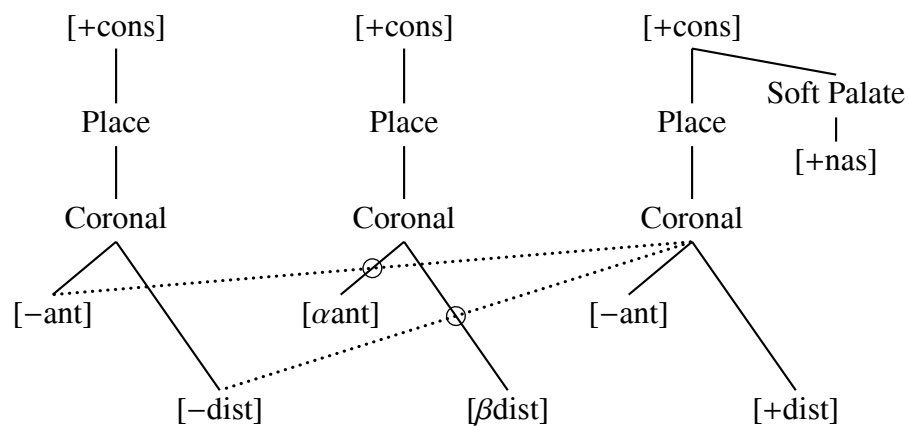


pst-asr

Tex macros for typesetting
autosegmental representations



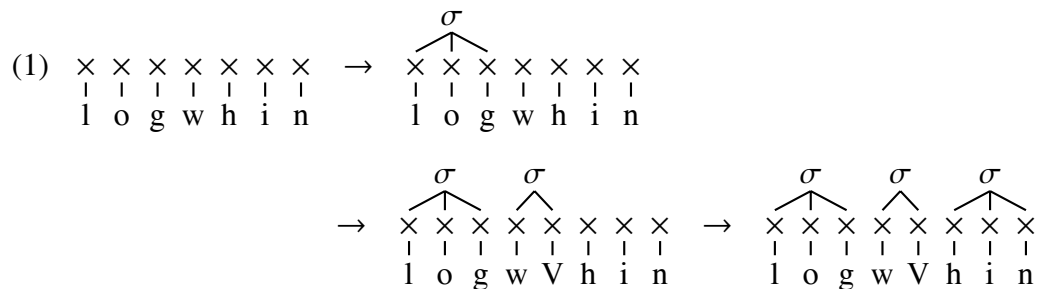
User's Guide

John Frampton
j.frampton@neu.edu

20 April 2006
Version 1.0

1. What *pst-asr* does

Typesetting autosegmental representations poses difficult problems. While no single representation poses an insurmountable problem, it is difficult to typeset autosegmental representations with sufficient ease to make them a regular part of the discussion. Halle and Idsardi do it in their paper on stress, but although their paper is full of multi-tiered representations, the relation between tiers in their stress analysis is particularly simple. Goldsmith has many illustrations in his book *Autosegmental and Metrical Phonology*. Consider the following derivation of syllable structure, adapted from Goldsmith's discussion (p. 119) of Yokuts syllabification. V represents an epenthetic vowel.



Without software which is specialized to construct autosegmental displays, creating (1) requires sufficient work that a linguist has to think twice about using this display in a handout or paper.

pst-asr makes it fairly easy.

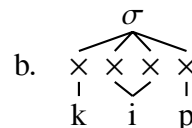
```
\ex
\asr logwhin\endasr
\to
\asr \3logwhin\endasr
\bigskip \hfill \to
\asr \3log\2wWhin\endasr
\to
\asr \3log\2wV\3hin\endasr
\kern1em
\xe
```

(Users will have to supply their own versions of `\ex...` `\xe` and `\to`.)

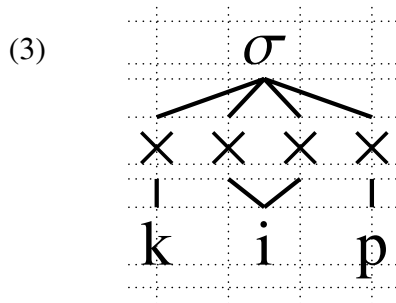
1.1. The components of a software package

What is required to allow code like (2a) to generate the display (2b)?

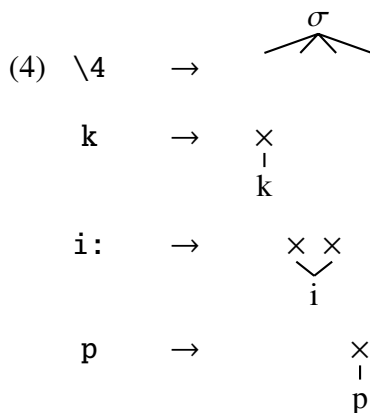
(2) a. `\asr \4ki:p\endasr`



Grid. A grid of some sort is necessary in order to provide a framework in which to place characters and establish reference points for drawing association lines.



Parser. A parser is needed in order to read the sequence `\4ki:p` from left to right, breaking it down into its component parts and translating them into the corresponding pieces of the display.



Extendability and Adjustability. The mechanism must be extendable, so that new tiers in addition to the phoneme, timing slot, and syllable tiers can be defined and means provided for placing elements on tiers and associating elements on different tiers. All of the many dimensions and choices that need to be made to specify a tiered structure must be easily adjustable and defaults provided so that a reasonable “first draft” is produced with minimal settings.

2. The grid

2.1. Horizontal dimensions

Parameters: `xgap`, `unitxgap`

Dimension register: `\ASRxgap`

The horizontal grid is set by the parameter `xgap`. `\psset{xgap=dimension}` sets a dimension register `\ASRxgap`. The dimension can be specified without explicit dimension units, in which case it is understood as being specified in `psxunits` (i.e. a multiple of `\psxunit`). If the boolean parameter `unitxgap` is set to true, then `\psset{xgap=dimension}` has the secondary effect `\psset{xunit=dimension}`. This makes it easy to specify horizontal positions,

because all the *pst-asr* macros understand numerical x-dimensions (i.e. ones without explicit dimensions) as being measured in psxunits. The default setting is `unitxgap=true`.

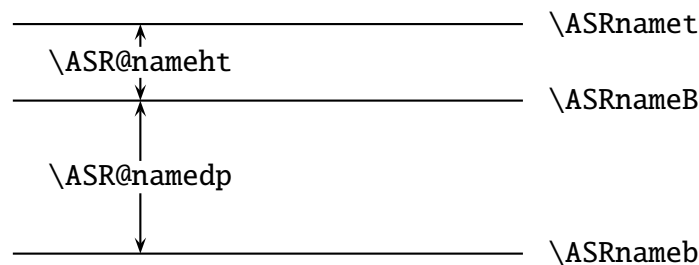
2.2. Vertical dimensions

Macro: `\newtier` (e.g. `\newtier{whatever}`)

Parameters: `whateverB`, `whateverht`, `whateverdp`, `whatever`

Macros: `\ASRwhateverB`, `\ASRwhateverht`, `\ASRwhateverb`

Executing `\newtier{name}` establishes four parameters: `nameB`, `nameht`, `namedp`, and `name`. Setting the parameters `nameB`, `nameht`, and `namedp` (assigning them dimensions) establishes 3 y-levels and 2 y-differences, all recorded in control sequences, as shown below.



Tiers named *ts* (timing tier), *sy* (syllable tier), and *ph* (phoneme tier) are defined in *pst-asr*. Users can define whatever other tiers they need. `\newtier` accepts a list of names as well as a single name.

The parameter `name` does not have an independent role, but gives a convenient way to set the other parameters. The *sy*-tier will be used for illustrative purposes. The parameters are `syB`, `syt`, `syb`, and `sy`. The most common way to set the *sy*-levels is by executing

$$\backslash\text{psset}\{\text{sy} = \text{baseline_specification height/depth_specification}\}$$

There are several options available for setting the baseline level.

$$(5) \text{ baseline_specification} \rightarrow \begin{pmatrix} y\text{-level} \\ (tier_name) \quad y\text{-offset} \\ * \end{pmatrix}$$

The baseline level can be specified directly, or as an offset to an already established tier baseline, or left unchanged (the `*` option). So, for example,

$$\backslash\text{psset}\{\text{sy}=(\text{ts}) \ 1 \ 3\text{ex} \ 1.5\text{ex}\}$$

will set the *sy*-baseline to be 1 psyunit above the *ts*-baseline. All dimensions can either be purely numerical, in which case they will be interpreted as being given in psyunits, or with explicit Tex dimensions (pt, ex, em, ...).

The options for setting the tier height and depth are:

$$(6) \text{ height/depth_specification} \rightarrow \left(\begin{array}{l} \text{height} \quad \text{depth} \\ \text{height/depth} \\ (\text{stuff}) \quad \text{extra_height} \quad \text{extra_depth} \\ (\text{stuff}) \quad \text{extra_height/depth} \\ (\text{stuff}) \end{array} \right)$$

If two dimensions are given, they are taken to be the height and depth of the tier. If a single dimension is given, it is taken to be both the height and depth of the tier. If there is a (*stuff*) term, the stuff is put in an hbox and the height and depth of the hbox, augmented by the given extra height and extra depth, are taken to be the height and depth of the tier. Two characters usually suffice for “stuff”; the tallest and deepest characters that appear on the tier. If no extra height or depth is specified, the current value of the parameter `extragap` is used to augment both the height and depth of the hbox to determine the height and depth of the tier. The core tiers are established in *pst-asr* by:

```
\newtier{ts,ph,sy}
\newpsstyle{medsylys}{unitxgap=true,xgap=2.5ex,extragap=.5ex,
  ts=0 ($\times$),ph=-3.5ex (pf),sy=3.5ex ($\sigma$)}
\psset{style=medsylys}
```

Here and throughout this User’s Guide, full width boxed code is a quotation from *pst-asr*.

2.3. Incremental modification

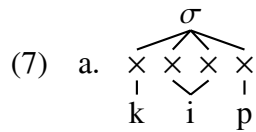
Small adjustments (in the *sy*-tier levels, for example) are often best handled by the parameters `syB`, `syht`, and `sydp`. These parameters (not the parameter `sy`) accept incremental modification, signalled by the prefix `!` on the value. So, for example, `\psset{syB=!2pt}` will raise the *sy*-baseline by 2pt. `\ASRsyB`, `\ASRsyht`, and `\ASRsydp` are all modified accordingly. The parameter `xgap` also accepts incremental modification.

3. The hbox that representations are built in

Macros: `\dbox`, `\enddbox`, `\dput`

`\dbox` initiates the construction of an hbox. It also initializes some dimension registers which are used to keep track of the bounding box of material typeset inside the box by the `\dput` macro. `\enddbox` closes up the box and uses those dimension to set the size of the resulting box so that it bounds the material inside the box which was typeset by `\dput`. The macro `\dput` is just like the PSTricks macro `\rput`, but adds the updating of the dimension registers which are used to set the size of the `dbox`.

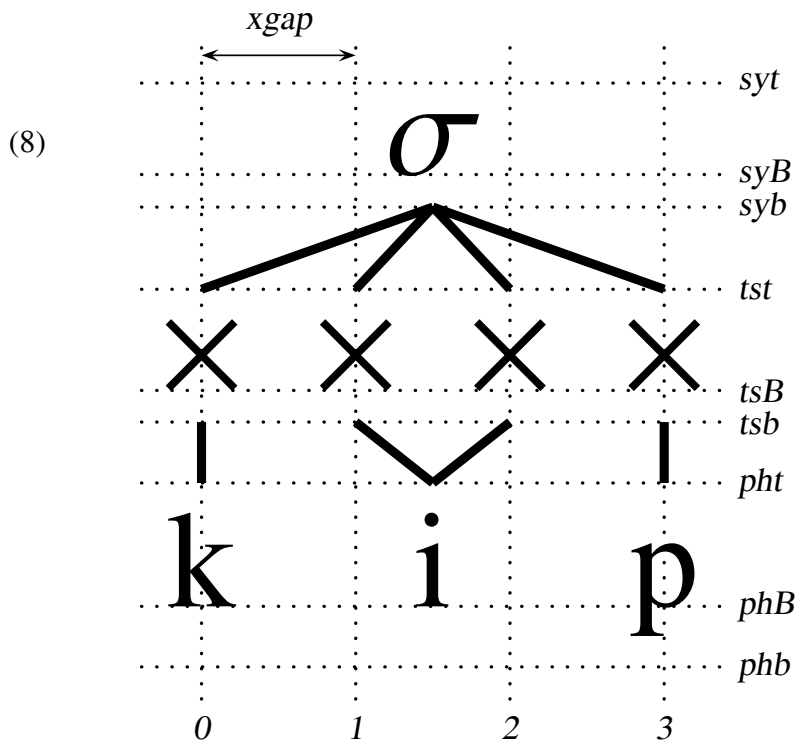
For example, (7b) produces (7a).



b. `\dbox`
`\dput[B](0,\ASRtsB){\times}`
`\dput[B](1,\ASRtsB){\times}`
`\dput[B](2,\ASRtsB){\times}`
`\dput[B](3,\ASRtsB){\times}`
`\dput[B](0,\ASRphB){k}`
`\psline(0,\ASRpht)(0,\ASRtsb)`
`\dput[B](1.5,\ASRphB){i}`
`\psline(1.5,\ASRpht)(1,\ASRtsb)`
`\psline(1.5,\ASRpht)(2,\ASRtsb)`
`\dput[B](3,\ASRphB){p}`
`\psline(3,\ASRpht)(3,\ASRtsb)`
`\dput[B](1.5,\ASRsyB){σ}`
`\psline(0,\ASRtst)(1.5,\ASRsyb)`
`\psline(1,\ASRtst)(1.5,\ASRsyb)`
`\psline(2,\ASRtst)(1.5,\ASRsyb)`
`\psline(3,\ASRtst)(1.5,\ASRsyb)`
`\enddbox`

There are 16 lines of code inside `\dbox... \enddbox`, one for each of the 4 timing slots, 3 phonemes, 1 syllable symbol, and 8 association lines. Be patient, we will soon see that (7a) can be produced much more simply. But we need to take one step at a time in order to understand how *pst-asr* operates.

An magnified version of (7a) is given below, with the grid explicit, which can be compared with the code. Each piece of the display corresponds to a line of code in (7b).



4. Putting elements on tiers and associating elements

`\tierput`, `\assoc`, `\tiersshortcuts`

The two general structure building macros are `\tierput` and `\assoc`.

`\tierput [x-offset] (x, tier) {stuff}`

If $x' = x + x\text{-offset}$, this is equivalent to `\dput [B] (x', \ASRtierB) {stuff}` with respect to what is displayed. Additionally, both x' and the tier name are saved (in the macros `\ASR@lasttierx` and `\ASR@lasttier`). If there is no offset argument, the offset defaults to 0.

`\assoc (xA, tierA) (xB, tierB)`

Suppose that tierA has levels y_A^B , y_A^{top} , and y_A^{bot} ; and that tierB has levels y_B^B , y_B^{top} , and y_B^{bot} . If $y_A^B > y_B^B$, then an association line is drawn from (x_A, y_A^{bot}) to (x_B, y_B^{top}) , otherwise, an association line is drawn from (x_B, y_B^{bot}) to (x_A, y_A^{top}) . If only a single point follows `\assoc`, the second point is assumed to be `(\ASR@lasttierx, \ASR@lasttier)`.

`pst-asr` contains

`\def\tiersshortcuts{\let\@=\tierput \let\-=\assoc}`

Now, the code (7b) above can be simplified:

```
(9) \dbox
\tiershortcuts
\@{0,ts}{$\times$}
\@{1,ts}{$\times$}
\@{2,ts}{$\times$}
\@{3,ts}{$\times$}
\@{0,ph}{k}\-(0,ts)
\@{1.5,ph}{i}\-(1,ts)\-(2,ts)
\@{3,ph}{p}\-(3,ts)
\@{1.5,sy}{$\sigma$}\-(0,ts)\-(1,ts)\-(2,ts)\-(3,ts)
\enddbox
```

This is an improvement, but to gain more significant simplification a parser is needed.

5. The parser

Macros: `\asr`, `\endasr`

Parameters: `tssym`, `asrB`, `everyph`, `everyasr`

Dimension register: `\xpos`

The code (10a) produces the display (10b), given one choice of parameter settings.

(10) a. `\asr kat\endasr`

```
b.  ×  ×  ×
    |  |  |
    k  a  t
```

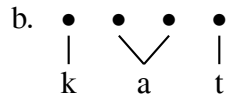
`\asr` starts a `dbox`, initializes the dimension register `\xpos` to 0 pt, and initiates parsing. `\xpos` keeps track of the horizontal progress of structure building, which proceeds from left to right as characters are read from left to right, until `\endasr` terminates the process, closing the `dbox`. After each character is read (unless it is followed by `:`), a timing slot, phoneme, and association are drawn, and a dimension `\xpos` is stepped by `\ASRxgap`. If `:` follows the character, a geminate is typeset and `\xpos` is stepped by twice `\ASRxgap`.

(11) a. `\asr ka:t\endasr`

```
b.  ×  ×  ×  ×
    |  \  |  |
    k   a  t
```

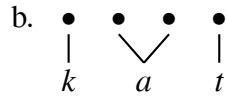
The symbol that is used for timing slots is determined by the parameter `tssym`, with the setting `tssym=\times` the default. So, for example:

(12) a. `\asr[tssym=\bullet] ka:t\endasr`



Phonemes are typeset by putting them in an hbox and placing the hbox appropriately. Inside each of these phoneme boxes, the macro `\ASR@everyph` is evaluated as the first order of business. Its body is set by the parameter `everyph`. So:

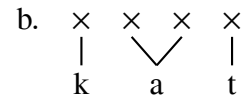
(13) a. `\asr[tssym=\bullet,everyph=\it] ka:t\endasr`



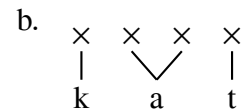
`everyph` is mainly intended as a way to allow the user to automatically switch to IPA fonts when typesetting phonemes.

The default is to put the baseline of the hbox that `\asr... \endasr` builds at the $y = 0$ level. This can be changed using the `asrB` parameter. The default is to put the baseline of the timing tier at the $y = 0$ level. Compare the following, assuming the default setting of `asrB` in (14a) and the default timing tier level throughout.

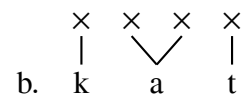
(14) 1. a. `\asr ka:t\endasr`



2. a. `\asr[asrB=1ex] ka:t\endasr`



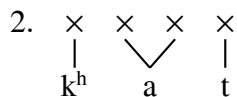
3. a. `\asr[asrB=\ASRphB] ka:t\endasr`



5.1. Special parser tokens: `{`, `|`, `<`, `'`, and control sequences

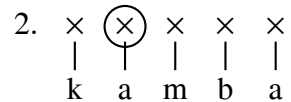
If the parser encounters `{`, it assumes a group follows and it treats the entire group as a single phoneme and typesets it accordingly.

(15) 1. `\asr {k$\rm ^h$}a:t\endasr`



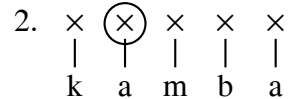
If the parser encounters `|`, parsing is temporarily suspended until a second `|` is encountered. `|\...|` is therefore a region in the code where macros of various kinds can be executed during the parse of a string of phonemes.

(16) 1. `\asr k|\pscircle(\xpos,.55ex){1.5ex}|amba\endasr`



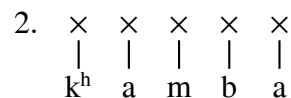
If the parser encounters a control sequence, it is simply evaluated, not set as a phoneme, and the parser moves on to the next token in the string.

(17) 1. `\def\goop{\pscircle(\xpos,.55ex){1.5ex}}
\asr k\goop amba\endasr`



Control sequences which are intended to expand to phonemes must be enclosed in braces so that they receive the correct interpretation.

(18) 1. `\def\kh{k$\rm ^h$}%
\asr {\kh}amba\endasr`

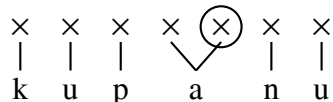


If the parser encounters the character <, it assumes that the next token is a control sequence (say \CS). <\CS is read and has the same effect as:

`|{\stepxpos{-\ASRxgap}\CS}|`

The modification of \xpos is grouped, so it has no effect outside the group.

< is provided so that there is an easy way to evaluate control sequences at the second timing slot of a geminate. Suppose you want to use \goop from (17) to typeset:



The solution is: `\asr kupa:<\goop nu\endasr`

If the parser encounters ', the following token is set as a timing tier delimiter. The details are explained in Appendix A.

If the parser encounters \endasr, parsing is terminated and the hbox is closed.

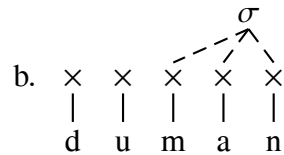
6. Syllables

Macros: \varsyl, \qsyl, \monosysym, \bisysym, \1, \2, \3, \4, \5, \DefList
PST styles: medsyls, bigsyls
parameter: sysym

The default syllable symbol is set by the parameter sysym. The general syllable building macro is \varsyl, with syntax:

`\varsyl [pars] x-offset {x-offset1, ..., x-offsetn}`

(19) a. `\asr du|\varsyl[linestyle=dashed]{1.5}{0,1,2}|man\endasr`

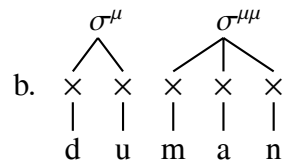


Note that `\varsyl` is in the nonparsing `|\dots|` environment. Macros which take arguments must be outside the parsing environment. Note also that the x -offsets from the current `\xpos` are required, not absolute positions. Finally, note that the parser skips over spaces (i.e. the one at the end of the first line in the code above).

`pst-asr` includes:

```
\def\monosysm{${\sigma}^{\mu}\mu}\mskip-10mu$}
\def\bisysm{${\sigma}^{\mu}\mu}\mskip-18mu$}
```

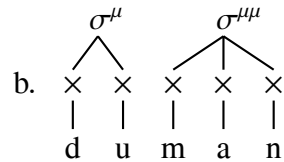
(20) a. `\asr |\varsyl[sysym=\monosysm]{.5}{0,1}|du|\varsyl[sysym=\bisysm]{1}{0,1,2}|man\endasr`



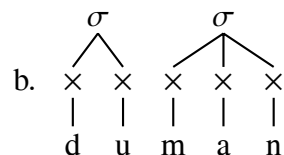
`\qsyl (symbol) n`

`\qsyl` is a streamlined version of `\varsyl` which does not take parameters and builds a symmetric syllable over the following n timing slots. The `(symbol)` argument is optional. If it is present, the symbol is used as the syllable symbol. If it is not, the syllable symbol set by the `sysym` parameter (accessible as `\ASR@sysym`) is used.

(21) a. `\asr |\qsyl(\monosysm)2|du|\qsyl(\bisysm)3|man\endasr`



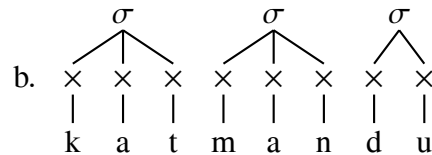
(22) a. `\asr |\qsyl2|du|\qsyl3|man\endasr`



pst-asr contains:

```
\def\1{\qsyl 1}
\def\2{\qsyl 2}
\def\3{\qsyl 3}
\def\4{\qsyl 4}
\def\5{\qsyl 5}
```

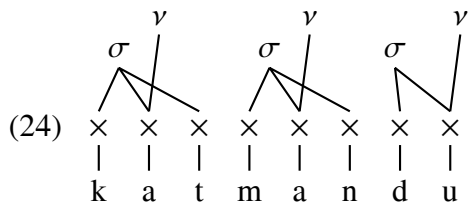
(23) a. `\asr \3kat\3man\2du\endasr`



Macros that do not take arguments can appear in the parsing environment. They are simply evaluated and the parser proceeds to the next token.

6.1. Example

We have accumulated enough machinery to typeset a moderately complex example. Suppose you believe (as I do) that syllables have an autosegmental structure, not a constituent structure. Then you might want to typeset something like (24), with nuclei on a different tier than syllables.



One way to typeset (24) is given in (25). The style `bigsyls` is defined in *pst-asr*.

```
\newsstyle{bigsyls}{extragap=.6ex,unitxgap=true,xgap=3.5ex,
ts=0pt ($\times$),sy=5.5ex ($\sigma$) .7ex,
ph=-4.5ex (pf)}
```

```
(25) \newtier{nuc}
      \tiershortcuts
      \asr[style=bigsyls,nuc=(sy) 1em ($\nu$)]
      |\varsyl{.4}{0,1,2}|kat
      |\varsyl{.4}{0,1,2}|man
      |\varsyl{-.1}{0,1}|du
      |\@[.2](1,nuc){$\nu$}\-(1,ts)
      |\@[.2](4,nuc){$\nu$}\-(4,ts)
      |\@[.2](7,nuc){$\nu$}\-(7,ts)
      \endasr
```

One weakness in the code in (25) is that the settings which must be tweaked to get a good looking display (.4, -.1, and .2) are spread out in various places in the code. This makes adjustment difficult. *pst-asr* provides a convenient way to concentrate the parameters which might need to be adjusted.

```
(26) \newtier{nuc}
      \DefList{\syloffA{.4},\syloffB{-.1},\nucoeff{.2}}
      \tiershortcuts
      \asr[style=bigsyls,nuc=(sy) 1em ($\nu$)]
      |\varsyl{\syloffA}{0,1,2}|kat
      |\varsyl{\syloffA}{0,1,2}|man
      |\varsyl{\syloffB}{0,1}|du
      |\@[ \nucoeff ](1,nuc){$\nu$}\-(1,ts)
      |\@[ \nucoeff ](4,nuc){$\nu$}\-(4,ts)
      |\@[ \nucoeff ](7,nuc){$\nu$}\-(7,ts)
      \endasr
```

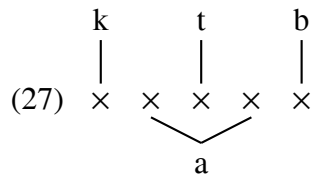
`\DefList` distributes `\def` over the list which follows and ignores following spaces. Note also that inside `\asr... \endasr`, spaces which follow `|`, or which follow tokens or groups which are parsed as phonemes, are ignored.

If you prefer writing `\@` and `\-` to writing `\tierput` and `\assoc`, you might want to execute `\tiersshorts` someplace outside any group so that the shortcuts are available globally. There is an option. *pst-asr* provides a way for it to be automatically enforced inside `\asr... \endasr`, but not outside. When `\asr` is executed, immediately after construction of the `dbox` is begun, the macro `\ASR@everyasr` is inserted. The body of that macro (which does not take arguments) is set by means of the parameter `everyasr`. So if you say `\psset{everyasr=\tiersshortcuts}`, then the shortcuts will be in effect inside `\asr... \endasr`, but not outside it.

7. Inserting timing slots and phonemes and associating them

Macros: `\varph`, `\bare`, `\X`, `\asrsetkeys`

Suppose you would like to typeset:



Three macros which are provided in *pst-asr* are relevant.

`\bare symbol`

The symbol is typeset on the timing tier at the current position, then the current position is advanced to the position of the next timing slot.

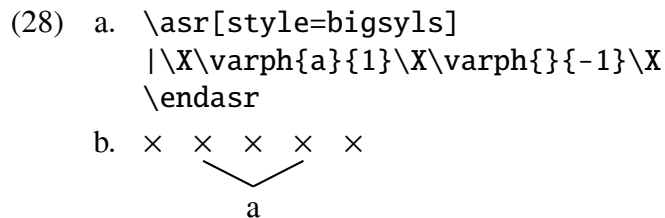
`\X`

Defined to be `\bare{\ASR@tssym}`.

`\varph phoneme x-offset`

The phoneme is typeset on the phoneme tier at the specified offset from the current position and the current position is advanced to the position of the next timing slot.

Then, (28a) produces (28b).

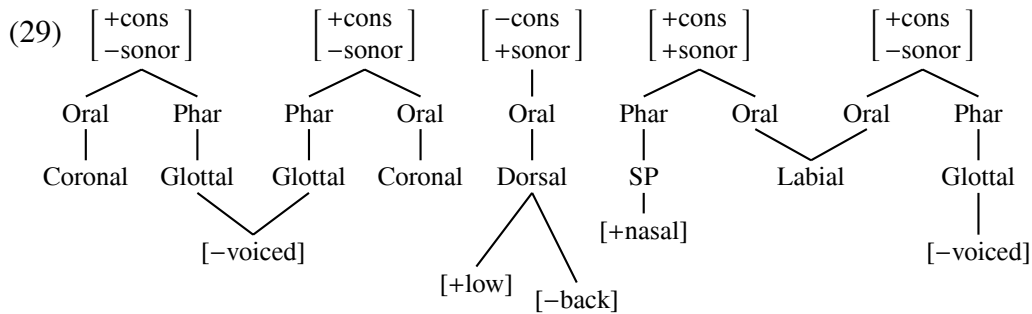


The full code for (27) is:

```
\asr[style=bigstyle]
|\qsyl(k)1 \X \varph{a}1 \qsyl(t)1 \X \varph{}{-1}
 \qsyl(b)1 \X
\endasr
```

7.1. Example

The display (29) is from Kenstowicz (1995:). A surprising amount of it can be done using the standard parsing and syllable building facilities of *pst-asr*.



In order to fit on the page, it is set in tenpoint. Since all dimension are given in ex and em units, the effect is that the whole diagram is reduced, including all the tier levels.

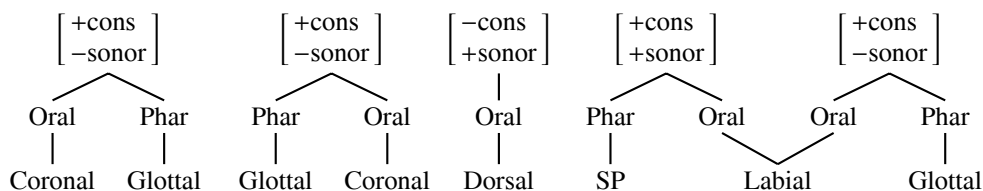
We first need a macro to construct the feature matrices.

```
\def\#1#2{%
  $\left[\matrix{\rm #1cons\hfill\cr \rm#2sonor\cr}\right]$}
```

If you are going to use this macro outside this particular display, a more informative name should be given to it.

Most of the diagram above can be constructed using the three preassigned tiers. The code below produces the diagram which follows it.

```
\asr[unit=2em,xgap=4.2em,sy=1.5 (\+\+),
  ts=0 (G),ph=-1.2 (C),tssym=Oral,sysym=\+\-]
\2{Coronal}|\varph[tssym=Phar]{Glottal}{0}|
\2|\varph[tssym=Phar]{Glottal}0|{Coronal}
|\qsyl(\+\-+)1|{Dorsal}
|\qsyl(\+\+ )2|\varph[tssym=Phar]{SP}{0}|{Labial}
:<\2|\varph[tssym=Phar]{Glottal}{0}|
\endasr
```



If you want to understand how the parser operates, it is worth studying this example and making sure you understand why it produces the display that it does.

The full code for (29) is given below. Defining \PharGlot and \PharSP is not necessary, but does make the code more readable. \asrsetkeys is a more efficient alternative to \psset if all the keys which must be set are asr-keys. \psset must search the entire family of keys associated with PSTricks, while \asrsetkeys searches for keys only in the asr subgroup of that family.

```

\psset{tssym=Oral,sysym=\+-,unit=2em,xgap=4.2 em,
  sy=1.5 (\++),ts=0 (0),ph=-1.2 (C),asrB=\ASRsyB}
\newtier{voice,nasal,low,back}
\asrsetkeys{voice=(ph) -1.4 ([],nasal=(ph) -1 ([],
  low=(ph) -2 ([],back=(ph) -2.3 ([])}
\DefList{\lowoff{-.5},\backoff{.4}}
\def\PharGlot{\varph[tssym=Phar]{Glottal}0}%
\def\PharSP{\varph[tssym=Phar]{SP}0}%
\asr
\2{Coronal}\PharGlot
\2\PharGlot{Coronal}
|\qsyl(\-+)1|{Dorsal}
|\qsyl(\++ )2|\PharSP{Labial}
:<\2\PharGlot
|@(1.5,voice){\feat{-voiced}}
  \-(1,ph)\-(2,ph)
|@(8,voice){\feat{-voiced}}
  \-(8,ph)
|@(5,nasal){\feat{+nasal}}
  \-(5,ph)
|@[\lowoff](4,low){\feat{+low}}
  \-(4,ph)
|@[\backoff](4,back){\feat{-back}}
  \-(4,ph)
\endasr

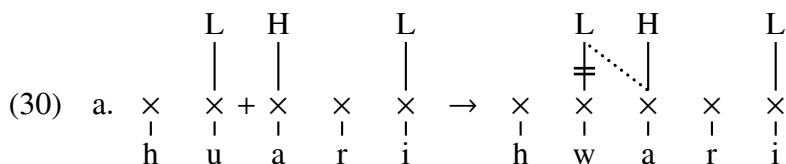
```

8. Marking disassociation

Macros: \xedparlines, \xedcirc, \feat

Parameters: xed, xedtype, xedratio, xedht, xedsep, xedlinewidth

Representations like the one on the right side of (30a) are well known. It is supposed to portray the delinking of the initial high tone, resulting from glide formation, and its subsequent linking to the following vowel. *pst-asr* provides the means to typeset them relatively easily. The code is given in (30b).



```

b. \asr[xgap=2em, syB=2.5em] hwari |
   \@ (1, sy) {L}
     \- [xed=true, xedratio=.4] (1, ts) \- [style=dotted] (2, ts)
   \@ (2, sy) {H}
     \- (2, sy) (2, ts)
   \@ (4, sy) {L}
     \- (4, ts)
   \endasr

```

A later section will explain how to easily typeset the timing tier junction “+” which appears on the left side of (30a).

If the boolean parameter `xed` is set to true, `\assoc` places an xed-mark on the association line it draws. Its position along the line is determined by the setting of the parameter `xedratio`. `xedratio=1` places the mark at end of the association line corresponding to the first of the two points, or at the position of the last `\tierput` if `\assoc` has only one argument. `xedratio=0` places it at the other end of the association line. The type of xed-mark that is drawn depends on the setting of the parameter `xedtype`, which is set to the macro which draws the xed-mark. *pst-asr* provides two xed-marks, `\xedparlines`, which is used above, is the default. The parameters `xedsep`, `xedlinewidth`, and `xedht` are used to modify the characteristics of `\xedparlines`, in easily predicatable ways. `\xedcirc` is also available, which is used to indicate crossed association lines. `\xedcirc` cannot be modified by parameter setting as written, but users can easily modify the definition if they want to. *pst-asr* contains:

```

\def\xedcirc{%
  \pscircle[linestyle=solid,linewidth=.5pt](0,0){.7ex}}

```

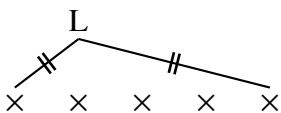
The default parameter settings which are relevant to xed-marks, as set by *pst-asr*, are:

```

xedsep=2.2pt, xedht=8pt, xedratio=.5, xedlinewidth=1pt,
xedtype=\xedparlines, xed=false

```

The xed-mechanism is clever enough to rotate an `\xedparlines` mark so that it is perpendicular to the association line it marks.

(31) a. 

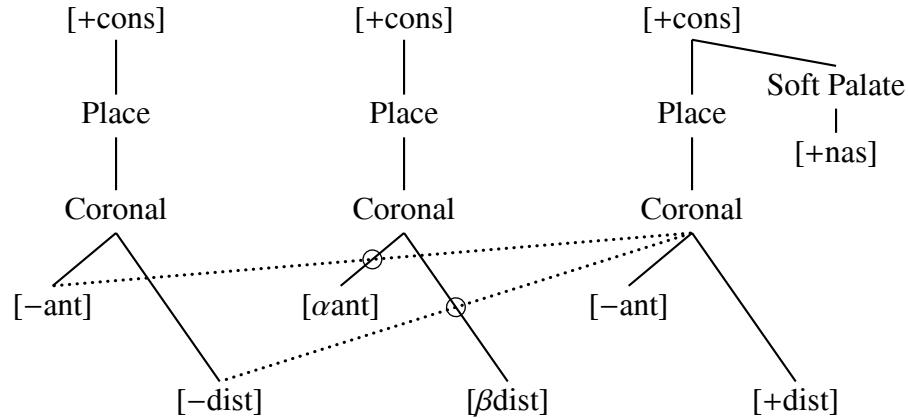
```

b. \asr[xgap=2em, syB=2.5em]
   \X\X\X\X\X
   \@ (1, sy) {L}
     \- [xed=true] (0, ts)
     \- [xed=true] (4, ts)
   \endasr

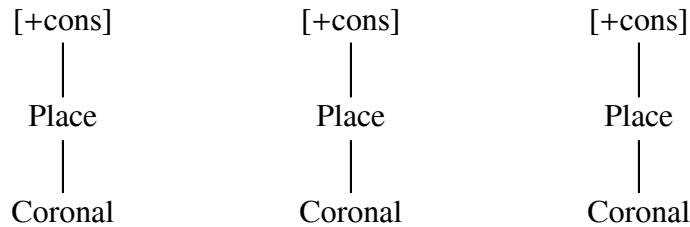
```

8.1. Example (coronal assimilation in Sanskrit)

Halle (1995) gives the following diagram to illustrate why intervening coronal consonants block coronal assimilation, which proceeds to the left from retroflex coronals.



We begin the construction with (32), commandeering the *ts*, *ph*, and *sy* tiers for a different use.



The macro `\feat`, defined by

```
\def\feat#1{\rm [#1]}
```

is useful here and generally. The display above is generated by

```
\psset{xcgap=1.5in,yunit=3em,ts=0 (Pg),sy=1 ([),
  ph=-1 (Cg),tssym=Place,sysym=\feat{+cons},
  everyph=Coronal}
```

This skeleton must be fleshed out by defining new tiers and building the rest of the structure. The full code is:

```
\newtier{softpal,ant,dist,nasal}%
\newpsstyle{crossing}{xed=true,xedtype=\xedcirc,
  style=dotted}
\tiersshortcuts
\psset{xcgap=1.5in,yunit=3em,ts=0 (Pg),sy=1 ([),
  ph=-1 (Cg),softpal=.3 (Sg),
  nasal=-.4 ([),ant=-2 ([),dist=-3 ([),
  tssym=Place,sysym=\feat{+cons},everyph=Coronal}
\DefList{\softpalA{2.5},\antoffset{-.22},\distoffset{.36}}
```

```

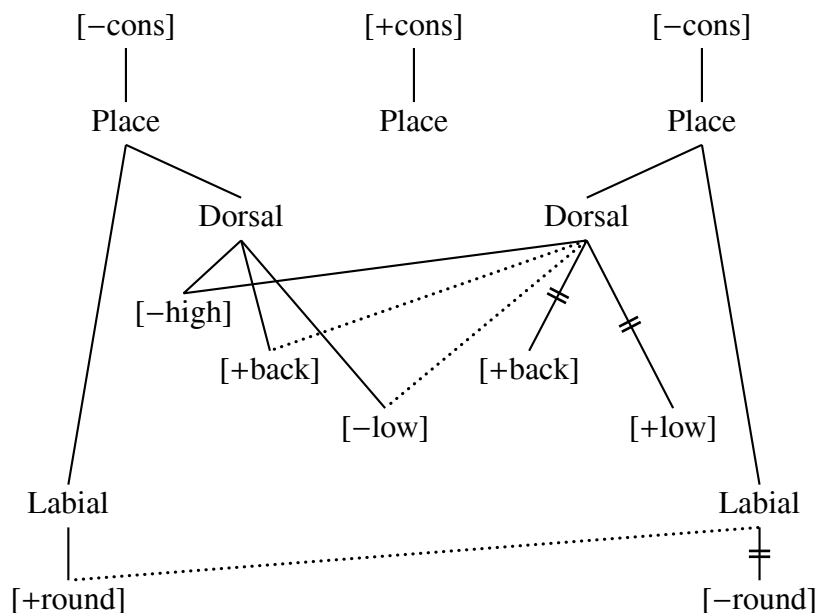
%
\asr \1{\1{\1}|
\@(\softpalA,softpal){Soft Palate}
  \-(2,sy)
\@(\softpalA,nasal){\feat{+nas}}
  \-(\softpalA,softpal)
\@(\antoffset,ant){\feat{-ant}}
  \-(0,ph)
  \-[style=crossing](2,ph)
\@[1](\antoffset,ant){\feat{\alpha ant}}
  \-(1,ph)
\@[2](\antoffset,ant){\feat{-ant}}
  \-(2,ph)
\@(\distoffset,dist){\feat{-dist}}
  \-(0,ph)
  \-[style=crossing](2,ph)
\@[1](\distoffset,dist){\feat{\beta dist}}
  \-(1,ph)
\@[2](\distoffset,dist){\feat{+dist}}
  \-(2,ph)
\endasr

```

Note that the depth of the *softpal*, *ts*, and *ph* tiers is increased slightly by including a character with a descender in the height/depth setting of *softpal*, *ts*, and *ph*. To my eye, this looks better. Unless you want the four tiers and the definitions to be global, the whole construction should be grouped.

8.2. Example (Wikchammi vowel harmony)

This is from Halle (1995:32).



```

\def\feat#1{\rm [#1]$}%
\newtier{dorsal,high,back,low,labial,round}
\psset{xcgap=1.5in,yunit=3em,ts=0 (Pg),sy=1 (C),
  dorsal=-1 (Dg),high=-2 ([),back=-2.6 ([),
  low=-3.2 ([),labial=-4 (Lg),round=-5 ([),
  sysym=\feat{-cons},tssym=Place}
\DefList{\dorsalA{.4},\dorsalB{1.6},\highoffset{-.2},%
  \backoffsetA{.1},\backoffsetB{-.2},\lowoffsetA{.5},%
  \lowoffsetB{.3},\labialA{-.2},\labialB{2.2}}
\asr \1\X\X\1\X
|@(1,sy){\feat{+cons}}
  \-(1,ts)
|@(\dorsalA,dorsal){Dorsal}\-(0,ts)
|@(\dorsalB,dorsal){Dorsal}\-(2,ts)
|@[\highoffset](\dorsalA,high){\feat{-high}}
  \-(\dorsalA,dorsal)
  \-(\dorsalB,dorsal)
|@[\backoffsetA](\dorsalA,back){\feat{+back}}
  \-(\dorsalA,dorsal)
  \-[style=dotted](\dorsalB,dorsal)
|@[\backoffsetB](\dorsalB,back){\feat{+back}}
  \-[xed=true](\dorsalB,dorsal)
|@[\lowoffsetA](\dorsalA,low){\feat{-low}}
  \-(\dorsalA,dorsal)
  \-[style=dotted](\dorsalB,dorsal)
|@[\lowoffsetB](\dorsalB,low){\feat{+low}}
  \-[xed=true](\dorsalB,dorsal)
|@(\labialA,labial){Labial}
  \-(0,ts)
|@(\labialB,labial){Labial}
  \-(2,ts)
|@(\labialA,round){\feat{+round}}
  \-(\labialA,labial)
  \-[style=dotted](\labialB,labial)
|@(\labialB,round){\feat{-round}}
  \-[xed=true](\labialB,labial)
\endasr

```

9. Reusing the syllable tier

Macros: \LevelsIncrement, \setxpos, \stepxpos

Instead of defining a new tier above the syllable tier, it is sometimes easiest to first put in the syllables, then raise the syllable tier and reuse it for the needed extra tier. The advantage is that the syllable building macros \varsyl and \qsyl can be

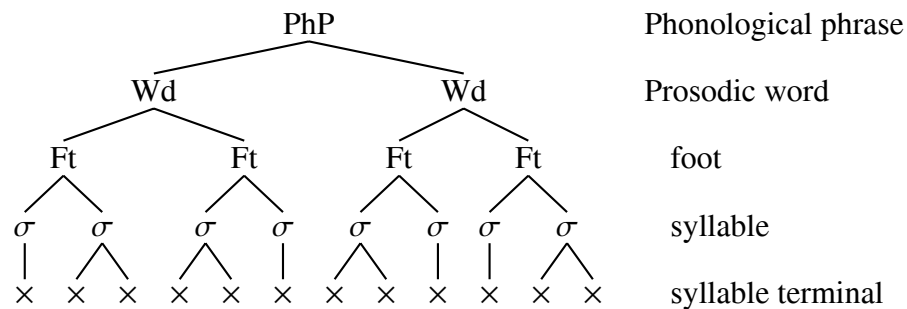

```

\def\.{\stepxpos{- .5}\tierput(\xpos,ts){\bf .}}%
\asr[ts=0 (x),sy=1.5 ($*$),ph=-1.6 (C),
     xgap=1.2em,syB=3.2ex,sysym=$*$]
C\(\1V\ .C\1V\ .C\1V\ .C\(\1VX\ .C\1V\ .C\1V\ .
   C\(\1VX\ .C\1V\ .C\1V\ .C\1V\ .C\1V
|\LevelsIncrement
  \setxpos1
  \1\C
  \stepxpos6
  \1
  \stepxpos7
  \1
\LevelsIncrement
  \setxpos1
  \1
\endasr

```

Unless you want the definitions of $\backslash C$ and $\backslash .$ to be global, grouping should be used to confine their scope. The same comment applies in general to preliminary definitions to $\backslash asr \dots \backslash endasr$.

The next example reproduces a diagram in Blevins (1995:210), intended to illustrate the “universal prosodic hierarchy.” It shows how to label tiers in addition to illustrating the use of $\backslash IncrementLevels$.



```

\asr[unit=3.6ex,yunit=1.3,ts=0 ($\times$),
     sy=1 ($\sigma$),xgap=1]
\1\X\2\X\X\2\X\X\1\X\2\X\X\1\X\1\X\2\X\X
|\dput[B1](12.5,\ASRtsB){syllable terminal}
\dput[B1](12.5,\ASRsyB){syllable}
\LevelsIncrement
\asrsetkeys{sysym=Ft,sy=* (F)}
  \setxpos0
  \varsyl{.75}{0,1.5}
  \varsyl{4.25}{3.5,5}
  \varsyl{7.25}{6.5,8}
  \varsyl{9.75}{9,10.5}

```

```

\dput[B1](12.5,\ASRsyB){foot}
\LevelsIncrement
\asrsetkeys{sysym=Wd,tsht=! .3ex}
\varsyl{2.5}{.75,4.25}
\varsyl{8.5}{7.25,9.75}
\dput[B1](12,\ASRsyB){Prosodic word}
\LevelsIncrement
\asrsetkeys{sysym=PhP}
\varsyl{5.5}{2.5,8.5}
\dput[B1](12,\ASRsyB){Phonological phrase}
\endasr

```

10. Representations with no timing tier

Sometimes abbreviated representations like (32) are desired for one reason or another. The parser uses the macros `\putph` and `\putgem` to typeset phonemes and geminates. These macros can be redefined so that phonemes are placed on the *ts* tier and geminates indicated by a doubled phoneme.

(32) $\begin{array}{c} \sigma & \sigma & \sigma \\ \wedge & \wedge & \wedge \\ k & a & t & m & a & n & d & u & u \end{array}$

The relevant code from *pst-asr* is given below. It is shown here so that users can see how simple it is. It can easily be modified by the user, if the need arises for some particular application.

```

\def\notsputph#1{%
  \tierput(\xpos,ts){#1}%
  \stepxpos{1}%
}
\def\notsputgem#1{%
  \notsputph{#1}%
  \notsputph{#1}%
}
\def\notsrep{%
  \let\putph=\notsputph
  \let\putgem=\notsputgem
  \asrsetkeys{ts=* (ky)}%
}

```

The code (33) produces (32).

```

(33) \notsrep
      \asr \3kat\3man\3du:\endasr

```

10.1. Example

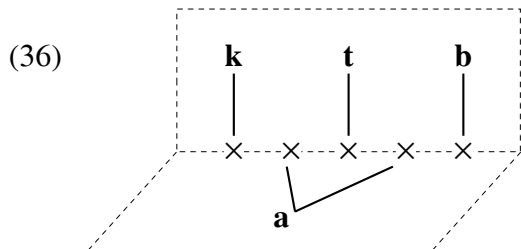
Displays like the ones in (34) are used to illustrate the preservation of tones when their supporting vowel is eliminated.

(34)	$\begin{array}{cccc} \text{H} & \text{L} & \text{H} & \text{L} \\ & & & \\ \text{c} & \text{e} & \text{d} & \text{e} + \text{a} & \text{r} & \text{i} \end{array}$	$\begin{array}{ccc} \text{L} & \text{H} & \text{L} \\ & & \\ \text{h} & \text{u} & + \text{a} & \text{r} & \text{i} \end{array}$	underlying
	$\begin{array}{cccc} \text{H} & \text{L} & \text{H} & \text{L} \\ & & & \\ \text{c} & \text{e} & \text{d} + \text{a} & \text{r} & \text{i} \end{array}$	$\begin{array}{ccc} \text{L} & \text{H} & \text{L} \\ & & \\ \text{h} & \text{w} & + \text{a} & \text{r} & \text{i} \end{array}$	syncope/glide formation
	$\begin{array}{cccc} \text{H} & \text{L} & \text{H} & \text{L} \\ & \dots & & \\ \text{c} & \text{e} & \text{d} + \text{a} & \text{r} & \text{i} \end{array}$	$\begin{array}{ccc} \text{L} & \text{H} & \text{L} \\ & \dots & & \\ \text{h} & \text{w} & + \text{a} & \text{r} & \text{i} \end{array}$	reassociation

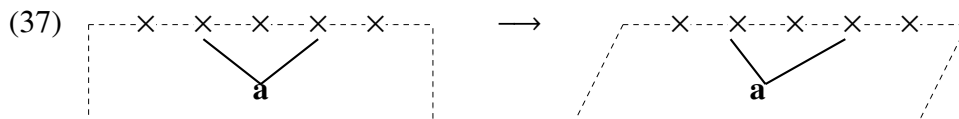
```
(35) \vtop{%
\def\H{\varsyl[sysym=H]0{0}}%
\psset{xgap=1em,sysym=L,syB=2.2em}
\notsrep
\halign{#\hfil&& \hskip3em #\hfil\cr
\asr c\H ed\le{+}$}\H ar\li\endasr&
\asr h\lu{+}$}\H ar\li\endasr&
underlying\cr
\noalign{\vskip1em}
%
\asr c\H ed{+}$}\H ar\li
|\tierput(2.5,sy){L}\endasr&
\asr hw{+}$}\H ar\li
|\tierput(1,sy){L}\endasr&
syncope/glide formation\cr
\noalign{\bigskip}
%
\asr c\H ed{+}$}\H ar\li
|\tierput(2.5,sy){L}
\assoc[style=tonedot](4,ts)\endasr&
\asr hw{+}$}\H ar\li
|\tierput(1,sy){L}
\assoc[style=tonedot](3,ts)\endasr&
reassociation\cr
}}
```

11. Morpheme tiers

Halle & Vergnaud (1987:79) use (36) to illustrate McCarthy's idea of planar segregation of consonants and vowels in the Semitic languages (see, for example, McCarthy, 1986, "OCP Effects: Gemination and Antigemination").



If the center line of the timing tier is at $y = 0$ and coordinates are transformed according to $(x, y) \rightarrow (x + .5y, y)$, the display transforms as shown in (37). Since $y < 0$, points will be moved to the left, a distance proportional to their y -coordinates.



The crucial macro in carrying this out is:

```
\skewx( x-coord , y-coord ) macro_name
```

It defines a macro with the given name which evaluates to the new x -coordinate of the given point. `\skewline` and `\skewput` use `\skewx` in the expected way.

```
\def\skewx(#1,#2)#3{%
  \pssetxlength\dimpuba{#1}%
  \pssetylength\dimpubb{#2}%
  \advance\dimpuba by \slanratio\dimpubb
  \edef#3{\the\dimpuba}%
}
\def\skewline#1(#2,#3)#4(#5,#6){%
  \skewx(#2,#3)\tempa
  \skewx(#5,#6)\tempb
  \psline#1(\tempa,#3)(\tempb,#6)%
}
\def\skewdput#1(#2,#3){%
  \skewx(#2,#3)\tempa
  \dput#1(\tempa,#3)%
}

```

An important fine point in (36) is the whitespace around the timing slots. The usual timing slot is put in a `ps-framebox`, with a solid (white) fill. The timing slots are then typeset after the dashed line is drawn.

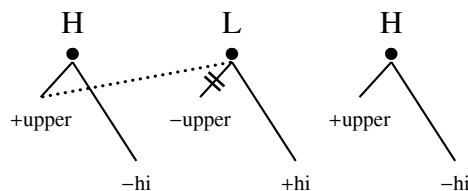
```
\def\ts{\psframebox*[framesep=0pt]{$\times$}}
```

We can now give the full code. Note on line 4 that the height of the *ph*-tier is made quite small and that the baseline of the display is made to correspond to the syllable level. On line 6, the full frame of the vertical page is drawn before the timing slots are overwritten. On line 12, the starting position of the skew line is fudged slightly (from 1 to 1.15) for better appearance. Lines 18 and 19 are used to size the bounding box.

```
1 \newpsstyle{pageborder}{linestyle=dashed,linewidth=.3pt,
2   dash=2pt 1.8pt}
3 \psset{unit=4ex,xgap=1,ts=-.5ex ($\times$),
4   ph=-1.3 (e) .5pt,sy=1.5 (s),tssym=\ts,asrB=1.5}
5 \DefList{\vpagetop{2.5},\hpagebot{-1.7},\slanratio{.9}}
6 \asr |\psframe[style=pageborder](-1,0)(5,\vpagetop)|
7 \X\X\X\X\X
8 |\skewx(-1,\hpagebot)\tempa
9 \skewx(5,\hpagebot)\tempb
10 \psline[style=pageborder]
11   (-1,0)(\tempa,\hpagebot)(\tempb,\hpagebot)(5,0)
12 \skewline(1.15,\ASRtsb)(2,\ASRpht)
13 \skewline(3,\ASRtsb)(2,\ASRpht)
14 \skewdput[B](2,\ASRphB){\bf a}
15 \@(\0,sy){\bf k}\-(0,ts)
16 \@(\2,sy){\bf t}\-(2,ts)
17 \@(\4,sy){\bf b}\-(4,ts)
18 \skewdput(-1,\hpagebot){}
19 \dput(0,\vpagetop){}
20 \endasr
```

12. Miscellaneous examples

12.1. Unusual timing slot symbol



```
\DefList{\hioff{.4},\upoff{-.2}}%
\def\#1{\scriptstyle\rm #1}%
\asr[unit=2em,xgap=5em,ts=0 ($\bullet$) 1ex 0,tssym=$\bullet$,
  phB=-1,syB=\ASRtst]
\X\X\X
|@\[\upoff](0,ph){\{\+upper\}}
  \-(0,ts)
```

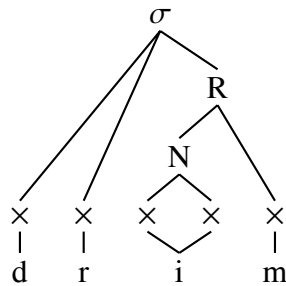
```

\-[style=dotted](1,ts)
\@[\upoff](1,ph)
  {\{-upper}}
  \-[xed=true,xedratio=.55](1,ts)
\@[\upoff](2,ph){\{+upper}}
  \-(2,ts)
\asrsetkeys{phB=-1}
\setxpos0
\varph{\{-hi}}{\hioff}
\varph{\{+hi}}{\hioff}
\varph{\{-hi}}{\hioff}
\@(0,sy){H}
\@(1,sy){L}
\@(2,sy){H}
\endasr

```

12.2. Syllable constituent structure

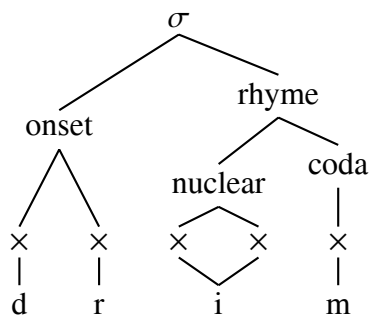
Blevins (1995), in *The Handbook of Phonological Theory*.



```

\newtier{nuclear,rhyme}
\psset{xgap=2em,yunit=1.8em,
  phB=-1,syB=3.5,nuclear=1 (N),rhyme=2.2 (R)}
\DefList{\sypos{2.2},\rhymepos{3.1}}
\hfil\asr |\varsyl{\sypos}{0,1}|
dri:m
| \@(2.5,nuclear){N}
  \-(2,ts)
  \-(3,ts)
\@(\rhymepos,rhyme){R}
  \-(\sypos,sy)
  \-(2.5,nuclear)
  \-(4,ts)
\endasr

```



```

\newtier{nuclear,rhyme,coda,onset}
\psset{xgap=2.5em,yunit=2em,
  phB=-1,nuclear=.9 (lg),coda=1.2 (dg),rhyme=2.3 (hy),
  onset=1.8 (tg),syB=3.5}
\DefList{\onsetpos{.5},\nuclearpos{2.5},\rhymepos{3.25}}
\asr dri:m
|@(\nuclearpos,nuclear){nuclear}
  \-(2,ts)
  \-(3,ts)
|@(\coda,coda){coda}
  \-(4,ts)
|@(\onsetpos,onset){onset}
  \-(0,ts)
  \-(1,ts)
|@(\rhymepos,rhyme){rhyme}
  \-(4,coda)
  \-(\nuclearpos,nuclear)
|@(\sy,sy){\sigma}
  \-(\onsetpos,onset)
  \-(\rhymepos,rhyme)
\endasr

```

12.3. Derivation of foot structure

- (38) a. $\omega \ \omega \ \omega \ \omega$ b. $\omega \ \omega \ \omega - \omega$ c. $\omega \ \omega \ \omega \ \omega - \omega$
 $(\acute{\omega} \ \omega \ \omega \ \omega$ $(\acute{\omega} \ \omega \ \omega - \omega$ $(\acute{\omega} \ \omega \ \omega \ \omega - \omega$
 $(\acute{\omega} \ \omega) \omega \ \omega$ $(\acute{\omega} \ \omega) \omega - \omega$ $(\acute{\omega} \ \omega) \omega \ \omega - \omega$
 $(\acute{\omega} \ \omega) \acute{\omega}) \omega$ $(\acute{\omega} \ \omega) \acute{\omega}) \omega - \omega$

```

\psset{tssym=\omega}

```

```

\def\XS{\bare{\acute\omega}}
\psset{xgap=1.3em,phantomjunctures=true}
\def\JunctureChoose#1{\ifx#1-\hbox{-}\else #1\fi}

\hfil
\vtop{\openuplex
\halign{#\hfil\cr
\asr \X\X\X\X\endasr \cr
\asr '(\XS\X\X\X\endasr \cr
\asr '(\XS\X')\X\X\endasr \cr
\asr '(\XS\X')\XS')\X\endasr \cr
}}
\hfil
\vtop{\openuplex
\halign{#\hfil\cr
\asr \X\X\X'-\X\endasr \cr
\asr '(\XS\X\X'-\X\endasr \cr
\asr '(\XS\X')\X'-\X\endasr \cr
}}
\hfil
\vtop{\openuplex
\halign{#\hfil\cr
\asr \X\X\X\X'-\X\endasr \cr
\asr '(\XS\X\X\X'-\X\endasr \cr
\asr '(\XS\X')\X\X'-\X\endasr \cr
\asr '(\XS\X')\XS')\X'-\X\endasr \cr
}}

```

Appendix A: Parser extensions

Macro: \JunctureChoose

Parameters: juncsep, phantomjunctures

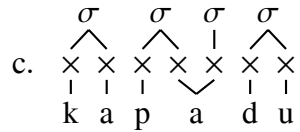
Special parser token: ', <

The use of < described below to displace the position at which a macro is evaluated to the position of the previous timing slot is of general interest. The other extensions are probably of much less interest to most users. They are included because they were developed to meet special needs I have had from time to time. There is a special mechanism for handling timing tier delimiters, in particular, which has hypertrophied. Since it does not consume very many lines of code in *pst-asr.tex*, I have left it in the public version of *pst-asr* in the hopes that it might be useful to someone else.

A.1 Displaced macro evaluation in parsing

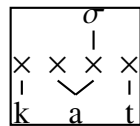
The special parser token < is somewhat more flexible than indicated in Section 5. If [x] follows <, then the following control sequence or active character is evaluated with \xpos incremented by x. So (39a) and (39b) both produce (39c).

- (39) a. `\asr \2ka\2pa:<\1\2du\endasr`
 b. `\asr \2ka\2<[2]\1pa:\2du\endasr`

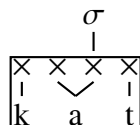


One caution is in order. Displaced evaluation using < does not contribute to determining the bounding box of `\asr... \endasr` since the evaluation takes place inside a group. Compare the following:

- (40) a. `\psframebox[framesep=0]`
`{\asr ka:|\stepxpos{-1}\1\stepxpos{1}|t\endasr}`



- b. `\psframebox[framesep=0]`
`{\asr ka:<\1t\endasr}`



A.2 Parsing timing tier delimiters

Suppose you want to typeset:

- (41) $\begin{array}{cccccc} \times & \times & \# & \times & \times & \times \\ | & | & | & | & | & | \\ t & a & o & g & u & \end{array}$

One way is:

- (42) `\asr[xgap=1.6em] taogu`
`|\tierput(1.5,ts){\#}`
`\endasr`

The parser provides a more convenient way to insert timing tier junctures. If the parser encounters ', it assumes that a juncture follows, which it typesets

midway between the current position and the position of the previous timing slot. So (41) is produced by:

(43) `\asr[xgap=1.6em] ta'\#ogu\endasr`

Actually, before the juncture is typeset, the parser checks to see if a second instance of ' follows. If it does, the second juncture is read and the two junctures typeset as a pair. So:

(44) a. $\begin{array}{ccccc} \times & \times & \times & \times & \times \\ | & | & | & | & | \\ t & a & o & g & u \end{array}$

b. `\asr[xgap=2em] ta'\#\#'\#ogu\endasr`

Extra space will be put in between the junctures if the parameter `juncsep` is set to a positive dimension. The default is 0pt. So:

(45) a. $\begin{array}{ccccc} \times & \times & \times & \times & \times \\ | & | & | & | & | \\ t & a & o & g & u \end{array}$

b. `\asr[xgap=2em,juncsep=.2ex] ta'\#\#'\#ogu\endasr`

Setting `juncsep` to a negative dimension is sometimes useful with certain combinations of junctures for making a more compact display.

Before typesetting the juncture, a substitution table is consulted (encoded in the macro `\JunctureChoose`). *pst-asr* contains:

```
\def\JunctureChoose#1{%
  \ifx#1<\langle\else\ifx#1>\rangle\else#1\fi\fi}
```

This particular substitution table has no particular significance, but it is given to serve as a model for users with special needs to use in defining their own substitution tables. With it:

(46) a. `\asr[xgap=2em] '<ta'>ogu\endasr`

b. $\begin{array}{ccccc} \langle & \times & \times & \rangle & \times & \times & \times \\ | & | & | & | & | & | & | \\ t & a & o & g & u & & \end{array}$

Junctures are always set in math mode.

Finally, the parameter `phantomjunctures` determines whether the junctures enter into the determination of the bounding box of the display. The default is `phantomjunctures=false`. Setting this parameter to true is sometimes helpful if there are multiple displays that need to be left aligned.

A.3 Modifying the parser

A moderately skilled Tex programmer can modify the parser so that it takes special action if certain tokens are encountered. Below is a fragment of the code the parser uses, which gives an idea of how the parser operates.

```

\def\ASR@d{\futurelet\temp\ASR@dd}
\def\ASR@dd{%
  \ifx\temp|\let\next=\ASR@pushasr\else
  \ifx\temp'\let\next=\ASR@juncture\else
  \ifx\temp<\let\next=\ASR@displace\else
  \ifx\temp\bgroup\let\next=\ASR@char \else
  \ifx\temp\endasr\let\next=\ASR@finish \else
  \ifcat\noexpand\temp\noexpand\ASR\let\next=\ASR@cs
  \else \let\next=\ASR@char
  \fi\fi\fi\fi\fi\fi\next
}
\def\ASR@char#1{%
  \def\ASR@hold{#1}%
  \@ifnextchar:\ASR@gem\ASR@charA
}
\def\ASR@charA{\putph{\ASR@hold}\ASR@d}
\def\ASR@gem#1{\putgem{\ASR@hold}\ASR@d}

```

Suppose, for example, that you have frequent need to typeset things like the display below, with “.” marking syllable boundaries:

```

  × × ×.× × ×.× ×
  | | | | | | | |
k a t m a n d u

```

A simple modification of \ASR@dd will allow you to say simply:

```
\asr kat.man.du\endasr
```

First, modify \ASR@dd as shown below, with the modifications boxed.

```

\def\ASR@dd{%
  \ifx\temp|\let\next=\ASR@pushasr\else
  \ifx\temp'\let\next=\ASR@juncture\else
  \ifx\temp<\let\next=\ASR@displace\else
  \ifx\temp.\let\next=\ASR@period\else
  \ifx\temp\bgroup\let\next=\ASR@char \else
  \ifx\temp\endasr\let\next=\ASR@finish \else
  \ifcat\noexpand\temp\noexpand\ASR\let\next=\ASR@cs
  \else \let\next=\ASR@char
  \fi\fi\fi\fi\fi\fi\fi\next
}

```

Then add:

```
\def\ASR@period#1{\tierput[-.5\ASR@gap](\xpos,ts){\bf .}\ASR@d}
```

Appendix B: Utility macros

B.1 Manipulating dimensions

```
\xaddto dimension_register dimension
```

```
\xsettosum dimension_register dimension1 dimension2
```

```
\ysettodiff dimension_register dimension1 dimension2
```

The dimensions can be: 1) dimension registers; 2) dimensions in Tex units or macros that expand to dimensions in Tex units; or numbers or macros that expand to numbers. Numeric dimensions are considered to be measured in psxunits (`\xaddto` and `\xsettosum`) or psyunits (`\ysettodiff`). The dimension register is set as the name of the macro indicates (*dimension*₁ – *dimension*₂ for `\ysettodiff`).

B.2 Interpolating a node between two points

The following macro will interpolate a node between two given points.

```
\interpolate(point1)(point2)% {ratio}{node_name}
```

The points can be in any description recognized by *pst-node* as a `\SpecialCoor` description.