))
```

```
;;; -*- mode: lisp; package: art-user; base: 10 -*-
;;

(defun do-at-onece (dir from to)
  "This function execute unit selection on multiple files"
  (loop for num from from to to
        for h-num = (format nil "~3,'0d" num)
        for header = (cl:concatenate 'string dir h-num)
        do
    (run-unit-select header)))


(defun run-unit-select (file-name-header &key (path "lm06:>takeda>us>data>")
                                              (debug nil)
                                              (phrase nil))
  "This function execute unit-select all at once"
  (let* ((ph-file (concatenate 'string path file-name-header ".ph"))
         (pp-file (concatenate 'string path file-name-header ".pp"))
         (header (concatenate 'string path file-name-header))
         (result-file (concatenate 'string path file-name-header ".ett"))
         (out-put-file (concatenate 'string path file-name-header ".oett"))
         (seg-inf-file (concatenate 'string path file-name-header ".seg"))
         (seq-inf-file (concatenate 'string path file-name-header ".seq"))
         (unt-inf-file (concatenate 'string path file-name-header ".unt"))
         (text (read-org-file ph-file))
         (pitch (read-org-file pp-file))
         (execute-on-special-phrase (if phrase nil t))
         (execute-on-this-phrase phrase)
         (stt-time (time)))
    (if (not debug)
        (progn
          (if-exists-make-new-version result-file)
          (if-exists-make-new-version out-put-file)
          (if-exists-make-new-version seg-inf-file)
          (if-exists-make-new-version seq-inf-file)
          (if-exists-make-new-version unt-inf-file)))
    (zl:loop for pre = nil then tx
             for ph from 1
             for tx in text
             for pt in pitch
             for left = (if (and pre (not (zerop (fifth pre))))
                            (car (reverse (zl-user::string-to-phoneme-list
                                           (second pre))))
                            "#")
             for right = (if (and (< ph (length text)) (not (zerop (fifth tx))))
                             (car (zl-user::string-to-phoneme-list
                                   (second (nth ph text))))
                             "#")
             if (or execute-on-special-phrase
                    (= ph execute-on-this-phrase))
       do
      (format t "~%          **********~%" ph)
      (format t   "          ***** For the ~a-th phrase *****~%" ph)
      (format t   "          **********~%" ph)
      (reset)
      (if debug
          (eval '(assert (debug))))
      (eval '(assert (phrase-left-context ,left)))
      (eval '(assert (phrase-right-context ,right)))
      (eval '(assert (file-name-header ,header)))
      (eval '(assert (text ,tx)))
      (eval '(assert (target-pitch-contour ,pt)))
      (run :silent-p t)
      (if (and (not debug) (> (fifth tx) 0))
          (with-open-file
            (res-stream out-put-file
                        :direction :output
                    :if-exists :append)
            (format res-stream "~%(~a 100 0 0 0 ~s ~s)"
                    ph (string "pau") (string "pau"))))
    )
```



```
    (if (not debug)
        (sort-ett-file header))
    (format t "~% Total Run Time was, ~a~%" (/ (- (time) stt-time) 3600.0))))

(defun if-exists-make-new-version (file-name)
  "Examine if specified file exists or not, and if exists creates
   a new version of it."
  (let ((dummy-stream))
    (if (setq dummy-stream (open file-name :direction :probe))
        (let ()
          (close dummy-stream)
          (close (open file-name :direction :output :if-exists :new-version))))))


(defun sort-ett-file (header)
  "This function sorts the output of the art's unit-selection"
  (let* ((out-file (concatenate 'string header ".ett"))
         (in-file (concatenate 'string header ".oett"))
         (result-list
           (with-open-file
             (in-stream in-file :direction :input)
             (zl:loop for result = (read in-stream nil nil)
                      while result
                      collect result)))
         (sorted-data
           (sort
             (sort result-list '(lambda (a b) (< (second a) (second b))))
             '(lambda (a b) (< (car a) (car b))))))
    (with-open-file
      (out-stream out-file :direction :output
                           :if-does-not-exist :create
                           :if-exists :append)
      (zl:loop for data in sorted-data
               for word-no = (nth 2 data)
               for stt = (nth 3 data)
               for end = (nth 4 data)
               for unit-description = (delete-comma (nth 5 data))
               for word-description = (delete-comma (nth 6 data))
               do
        (format out-stream "~a ~a ~a ~a ~a ~%"
                           word-no
                           stt
                           end
                           unit-description
                           word-description)))))

(defun delete-comma (string)
  (let* ((char-list
           (zl:loop for c being the array-element of string
                    if (not (char= #\, c))
                    collect c)))
    (concatenate 'string char-list)))

(defun read-org-file (file-name)
  " comment !! "
  (with-open-file
    (stream file-name :direction :input)
    (zl:loop for data = (read stream nil nil)
             while data collect data)))

;; Tools
(defun mean (value-list)
  "This function returns the mean value of the argument list."
  (if value-list
      (let* ((length (length value-list))
             (sum (eval (cons '+ value-list))))
        (if (zerop length)
            0
            (/ (float sum) length)))
      0))
```

```
;;; -*- mode: lisp; package: zetalisp-user; base: 10 -*-
;;;
;;;
;;;   Defined functions
;;;         convert-string-to-phonetic-properties
;;;         string-to-phoneme-symbols
;;;         vowelp
;;;         inquire-phonetic-property
;;;         inquire-string

;;
;; Function convert-string-to-phonetic-properties
;;

(defun convert-string-to-phonetic-property-list (string)
  "This function converts a string into a list of phonetic properties"
  (let* ((phoneme-sequence (string-to-phoneme-list string)))
    (if phoneme-sequence
        (progn
          (loop for phn in phoneme-sequence
                collect (inquire-phonetic-properties phn)))
        (format 't "Error: in parsing input string !~%"))))

;;
;; Function string-to-phoneme-list
;;

(defun string-to-phoneme-list (string)
  "This function separates a phonemic-symbol string into a list of elemental symbols"
  (loop with phn
        with result-list
        for char being the array-element of string
        do
    (if (or (vowelp char)
            (char= char #\N)
            (or (char= char #\#) (char= char #\space)))
        ;If input char is vowel or syllabic nasal then there is a boundary of CV-syllable
        (progn
          (if phn
              (if (roman-style-p phn)
                  (setq result-list (cons phn result-list ))
                  (progn (dbg)
                    (format 't "~%Error: [in string-to-phoneme-list]~%")
                    (format 't "       Fail to parse input string. ~%")
                    (return nil))))
          (setq result-list (cons (string char) result-list))
          (setq phn ()))
        (setq phn (cl:concatenate 'string phn (string char))))
        finally
          (if phn
              (if (roman-style-p phn)
                  (setq result-list (cons phn result-list ))
                  (progn (dbg) (format 't "~%Error: [in string-to-phoneme-list]~%")
                         (format 't "       Fail to parse input string. ~%"))
                  (return nil)))
          (if result-list
              (return (reverse result-list)))))

(defun vowelp (char)
  (or (char= char #\a) (char= char #\i) (char= char #\u) (char= char #\e)
      (char= char #\o) (char= char #\A) (char= char #\I) (char= char #\U)
      (char= char #\E) (char= char #\O)))

(defun vowel-or-Np (char)
  (or (char= char #\a) (char= char #\i) (char= char #\u) (char= char #\e)
      (char= char #\o) (char= char #\A) (char= char #\I) (char= char #\U)
      (char= char #\E) (char= char #\O) (char= char #\N)))

(defun roman-style-p (string)
  "this is a simple version !"
```



```
  (or (string= string "k") (string= string "s") (string= string "sh") (string= string "t")
      (string= string "ch") (string= string "ts") (string= string "n") (string= string "h")
      (string= string "h") (string= string "m") (string= string "y") (string= string "r")
      (string= string "w") (string= string "ky") (string= string "ny") (string= string "my")
      (string= string "hy") (string= string "ry") (string= string "g") (string= string "z")
      (string= string "j") (string= string "d") (string= string "p") (string= string "py")
      (string= string "b") (string= string "dd") (string= string "pp") (string= string "tt")
      (string= string "kk") (string= string "ppy") (string= string "kky") (string= string "cch")
      (string= string "tts") (string= string "ssh") (string= string "ss") (string= string "f")
      (string= string "by") (string= string "gy")))


;;
;; Function inquire-phonetic-property
;;

(defun inquire-phonetic-properties (phonetic-symbol)
  "This function generate the phonetic properties of a letter"
  (loop for phonetic-element in *phonetic-symbols-table*
        do
    (if (string= (car phonetic-element) phonetic-symbol)
        (return (let* ((vowel? (second phonetic-element))
                       (voiced? (third phonetic-element))
                       (maner (fourth phonetic-element))
                       (place (fifth phonetic-element))
                       (palatalized? (cl:sixth phonetic-element))
                       (gemination (cl:seventh phonetic-element))
                       (inseparable (cl:eighth phonetic-element)))
                  (make-instance 'phonetic-property-flavor
                                 :symbol phonetic-symbol
                                 :vowel? vowel?
                                 :voiced? voiced?
                                 :manner maner
                                 :place place
                                 :palatalized? palatalized?
                                 :gemination gemination
                                 :inseparable inseparable))))))

;;
;; Function inquire-string
;;

(defun inquire-string (phnetic-alignment)
  "This function convert a phonetic properties alignment into string.
   Notice: This function converts 'ha -> to 'wa."
  (loop with string
        for phoneme in phnetic-alignment
        for symbol = (phonetic-property-flavor-symbol phoneme)
        do
    (loop for char being the array-element of symbol
          do
      (setq string (cl:concatenate 'string string (list char))))
        finally
          (return string)))

;;
;; Function: Break-string
;;

(defun divide-string (string position)
  "This function returns two sequences"
  (let* ((alignment (if (stringp string)
```

```
133                             (string-to-phoneme-list string)
134                             string))
135             (front-portion)
136             (back-portion))
137         (loop for index from 0 to (- position 1)
138               do
139           (setq front-portion
140                 (cl:concatenate 'string front-portion (nth index alignment))))
141         (loop for index from position to (- (length alignment) 1)
142               do
143           (setq back-portion
144                 (cl:concatenate 'string back-portion (nth index alignment))))
145         (list front-portion back-portion)))
146
147   (defun break-string (alignment positions)
148     "This function breaks input string into arbiterary sub-strings"
149     (loop with break-positions =
150             (cl:concatenate 'list positions
151                                     (list (length alignment)))
152           for start-position = 0 then end-position
153           for end-position in break-positions
154           if (not (= start-position end-position))
155             collect (list-to-string
156                       (sublist alignment start-position
157                                end-position))))
158
159   (defun sublist (list stt end)
160     (loop for index from (max 0 stt) below (min end (length list))
161           collect (nth index list)))
162
163   (defun list-to-string (list)
164     (loop with string
165           for str in list
166           do
167       (setq string (cl:concatenate 'string string str))
168           finally
169             (return string)))
170
171   (defun delete-camma (string)
172     (loop with no-camma
173           for char being the array-element of string
174           do
175       (if (not (char= char #\,))
176           (setq no-camma (cl:concatenate 'string no-camma (string char))))
177           finally
178             (return no-camma)))
179
180
181   ;;
182   ;; Paese string, convert special symbols, like 0-> oo
183   ;;
184   (defun parse-string (string)
185     "This function convert special symbols into phonemic sequence."
186     (loop with pp-list = (convert-string-to-phonetic-property-list string)
187           with result-string
188           for pp in pp-list
189           for sym = (phonetic-property-flavor-symbol pp)
190           for new-sym =     (cond
191                               ((string= sym "A") "aa")
192                               ((string= sym "I") "ii")
193                               ((string= sym "U") "uu")
194   ;                           ((string= sym "E") "ee")
195   ;                           ((string= sym "O") "ou")
196                               (t sym))
197           do
198       (setq result-string (cl:concatenate 'string result-string new-sym))
199       finally (return result-string)))
```

```
;;; -*- mode: lisp; package: zetalisp-user; base: 10 -*-
;;;
;;;

;;; Phonetic Symbole

(defvar *phonetic-symbols-table*
        '(
; Vowels
          ("a" T T  "vowel" "vowel" NIL NIL NIL NIL)  ;ame (rain)
          ("i" T T  "vowel" "vowel" NIL NIL NIL NIL)  ;ai  (love)
          ("u" T T  "vowel" "vowel" NIL NIL NIL NIL)  ;au  (meet)
          ("e" T T  "vowel" "vowel" NIL NIL NIL NIL)  ;ie (house)
          ("o" T T  "vowel" "vowel" NIL NIL NIL NIL)  ;kome (rise)
; Long Vowels
          ("A" T T  "vowel" "vowel" NIL NIL "long vowel" NIL)  ;obAsan (grandmother)
          ("I" T T  "vowel" "vowel" NIL NIL "long vowel" NIL)  ;
          ("U" T T  "vowel" "vowel" NIL NIL "long vowel" NIL)  ;
          ("E" T T  "vowel" "vowel" NIL NIL "long vowel" NIL)  ;
          ("O" T T  "vowel" "vowel" NIL NIL "long vowel" NIL)  ;
; Typical Vowel Clusters
          ("ai" t t "vowel" "vowel" NIL NIL "doubled vowel" NIL)
          ("ou" t t "vowel" "vowel" NIL NIL "doubled vowel" NIL)
          ("ei" t t "vowel" "vowel" NIL NIL "doubled vowel" NIL)
; Syllabic Nsal
          ("N" T T  "nasal" "vowel" NIL NIL NIL NIL) ; haNsiN (a baseball team)
; Stop Consonants
          ("p" NIL NIL "stop" "lip" NIL NIL NIL NIL)       ; pA (deNneN)
          ("t" NIL NIL "stop" "alveolar" NIL NIL NIL NIL) ; takeda (a family name)
          ("k" NIL NIL "stop" "velar" NIL NIL NIL NIL) ; kazuya (a first name)
          ("b" NIL T "stop" "lip" NIL NIL NIL NIL) ; basho (place)
          ("d" NIL T "stop" "alveolar" NIL NIL NIL NIL) ; dame (No, you cant)
          ("g" NIL T "stop" "velar" NIL NIL NIL NIL) ; gakkO (a school)
; Nasal Consonants
          ("m" NIL T "nasal" "lip" NIL NIL NIL NIL) ; yama (mountain)
          ("n" NIL T "nasal" "alveolar" NIL NIL NIL NIL) ; neko (cat)
          ("G" NIL T "nasal" "velar" NIL NIL NIL NIL) ; Siga (a prefecture)
; Fricativse
          ("h" NIL NIL "fricative" "glotal" NIL NIL NIL NIL) ; ha ha ha (laughing)
          ("f" NIL NIL "fricative" "lip" NIL NIL NIL NIL) ; fu fu fu (laughing)
          ("s" NIL NIL "fricative" "labial" NIL NIL NIL NIL) ; Saga (a prefecture)
          ("sh" NIL NIL "fricative" "alveolar" NIL NIL NIL NIL) ; shigoto (job)
          ("z" NIL T "fricative" "labial" NIL NIL NIL NIL) ; shizuka (a name of girl)
          ("j" NIL T "fricative" "alveolar" T NIL NIL NIL) ;jumoN (spell)
; Affricates
          ("ch" NIL NIL "affricate" "alveolar" T NIL NIL NIL) ; chi (blood)
          ("ts" NIL NIL "affricate" "alveolar" NIL NIL NIL NIL) ; shigoto (job)
; Semi-vowels
          ("y" NIL T "semi-vowel" "palate" T NIL NIL NIL) ; yama (mountain)
          ("w" NIL T "semi-vowel" "lip" NIL NIL NIL NIL) ; watashi (I)
          ("r" NIL T "semi-vowel" "flap" NIL NIL NIL NIL) ; rAmeN (soup nudle)
; Palatalized sounds
          ("py" NIL NIL "stop" "lip" T NIL NIL NIL)
          ("ky" NIL NIL "stop" "velar" T NIL NIL NIL) ; kyA (a scream sound)
          ("hy" NIL NIL "fricative" "glotal" T NIL NIL NIL NIL); uhyoo
          ("by" NIL T "stop" "lip" T NIL NIL NIL) ; byakuya (white night)
          ("dy" NIL T "stop" "alveolar" T NIL NIL NIL) ; often is nagoya diresct
          ("gy" NIL T "stop" "velar" T NIL NIL NIL) ; gyA (a scream sound)
          ("my" NIL T "nasal" "lip" T NIL NIL NIL) ; myA (cats mew)
          ("ny" NIL T "nasal" "alveolar" NIL NIL NIL NIL) ; nyA (cats mu)
          ("ry" NIL T "semi-vowel" "flap" NIL NIL NIL NIL) ; Arya (woops)
; gemination sounds
          ("dd" NIL T "stop" "alveolar" NIL T NIL NIL) ; dame (No, you cant)
          ("pp" NIL NIL "stop" "lip" NIL T NIL NIL)       ; pA (deNneN)
          ("tt" NIL NIL "stop" "alveolar" NIL T NIL NIL) ; takeda (a family name)
          ("kk" NIL NIL "stop" "velar" NIL T NIL NIL) ; kazuya (a first name)
          ("ppy" NIL NIL "stop" "lip" T T NIL NIL)
          ("kky" NIL NIL "stop" "velar" T T NIL NIL) ; kyA (a scream sound)
          ("cch" NIL NIL "affricate" "alveolar" T T NIL NIL) ; ecchi (H)
          ("tts" NIL NIL "affricate" "alveolar" NIL T NIL NIL) ; shigoto (job)
          ("ssh" NIL NIL "fricative" "alveolar" NIL T NIL NIL) ; shigoto (job)
```



```
          ("ss" NIL NIL "fricative" "labial" NIL T NIL NIL) ; Saga (a prefecture)
; Word, phrase or sentence boundary
          ("#" NIL NIL "breath-break" "breath-break" NIL NIL NIL NIL)
          (" " NIL NIL "phrase-bounary" "phrase-boundary" NIL NIL NIL NIL)))

;;
;; Describe symbol
;;

(defun describe-phonetic-symbol (symbol)
  "This function describes the attributes of the symbol"
  (if (boundp '*phonetic-symbols-table*)
      (loop for phonetic-element in *phonetic-symbols-table*
            do
        (if (string= (car phonetic-element) symbol)
            (let* ((vowel? (second phonetic-element))
                   (voiced? (third phonetic-element))
                   (maner (fourth phonetic-element))
                   (place (fifth phonetic-element))
                   (palatalized? (sixth phonetic-element))
                   (gemination (seventh phonetic-element))
                   (inseparable (eighth phonetic-element)))
              (format 'T "~% symbol: ~a~%" symbol)
              (format 'T "    ~a~%" (if vowel?
                                        "is a vowel"
                                        "is not a vowel"))
              (format 'T "    ~a~%" (if voiced?
                                        "is a voiced sound"
                                        "is not a voiced sound"))
              (format 'T "    the manner of articulation is ~a~%" maner)
              (format 'T "    the place of articulation is ~a~%" place)
              (format 'T "    ~a~%" (if palatalized?
                                        "is palatalized"
                                        "is not palatalized"))
              (format 't "    ~a~%" (if gemination "is geminate" "is not geminate"))
              (if inseparable (format 't "    consists of ~a.~%" inseparable)))))))
```

```
;;; -*- mode: art; package: art-user; base: 10. -*-
;;;
;;;
;;;  Unit-select,
;;;  A hypothesis based non-uniform synthesis segmnet optimizer
;;;
;;;  Version 2.0
;;;  November 9, 1989
;;;
;;;  Kazuya Takeda, ATR Interpreting Telephony Research Laborotories.
;;;
;;;  The major revision from version 0.0 is:
;;;
;;;    Strategy of optimal unit selection at the last stage,
;;;    after calculating needed costs, was changed.
;;;
;;;  Version 2.0
;;;
;;;     The lattice of unit candidates is obtained incrementally
;;;
;;;  December 4, If consecutive phrases are not separated by pause
;;;    intra-phrase phonemic context should be taken account into
;;;    selection.
;;;
;;;
;;; Copy Right ATR Interpreting Telephony Research Laboratories
;;;

;; No merging
(def-viewpoint-levels (V merging nil))

;; Set switch "varbose" to Get information
(defrelation display-mode (?switch))
(deffacts display-switch-on (display-mode varbose))

;; execution controller
(defglobal ?*sal-just-after* =          -2
           ?*sal-after* =               -5
           ?*sal-just-before* =          2
           ?*sal-before* =               5
           ?*sal-initialize* =           0
           ?*sal-read-text* =            0
           ?*sal-make-lattice* =       -20
           ?*sal-search-entry* =       -40
           ?*sal-costing* =            -60
           ?*sal-optimal-units* =     -160
           ?*sal-decide-opt-seg* =    -170
           ?*sal-get-seg-info* =      -180
           ?*sal-decide-opt-seq* =    -190
           ?*sal-open-files* =        -300
           ?*sal-write-out* =         -310)
;;
;;  Defined relations
;;

;; Synthesis control file's name

(defrelation file-name-header (header)
  "The file-name-header is sotred in  fact.")

;; The contents of xxx.PH file

(defrelation text ((?phrase-id
                    ?text
                    ?mora-length
                    ?accent-pattern
                    ?pause-condition
                    ?Fo-condition
                    ?inf))
  "This relation describes output text, should be read from .PH file
   This form express the input text which should be synthesized.")
```



```
;;
;; If there is a fact debug, no file write out
;; will be done

(defrelation debug)

;;
;;
;; Start Processing

(defrule read-infile
        "If no file name header is defined, read from terminal"
  (declare (salience ?*sal-read-text*))
  (not (exists (file-name-header ?)))
  =>
  (printout t t "Input file name's header: ")
  (assert (file-name-header = (read))))


;;
;; Read input Informations
;;

(defrule start-processing
        "This is the trigger of all processing ."
  (declare (salience (+ ?*sal-just-after* ?*sal-read-text*)))
  (file-name-header ?file-name-header)
  (not (exists (text (?phrase-id $ ?))))
  =>
  ;;
  ;; Read input text
  ;;
  (bind ?texts
        (read-org-file (concatenate 'string ?file-name-header ".ph")))
  (for x in ?texts do
       (assert (text = (seq$ x))))
  ;;
  ;; Read target pitch-contour
  ;;
  (bind ?pitchs
        (read-org-file (concatenate 'string ?file-name-header ".pp")))
  (for x in ?pitchs do
       (assert (target-pitch-contour = (seq$ x)))))

;;
;; Load Phonetic Property of the Input Text
;;

(defrule phonetic-property-of-input-text
        "This rule load phonetic-property of input text."
  (declare (salience ?*sal-read-text*))
  (text (?phrase-id ?text-string $ ?))
  =>
  (assert
    (pp-list ?phrase-id
             =(convert-string-to-phonetic-property-list ?text-string))))

;;
;;      END INITIALIZATION
;;


;;
;;
;; Expand Input String into SubString Sequneces
;;
;;

(defrule generates-substring-sequence
        "This rule generates some substring-sequences,
         by deviding a substring preciser."

```

```
143     (declare (salience (+ ?*sal-before* ?*sal-make-lattice*)))
144     (not (exists (this-is-optimal)))
145                ; To avoid expanding "pause" segment which possiblly
146                ; inserted at the end.
147
148
149     (viewpoint ?root
150                (not (exists (hypothesized-sequence $ ?)))
151          ?x <- (not-exists ($ ?unit-seq ?criterion-no))
152                (not (exists (expanded-sequence ?unit-seq $ ?)))
153   ;            (not (exists (not-exists ($ ? ?cr-no
154   ;                                     & \: (< ?cr-no ?criterion-no))))))
155                )
156     (test (< ?criterion-no 6))
157
158     (viewpoint ?v
159                (sequence ?score ($ ?pre ($ ?unit-seq ?) $ ?fol))
160                (speech-unit ?order ($ ?unit-seq ?))
161                (not (exists (expanded ?order))))
162
163
164     =>
165     (bind ?unit (list*$ ?unit-seq))
166
167     (if (primitivep ?criterion-no ?unit)
168         then
169         (printout t t "Expanding unit " ?unit " into sub units.")
170         (printout t t " ---------------------------------------- ")
171         (printout t t " by the " (1+ ?criterion-no) "th criterion.")
172
173         (bind ?new-units (get-new-unit ?criterion-no ?unit))
174
175         (if ?new-units
176             then
177             (at ?v (assert (expanded ?order)))
178             (at ?root (assert (expanded-sequence ?unit-seq ?new-units)))
179             (for units in ?new-units
180                  do
181                  (bind ?unit-list (list*$ units))
182                  (printout t t (mapcar 'car (second ?unit-list))
183                            "(score " (car ?unit-list) ")")
184                  (bind ?new-score (+ (car ?unit-list) ?score))
185                  (bind ?un (second ?unit-list))
186                  (bind ?u (seq$ ?un))
187
188                  (at ?root
189                      (assert
190                        (substring-sequence
191                          (?new-score ($ ?pre $ ?u $ ?fol))))))
192             else
193             (bind ?new-unit-prop
194                   (reverse (cons (1+ ?criterion-no)
195                                  (reverse ?unit))))
196             (at ?root
197                 (assert (not-exists ?new-unit-prop)))
198             (printout t t)
199             (printout t t " ---------------------------------------- "))))
200
201   ;;
202   ;;
203   ;; generate preciser sequence when it has been devided before hand
204   ;;
205   ;;
206   (defrule generates-existsing-substring-sequence
207            "This rule generates some substring-sequences,
208             by deviding a substring preciser."
209
210     (declare (salience (+ ?*sal-before* ?*sal-make-lattice*)))
211     (not (exists (this-is-optimal)))
212
213     (viewpoint ?root
```



```
214                (not (exists (hypothesized-sequence $ ?)))
215          ?x <- (not-exists ($ ?unit-seq ?criterion-no))
216                (expanded-sequence ?unit-seq ?units)
217   ;            (not (exists (not-exists ($ ? ?cr-no
218   ;                                     & \: (< ?cr-no ?criterion-no))))))
219                )
220
221     (viewpoint ?v
222                (sequence ?score ($ ?pre ($ ?unit-seq ?) $ ?fol))
223                (speech-unit ?order ($ ?unit-seq ?))
224                (not (exists (expanded ?order))))
225
226     =>
227
228
229     (bind ?new-units (list*$ ?units))
230
231     (at ?v (assert (expanded ?order)))
232     (for un in ?new-units
233          do
234          (bind ?unit-list (list*$ un))
235          (bind ?new-score (+ (car ?unit-list) ?score))
236          (bind ?un (second ?unit-list))
237          (bind ?u (seq$ ?un))
238
239          (at ?root
240              (assert
241                (substring-sequence
242                  (?new-score ($ ?pre $ ?u $ ?fol))))))
243
244   ;;
245   ;;
246   ;; First of all, the original input text is hypothesized
247   ;; to be a synthesis unit
248   ;;
249   ;;
250
251   (defrule initial-try
252            "This rule try to hypothesize the original
253             text as a synthesis unit."
254
255     (declare (salience ?*sal-initialize*))
256
257     (text (?phrase-id ?text $ ?))
258     (pp-list ?phrase-id ?pp)
259
260     =>
261
262     (printout t t "initially the original text is the only one.")
263
264     (bind ?primitive-unit
265           (list ?text 0.0 (length (list$ ?pp)) "initial" "final" 0))
266
267     (assert (substring-sequence (0 (?primitive-unit)))))
268
269   ;;
270   ;;
271   ;; Load Sequence Candidates
272   ;;
273   ;;
274
275   (defrule load-sequence-candidate
276            "this rule generates sequence candidates."
277     (declare (salience ?*sal-make-lattice*))
278     (not (exists (this-is-optimal)))
279
280     ?x <- (substring-sequence (?score ?sequence-inf))
281     (not (exists (substring-sequence
282                    (?score-b  & \: (< ?score-b ?score) $ ?))))
283     ;
284     ; Pick up the lowest cost candidate
```

```
285     ;
286
287     (or (not (exists (hypothesized-sequence $ ?)))
288         ;
289         ; No sequence is hypothesized
290         ;
291         (exists (hypothesized-sequence ?score-b & \: (= ?score-b ?score) $ ?)))
292         ;
293         ; Previously hypothesized sequence's cost is the same with this one.
294         ;
295     =>
296     (printout t t " ==== Assert ====> ")
297     (printout t (mapcar 'car (list*$ ?sequence-inf)))
298     (retract ?x)
299     (at ?root
300         (assert (hypothesized-sequence ?score ?sequence-inf)
301                 (substring-sequence-memory 0 0 (?score ?sequence-inf)))))
302
303   ;;
304   ;;  Some facts which describes the hypothesized unit sequences
305   ;;  have been asserted by this rule.
306   ;;
307
308   ;; Search Units -----------
309   ;;
310   ;;
311   (defrule hypothesize-unit-string
312            "this rule memorize elemental unit strings for
313             hypothesized unit-sequence."
314     (declare (salience ?*sal-search-entry*))
315     (hypothesized-sequence ?score ?sequence-inf)
316
317     =>
318
319     (for index from 0
320          as unit in$ ?sequence-inf
321          do
322          (assert (unit-memory = index = (list$ unit))))
323     ;
324     ; For information write out
325     ; hyphothesized units are written as facts
326     ;
327     (sprout
328       (assert (each-score))
329       (assert (would-be-a-candidate))
330       (assert (sequence ?score ?sequence-inf))
331       (for index from 0
332            as unit in$ ?sequence-inf
333            do
334            (assert (speech-unit = index = unit)))))
335
336   ;;
337   ;;
338   ;; Examin if the unit is included in the database
339   ;;
340
341   (defrule examine-unit-existence
342            "This rule searches the SUE dictionary for unit string."
343
344     (declare (salience ?*sal-search-entry*))
345
346     (viewpoint ?v (speech-unit ?order (?us ?ust ?uen ?usc ?uec ?criterion)))
347     (test (string-not-equal ?us "pau"))
348     (viewpoint ?v (not (exists (can-not-completed))))
349     (viewpoint ?root (not (exists (searched (?us ?ust ?uen ?usc ?uec)))))
350   ;  (viewpoint ?root (not (exists (found-unit (?us ?ust ?uen ?usc ?uen)))))
351   ;  (viewpoint ?root (not (exists (not-exists (?us ?ust ?uen ?usc ?uec ?))))))
352
353     =>
354
355     (bind ?list-unit (list ?us ?ust ?uen ?usc ?uec))
```



```
356
357     (at ?root (assert (searched (?us ?ust ?uen ?usc ?uec))))
358     (bind ?unit-candidates (examine-dictionary ?list-unit))
359          ;
360          ; Execute Search Command
361          ;
362     (if ?unit-candidates
363         then
364         (printout t t (car ?list-unit ) " Found  -----")
365         (at ?root (assert (found-unit ?list-unit)))
366         (at ?v (assert (found ?order)))
367         else
368         (printout t t (car ?list-unit)  " Not found --")
369         (at ?root (assert (not-exists (?us ?ust ?uen ?usc ?uec ?criterion))))))
370
371
372   (defrule fill-up-elemental-unit
373            "this rule assert a fact found, if the existence of the
374             unit can be comfirmed."
375     (declare (salience ?*sal-search-entry*))
376     (viewpoint ?v (speech-unit ?order (?us ?ust ?uen ?usc ?uec ?criterion)))
377     (viewpoint ?v (not (exists (can-not-completed))))
378     (viewpoint ?root (found-unit (?us ?ust ?uen ?usc ?uec)))
379
380     =>
381
382     (at ?v (assert (found ?order))))
383
384   (defrule delete-hypothesese-if-luck-of-unit
385            "This rule delete availabel hypotheses
386             if a component unit is not exits."
387     (declare (salience (+ ?*sal-search-entry* ?*sal-just-before*)))
388     (viewpoint ?root (not-exists ?unit))
389     (viewpoint ?v (speech-unit ?order ?unit)
390                (not (exists (can-not-completed))))
391     =>
392     (at ?v (assert (can-not-completed))))
393
394   (defrule delete-hypothesis-viewpoint
395            "this rule inactivate viewpoint by retracting
396             the relation wold-be-a-candidate."
397     (declare (salience 0))
398     (viewpoint ?v (can-not-completed))
399     (viewpoint ?v ?x <- (would-be-a-candidate))
400     =>
401     (retract ?x))
402
403   (defrule delete-sequence-candidate-if-can-not-complete
404            "The sequence one of elemental unit of which
405             can not be completed, will be removed from
406             hyotheses."
407     (declare (salience (+ ?*sal-just-after* ?*sal-search-entry*)))
408     ?x <- (hypothesized-sequence ?score ?sequence-inf)
409     (viewpoint ?v (sequence ?score ?sequence-inf))
410
411     (viewpoint ?v (not (forall (speech-unit ?no $ ?)
412                                (found ?no))))
413
414     =>
415
416     (printout t t "                    ")
417     (printout t (mapcar 'car (list*$ ?sequence-inf)) " == retracted ==> ")
418     (at ?v (assert (can-not-completed)))
419     (retract ?x))
420   ;
421   ; --------- If no candidate is obtained till now ------------------------
422   ;;
423   ;
424   ;(defrule check-not-existing-unit
425   ;        "this rule check, if the unit is
426   ;          primitive or not"
```

```
427   ;   (declare (salience (+ ?*sal-just-after* ?*sal-search-entry*)))
428   ;   (viewpoint ?root ?x <- (not-exists ?unit))
429   ;   (viewpoint ?root (current-criterion ?no-a))
430   ;   (viewpoint ?root (not (exists (current-criterion ?no-b & \: (< ?no-a ?no-b)))))
431   ;
432   ;   =>
433   ;
434   ;   (bind ?list-unit (list*$ ?unit))
435   ;   (retract ?x)
436   ;   (bind ?criterion-no ?no-a)
437   ;   (if (primitivep ?criterion-no ?list-unit)
438   ;       then
439   ;       (at ?root (assert (not-exists ?unit primitive)))
440   ;       else
441   ;       (printout t t ?unit " is not primitive>")))
442   ;
443   ;(defrule count-up-criterion
444   ;         "This rule count up criterion for more precise
445   ;           search."
446   ;   (declare (salience (+ ?*sal-just-after* ?*sal-search-entry*)))
447   ;   (not (exists (would-be-a-candidate)))
448   ;   (not (exists (hypothesized-sequence ? ?)))
449   ;   (not (exists (not-exists ?)))
450   ;
451   ;   =>
452   ;
453   ;   (printout t t "count up criterion !!")
454   ;   (at ?root (assert (current-criterion = (count-up-criterion)))))
455   ;
456   ;(defrule retry-with-samller-unit
457   ;         "This rule redivide the unit the speech segment corresponding
458   ;           to which can not be found."
459   ;
460   ;   (declare (salience ?*sal-search-entry*))
461   ;   (not (exists (would-be-a-candidate)))
462   ;   (not (exists (hypothesized-sequence ? ?)))
463   ;   (pp-list ?phrase-id ?pplist)
464   ;
465   ;   (viewpoint ?root (current-criterion ?no-a))
466   ;   (viewpoint ?root (not (exists (current-criterion ?no-b
467   ;                                                    & \: (< ?no-a ?no-b)))))
468   ;   ; This rule will be fired after all hypotheses are
469   ;   ; examined
470   ;
471   ;   (viewpoint ?root ?x <- (not-exists ?no-unit primitive))
472   ;   (viewpoint ?v1 (speech-unit ?index ?no-unit))
473   ;   (viewpoint ?v1 ?y <- (sequence ?score ?sequence-inf))
474   ;
475   ;   =>
476   ;
477   ;   (bind ?new-unit-sequence
478   ;         (make-smaller-unit
479   ;           (list*$ ?sequence-inf)
480   ;           ?index
481   ;           ?score
482   ;           (list$ ?pplist)
483   ;           ?no-a))
484   ;   (printout t t "target unit: " ?no-unit)
485   ;   (printout t t "current criterion: " (get-current-criterion))
486   ;   (printout t t "obtained sequence: " ?new-unit-sequence)
487   ;
488   ;   (if ?new-unit-sequence
489   ;       then
490   ;       (printout t t (mapcar 'car (list*$ (second ?new-unit-sequence))))
491   ;       (retract ?y)
492   ;       (at ?root (assert (substring-sequence 0 ?new-unit-sequence)))
493   ;       (at ?root (assert (substring-sequence-memory 0 ?new-unit-sequence)))))
494   ;       else
495   ;       (retract ?x)))
496   ;       (at ?root (assert (not-exists ?no-unit primitive)))))
497   ;
```



```
498   ;; --------- Select Optimal Unit !!
499
500   (defrule obtaine-speech-unit-candidates
501            "This rule searches sue for actual unit candidates
502             and sort them by their context similarity."
503     (declare (salience ?*sal-costing*))
504     (viewpoint ?root (hypothesized-sequence ?score ?sequence-inf))
505     (viewpoint ?v (sequence ?score ?sequence-inf))
506     (viewpoint ?v (speech-unit ?index ?unit))
507     (viewpoint ?root
508                ; To avoid extra search
509                (not (exists (examined-candidates ?unit $ ?)))
510                (pp-list ?phrase-id ?pp-list)
511                ; In order to examine intra-phrase context
512                (phrase-left-context ?left-context)
513                (phrase-right-context ?right-context))
514     =>
515     (bind ?left-context-pp (car (get-pp-from-str ?left-context)))
516     (bind ?right-context-pp (car (get-pp-from-str ?right-context)))
517     (bind ?list-unit (list$ ?unit))
518     (bind ?list-pp (list$ ?pp-list))
519     (printout t t "<<< Examining Segment Property " (car ?list-unit) " >>>")
520     ; To avoid multiple serach for a entry
521
522     (bind ?unit-candidates (search-dictionary ?list-unit
523                                               ?list-pp
524                                               ?left-context-pp
525                                               ?right-context-pp))
526
527     (at ?root (assert (examined-candidates ?unit ?unit-candidates)))
528     (for no from 1 to 30
529          as unit in ?unit-candidates
530          do
531          (at ?v (assert (candidate-unit ?index = no = unit)))))
532
533   (defrule obtaine-speech-unit-candidates-from-memory
534            "This rule searches sue for actual unit candidates
535             and sort them by their context similarity."
536     (declare (salience ?*sal-costing*))
537     (viewpoint ?root (hypothesized-sequence ?score ?sequence-inf))
538     (viewpoint ?v (sequence ?score ?sequence-inf))
539     (viewpoint ?v (speech-unit ?index ?unit))
540     (viewpoint ?root
541                ; The property of the unit was already examined
542                (examined-candidates ?unit ?unit-candidates))
543     =>
544
545     (for no from 1 to 30
546          as unit in$ ?unit-candidates
547          do
548          (at ?v (assert (candidate-unit ?index = no = unit)))))
549
550
551
552   (defrule obtaine-pitch-similarity
553            "This rule compute similarity of pitch."
554     (declare (salience ?*sal-costing*))
555     (viewpoint ?v (candidate-unit ?index ?no ?candidate-unit-property))
556     (viewpoint ?v (speech-unit ?index ?speech-unit-property))
557
558     (viewpoint ?root (target-pitch-contour ?pitch-pattern))
559     =>
560     (bind ?pitch-cost (pitch-cost
561                          (list$ ?pitch-pattern)
562                          (list$ ?speech-unit-property)
563                          (list$ ?candidate-unit-property)))
564     ; For information write out
565     (bind ?list-candidate-prop (list$ ?candidate-unit-property))
566     (bind ?cost-list
567           (list (seventh ?list-candidate-prop)
568                 ; This is context similarity
```

```
569                ?pitch-cost
570                ; This is pitch similarity
571                (+ (fifth ?list-candidate-prop)
572                 (sixth ?list-candidate-prop))))
573                ; This is ease of extraction
574
575      (assert (segment-memory
576                ?index ?cost-list
577                ( = (car ?list-candidate-prop)
578                  = (second ?list-candidate-prop)
579                  = (third ?list-candidate-prop)
580                  = (fourth ?list-candidate-prop))))
581
582
583      (at ?v (assert (candidate-with-cost ?index
584                                          = (get-total-cost ?cost-list)
585                                          ?cost-list
586                                          ?list-candidate-prop))))
587
588
589    (defrule decide-optimal-candidate
590             "At last, by this rule, the optimal units for
591              the correspoinding sequence are decided."
592      (declare (salience ?*sal-optimal-units*))
593      (viewpoint ?v1
594                 (candidate-with-cost ?index
595                                      ?cost-a
596                                      ?cost-list
597                                      (?string ?stt ?end ?word $ ?other-inf))
598                 ;
599                 ; I fixed awful bug ! at here.
600                 ;
601                 (not (exists
602                        (candidate-with-cost ?index
603                                             ?cost-b & \: (< ?cost-b ?cost-a)
604                                             ?cost-list-b
605                                             ?))))
606
607      (viewpoint ?v1 (not (exists (optimal-unit ?index $?))))
608      (viewpoint ?v1 ?x <- (each-score $ ?scores))
609      =>
610      (bind ?score-list (list$ ?scores))
611      (bind ?score-list (cons ?cost-a ?score-list))
612      (at ?v1 (assert (each-score $ ?score-list)))
613      (retract ?x)
614      (at ?v1 (assert (optimal-unit ?index ?string ?stt ?end ?word))))
615
616    (defrule mean-cost-of-unit
617             "calculate mean cost of elemental units."
618      (declare (salience (+ ?*sal-optimal-units* ?*sal-after*)))
619      (viewpoint ?v (each-score $ ?score-list))
620      =>
621      (bind ?mean-score (mean (list$ ?score-list)))
622      (if (zerop ?mean-score)
623          then
624          else
625          (at ?v (assert (mean-unit-score ?mean-score)))))
626    ;;
627    ;; For debug
628    ;;
629    (defrule debug-for-dos
630             "This is for debug."
631      (declare (salience ?*sal-decide-opt-seq*))
632      (viewpoint ?v1 (mean-unit-score ?score-a))
633      (viewpoint ?v2
634      (not (exists   (mean-unit-score ?score-b & \: (< ?score-b ?score-a)))))
635      =>
636      (printout t t "this is best." ?score-a))
637
638
639    (defrule decide-optimal-sequence
```



```
640              "Finally, the optimal sequence is decided
641                according to the suitabilities of elemental
642                units and suitability of the sequence itself."
643      (declare (salience ?*sal-decide-opt-seq*))
644
645      (viewpoint ?v1 (mean-unit-score ?score-a)
646                     (sequence ? ?sequence-inf))
647      (viewpoint ?v2
648                 (not (exists
649                        (mean-unit-score
650                          ?score-b & \: (< ?score-b ?score-a)))))
651      (viewpoint ?root ?x <- (substring-sequence-memory
652                               ?order
653                               0
654                               (?c ?sequence-inf)))
655
656      =>
657      (retract ?x)
658      (at ?v1 (assert (this-is-optimal)))
659      (at ?root
660          (assert (substring-sequence-memory ?order 1 (?c ?sequence-inf)))))
661    ;  (printout t t ?sequence-inf))
662    ;;
663    ;; Inserteing pause segment is done by run-unit-select.
664    ;;
665    ;(defrule insert-pause-segment
666    ;        "this rule insert pause as a unit segment."
667    ;  (declare (salience ?*sal-decide-opt-seq*))
668    ;
669    ;  (viewpoint ?root (text (?pid ?text ? ? ~0 $ ?)))
670    ;  (viewpoint ?v (this-is-optimal)
671    ;                (optimal-unit ?odr-a ~"pau" $ ?)
672    ;                (not (exists (optimal-unit ?odr-b & \: (> ?odr-b ?odr-a) $ ?))))
673    ;
674    ;  =>
675    ;
676    ;  (at ?v (assert
677    ;           (speech-unit = (+ ?odr-a 1) ("pau" 0 0 "pause" "pause" 0))
678    ;           (optimal-unit = (+ ?odr-a 1) "pau" 0 0 0))))
679
680    ; ------------------------------------------------------------------------
681
682    ;;;
683    ;;;  TIMER
684    ;;;
685
686    (defrelation start-time (?time))
687    (defrule start-time
688             "record time"
689      (declare (salience *maximum-salience*))
690      =>
691      (bind ?start-time (time))
692      (assert (start-time ?start-time))
693      (printout t t "start-time: " ?start-time))
694
695    (defrule end-time
696             "record time"
697      (declare (salience *minimum-salience*))
698      (start-time ?stt-time)
699      =>
700      (bind ?end-time (time))
701      (bind ?tooktime (/ (float (- ?end-time ?stt-time)) 3600.0))
702      (printout t t "run-time: " ?tooktime "(min)"))
703    ;
704    ;;; ------------------------------------------------------------------
705    ;; For demonstration output
706
707    (defrule open-information-files
708             "At last by the results are written onto the file."
709      (declare (salience ?*sal-open-files*))
710      (viewpoint ?root (file-name-header ?file-name-header))
```

```
711     (viewpoint ?root (not (exists (debug))))
712     =>
713     (printout t t "open files for information memory.")
714     (bind ?result-file (concatenate 'string ?file-name-header ".oett"))
715     (bind ?unit-file (concatenate 'string ?file-name-header ".unt"))
716     (bind ?sequence-file (concatenate 'string ?file-name-header ".seq"))
717     (bind ?segment-file (concatenate 'string ?file-name-header ".seg"))
718     (assert (result-file ?result-file))
719     (assert (unit-file ?unit-file))
720     (assert (sequence-file ?sequence-file))
721     (assert (segment-file ?segment-file)))
722
723   (defrule write-out-results
724           "This rule writes out sequences for experiments
725            and demonstration."
726     (declare (salience ?*sal-write-out*))
727     (viewpoint ?v (this-is-optimal))
728     (viewpoint ?v (sequence ?index ?seq))
729     =>
730     (printout t t  "Obtained sequence " (mapcar 'car (list*$ ?seq))))
731
732
733   (defrule write-out-results-into-file
734           "This rule writes out sequences for experiments
735            and demonstration."
736     (declare (salience ?*sal-write-out*))
737     (viewpoint ?root (result-file ?file))
738     (viewpoint ?root (text (?phrase-id $ ?)))
739   ;; The final result is in the viewpoint in which the fact
740   ;;  "will-be-used" is.
741     (viewpoint ?v (this-is-optimal))
742     (viewpoint ?v (speech-unit ?index (? ?stt ?end $ ?)))
743     (viewpoint ?v (optimal-unit ?index ? ?stt-word ?end-word ?word))
744     =>
745     (with-open-file
746       (stream ?file #l:direction #l:output
747                     #l:if-does-not-exist #l:create
748                     #l:if-exists #l:append)
749       ;
750       ; [Takeda Aug 27]
751       ; Written objects should be lisp objects
752       ;
753       (bind ?stt-in-phoneme (convert-to-ph-count-stt
754                               ?stt ?stt-word))
755       (bind ?end-in-phoneme (convert-to-ph-count-end
756                               ?end ?end-word))
757       (bind ?the-string (get-the-string ?word
758                                         ?stt-in-phoneme
759                                         ?end-in-phoneme))
760       (bind ?whole-word (get-word ?word))
761       (printout stream t
762                 "(" ?phrase-id
763                 " " ?index
764                 " " ?word " "
765                 ?stt-in-phoneme " "
766                 ?end-in-phoneme
767                 " \"" ?the-string "\""
768                 " \"" ?whole-word "\")")))
769
770   (defrule write-out-units
771           "This rule writes out synthesis unit cadidates."
772     (declare (salience ?*sal-write-out*))
773     (text (?phrase-id $ ?))
774     (viewpoint ?root (unit-file ?file))
775     (viewpoint ?root (unit-memory ? ?unit-string-prop))
776     =>
777     (bind ?unit-property (list ?phrase-id (list$ ?unit-string-prop)))
778     (with-open-file
779       (stream ?file #l:direction #l:output
780                     #l:if-does-not-exist #l:create
781                     #l:if-exists #l:append)
```



```
782       (printout stream t ?unit-property)))
783
784   (defrule write-out-sequences
785           "This rule writes out sequences for
786            experiments and demonstration."
787     (declare (salience ?*sal-write-out*))
788     (text (?phrase-id $ ?))
789     (viewpoint ?root (sequence-file ?file))
790     (viewpoint ?v (this-is-optimal))
791     (viewpoint ?v (sequence ?cost $?))
792     (viewpoint ?root (substring-sequence-memory ?order ?flag
793                                                 (?cost-b & \: (<= ?cost-b ?cost)
794                                                               $ ?seqs)))
795     =>
796     (bind ?seq-info (list ?phrase-id ?cost ?flag (list*$ ?seqs)))
797     (with-open-file
798       (stream ?file #l:direction #l:output
799                     #l:if-does-not-exist #l:create
800                     #l:if-exists #l:append)
801     (printout stream t ?seq-info)))
802   ;  (printout t t ?seq-info))
803
804   ;
805   (defrule wirte-out-speech-segments
806           "By this rule we can obtaine the informations for
807            the speech segments."
808     (declare (salience ?*sal-write-out*))
809     (text (?phrase-id $ ?))
810     (viewpoint ?root (segment-memory $ ?segment-prop))
811     (viewpoint ?root (segment-file ?file))
812     =>
813     (with-open-file
814       (stream ?file #l:direction #l:output
815                     #l:if-does-not-exist #l:create
816                     #l:if-exists #l:append)
817     (printout stream t (cons ?phrase-id (list*$ ?segment-prop)))))
818
819
820
```