Disclaimer

- The material provided in this document is not my original work and is a summary of some one else's work(s).
- A simple Google search of the title of the document will direct you to the original source of the material.
- I do not guarantee the accuracy, completeness, timeliness, validity, non-omission, merchantability or fitness of the contents of this document for any particular purpose.
-

Chapter # 2: Using HTK

HMM TOOL KIT HTK

Outline

- ✓ Introduction
- ✓ Data Preparation
- ✓ Creating Monophone HMMs
- ✓ Creating Tied-State Triphones
- ✓ Recognizer Evaluation
- ✓ Running the Recognizer Live
- ✗ Adapting the HMMs
- ✗ Semi-Tied and HLDA transforms

Introduction

- The construction of a recognizer for simple voice dialing applications will be described
- The recognizer will be designed to recognize continuously spoken digit strings and a limited set of names

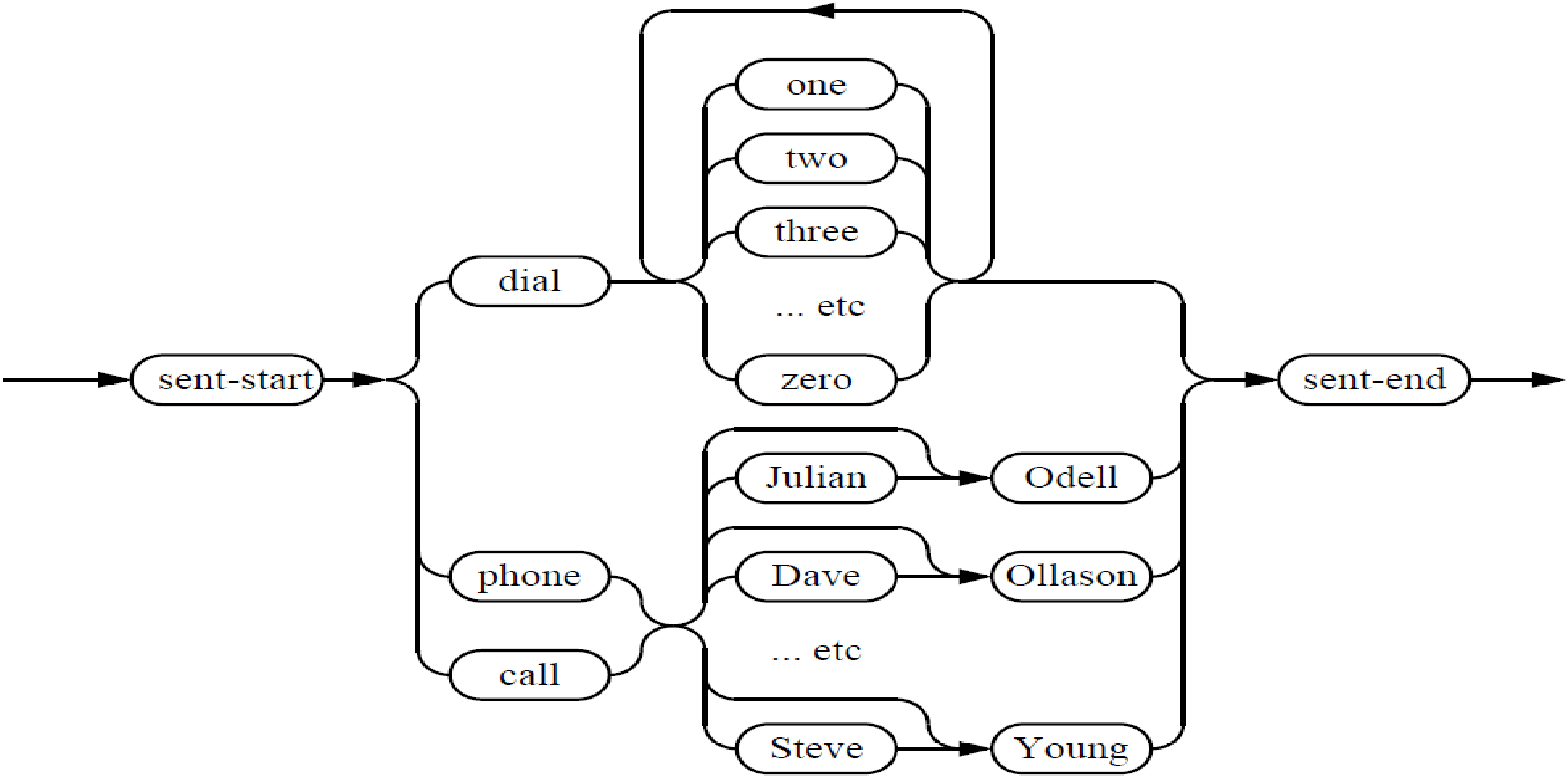
Data Preparation

- For training speech will be recorded from scratch and to do this scripts are needed to prompt for each sentence
- The prompt scripts will be used in conjunction with a pronunciation dictionary to provide the initial phone level transcriptions needed to start the HMM training process
- A task grammar will be used as a random generator for test data
- Before the data can be recorded
  - A phone set must be defined
  - A dictionary must be constructed to cover both training and testing
  - A task grammar must be defined

Step 1 - The Task Grammar

- HTK provides a grammar definition language for specifying simple task grammars
- It consists of a set of variable definitions followed by a regular expression describing the words to recognize
- For the voice dialing application, a suitable grammar might be as

```
$digit = ONE | TWO | THREE | FOUR | FIVE |
         SIX | SEVEN | EIGHT | NINE | OH | ZERO;
$name  = [ JOOP ] JANSEN |
         [ JULIAN ] ODELL |
         [ DAVE ] OLLASON |
         [ PHIL ] WOODLAND |
         [ STEVE ] YOUNG;
( SENT-START ( DIAL <$digit> | (PHONE|CALL) $name) SENT-END )
```

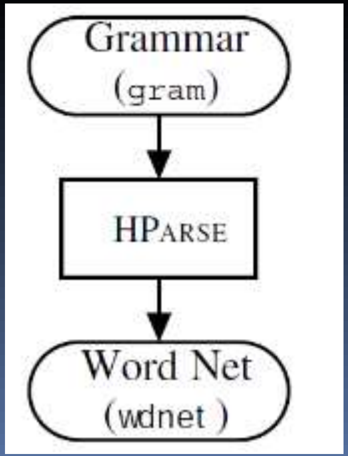



```
$digit = ONE | TWO | THREE | FOUR | FIVE |
         SIX | SEVEN | EIGHT | NINE | OH | ZERO;
$name  = [ JOOP ] JANSEN |
         [ JULIAN ] ODELL |
         [ DAVE ] OLLASON |
         [ PHIL ] WOODLAND |
         [ STEVE ] YOUNG;
( SENT-START ( DIAL <$digit> | (PHONE|CALL) $name) SENT-END )
```

Step 1 - The Task Grammar

- The HTK recognizer requires a word network to be defined using a low level notation called HTK Standard Lattice Format (SLF) in which each word instance and each word-to-word transition is listed explicitly
- This word network can be created automatically from the grammar above using the HParse tool

HParse gram wdnet

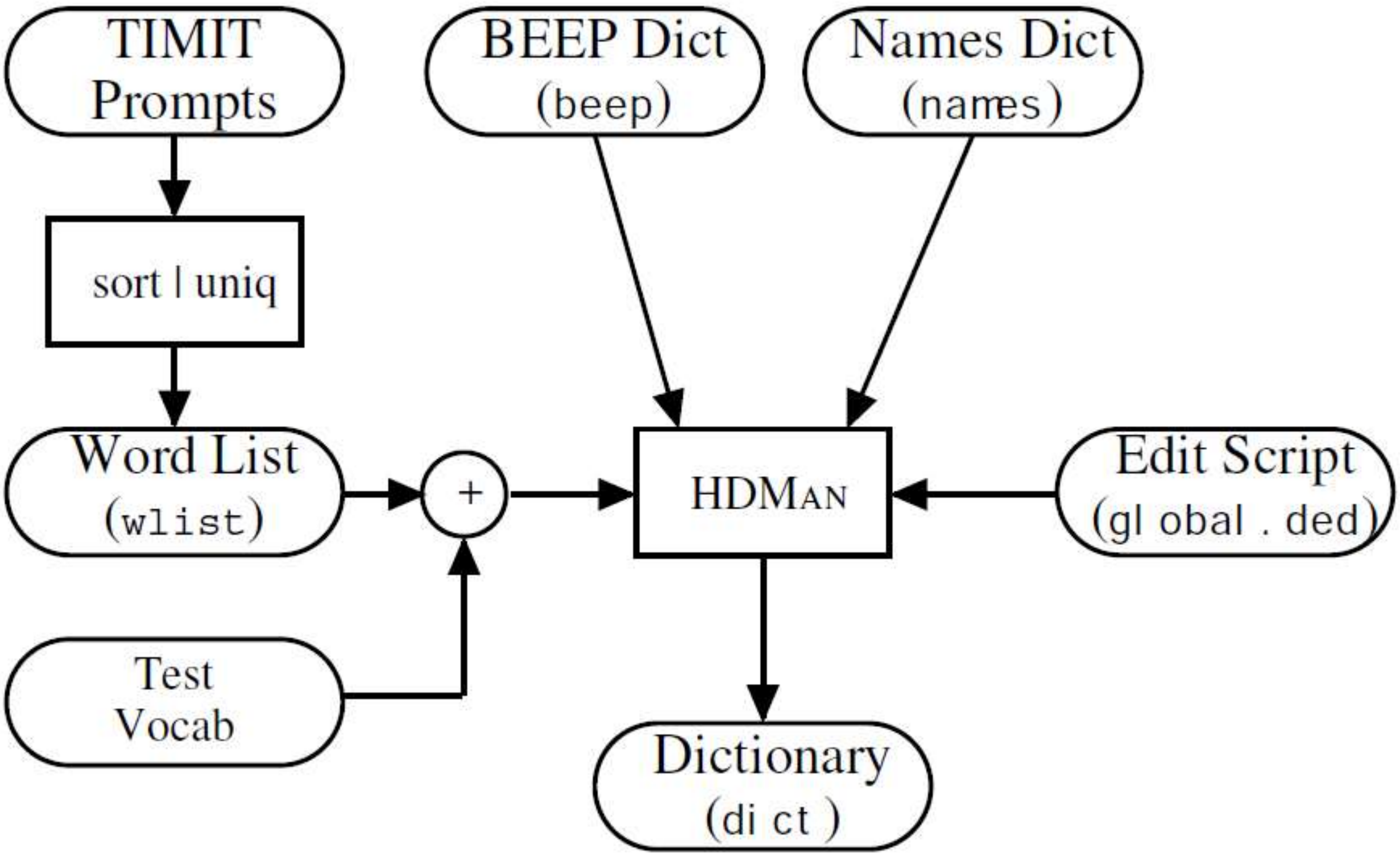


Step 2 - The Dictionary

- The first step in building a dictionary is to create a sorted list of the required words
- For robust acoustic models, the training data will consist of English sentences unrelated to the phone recognition task
- The first few items from the TIMIT DB might be as follows

```
S0001 ONE VALIDATED ACTS OF SCHOOL DISTRICTS
S0002 TWO OTHER CASES ALSO WERE UNDER ADVISEMENT
S0003 BOTH FIGURES WOULD GO HIGHER IN LATER YEARS
S0004 THIS IS NOT A PROGRAM OF SOCIALIZED MEDICINE
etc
```

Step 2 - The Dictionary



Step 2 - The Dictionary

- Create a new dictionary called dict by searching the source dictionaries beep and names to find pronunciations for each word in wlist

```
HDMan -m -w wlist -n monophones1 -l dlog dict beep names
```

- names is a manually constructed file containing pronunciations for the proper names used in the task grammar
- The general format of the dictionary is

WORD [outsym] p1 p2 p3 ....

Step 2 - The Dictionary

A		ah sp
A		ax sp
A		ey sp
CALL		k ao l sp
DIAL		d ay ax l sp
EIGHT		ey t sp
PHONE		f ow n sp
SENT-END	[]	sil
SENT-START	[]	sil
SEVEN		s eh v n sp
TO		t ax sp
TO		t uw sp
ZERO		z ia r ow sp

Step 3 - Recording the Data

- The training and test data will be recorded using the HTK tool HSLab (a combined waveform recording and labelling tool)
- HSLab is invoked by typing

  HSLab noname

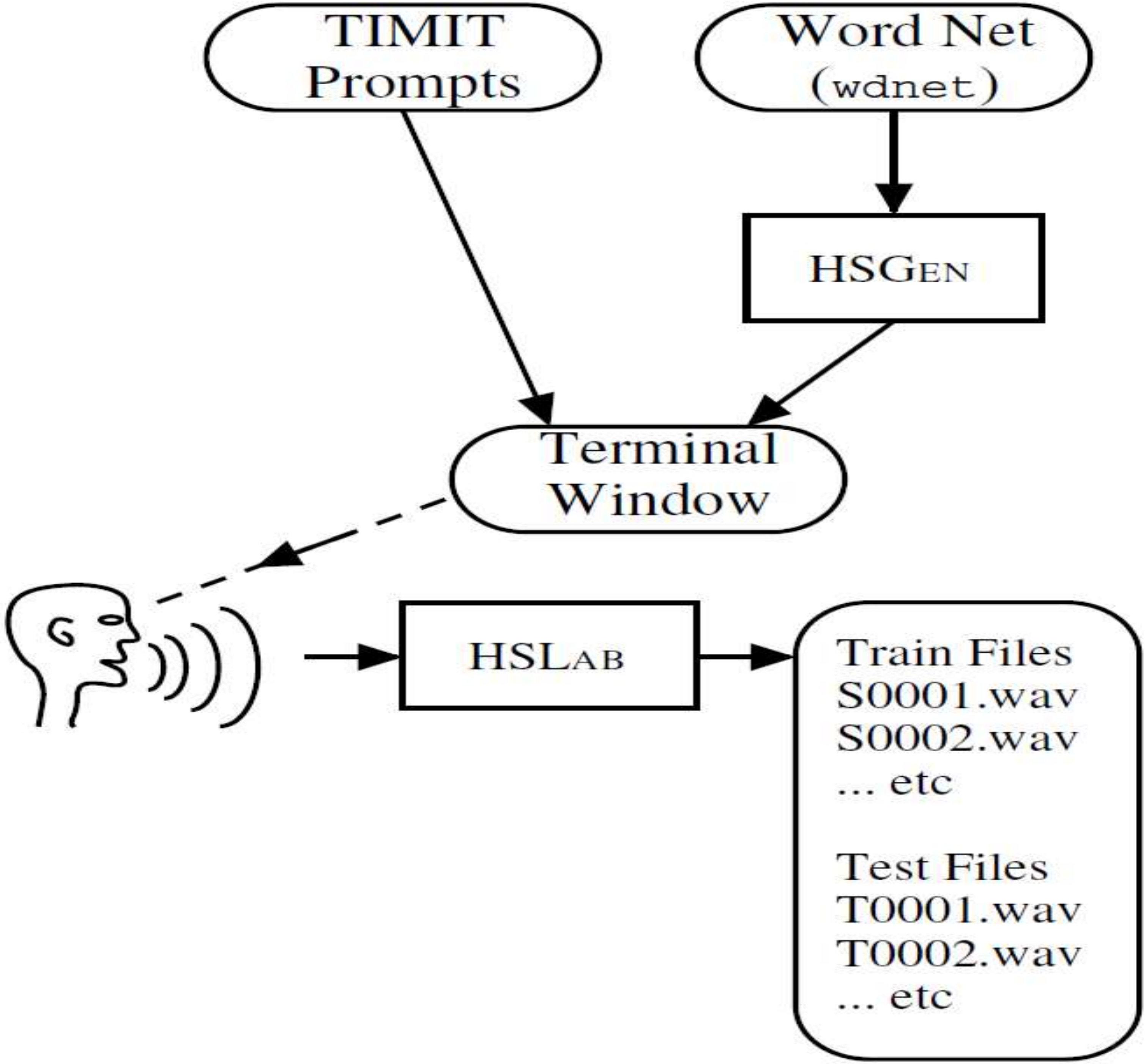
- Each time the record button is pressed, it writes the subsequent recording alternately to a file called noname_0. and to a file called noname_1.
- Thus, it is simple to write a shell script which for each successive line of a prompt file, outputs the prompt, waits for either noname_0. or noname_1. to appear, and then renames the file to the name prepending the prompt

Step 3 - Recording the Data

- The prompts for test sentences need to be generated before recording them
- The tool HSGen can be used to do this by randomly traversing a word network and outputting each word encountered

HSGen -l -n 200 wdnet dict > testprompts

```
1.   PHONE YOUNG
2.   DIAL OH SIX SEVEN SEVEN OH ZERO
3.   DIAL SEVEN NINE OH OH EIGHT SEVEN NINE NINE
4.   DIAL SIX NINE SIX TWO NINE FOUR ZERO NINE EIGHT
5.   CALL JULIAN ODELL
...  etc
```


TIMIT Prompts
Word Net (wdnet)
HSGen
Terminal Window
HSLab
Train Files
S0001.wav
S0002.wav
... etc
Test Files
T0001.wav
T0002.wav
... etc

Step 4 - Creating the Transcription Files

- Every file of training data must have an associated phone level transcription
- An orthographic transcription in HTK label format can be created using a text editor or a scripting language

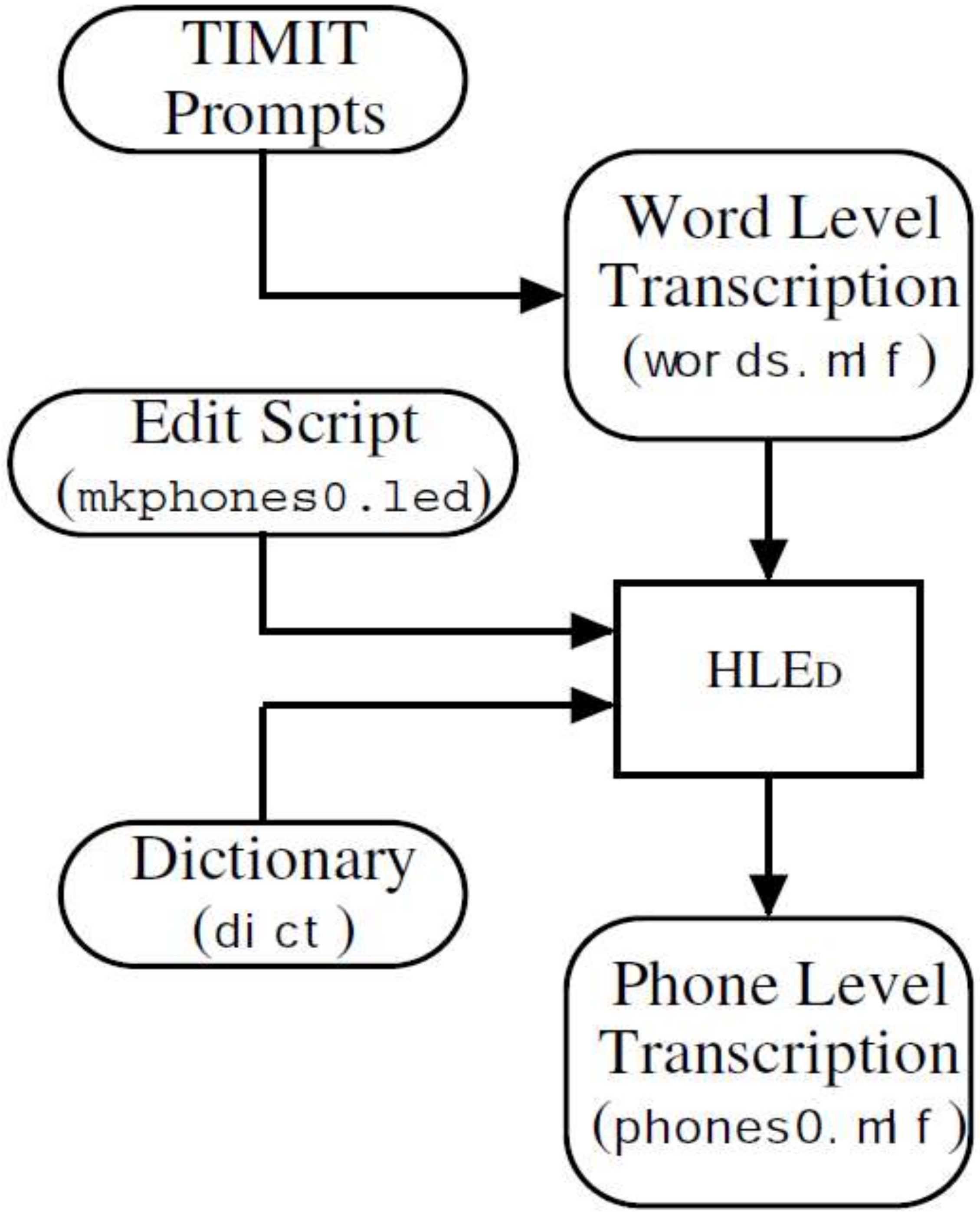
```
#!MLF!#
"*/S0001.lab"
ONE
VALIDATED
ACTS
OF
SCHOOL
DISTRICTS
.
"*/S0002.lab"
TWO
OTHER
CASES
ALSO
WERE
UNDER
ADVISEMENT
.
"*/S0003.lab"
BOTH
FIGURES
(etc.)
```

Step 4 - Creating the Transcription Files

- Once the word level MLF has been created, phone level MLFs can be generated using the label editor HLEd

```
HLEd -l '*' -d dict -i phones0.mlf mkphones0.led words.mlf
```

mkphones0	Output
EX	#!MLF!#
IS sil sil	"*/S0001.lab"
DE sp	sil
	w
	ah
	n
	v
	ae
	l
	ih
	d
	.. etc


TIMIT Prompts
Word Level Transcription (words.mlf)
Edit Script (mkphones0.led)
HLEd
Dictionary (dict)
Phone Level Transcription (phones0.mlf)

Step 5 - Coding the Data

- The final stage of data preparation is to parameterize the raw speech waveforms into sequences of feature vectors
- Coding can be performed using the tool HCopy configured to automatically convert its input into MFCC vectors

```
HCopy -T 1 -C config -S codetr.scp
```

Step 5 -

- The fi ration is to parame veforms into sequen
- Coding e tool HCopy configu its input into MFCC v

```
HCopy codetr.scp
```

```
# Coding parameters
TARGETKIND = MFCC_0
TARGETRATE = 100000.0
SAVECOMPRESSED = T
SAVEWITHCRC = T
WINDOWSIZE = 250000.0
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
ENORMALISE = F
```

Step 5 -

- The fi ration is to parame veforms into sequen
- Coding e tool HCopy configu its input into MFCC v

```
# Coding parameters
TARGETKIND = MFCC_0
TARGETRATE = 100000.0
SAVECOMPRESSED = T
SAVEWITHCRC = T
WINDOWSIZE = 250000.0
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
ENORMALISE = F
```

```
HCopy codetr.scp
```

```
/root/sjy/waves/S0001.wav /root/sjy/train/S0001.mfc
/root/sjy/waves/S0002.wav /root/sjy/train/S0002.mfc
/root/sjy/waves/S0003.wav /root/sjy/train/S0003.mfc
/root/sjy/waves/S0004.wav /root/sjy/train/S0004.mfc
(etc.)
```

Creating Monophone HMMs

- A set of identical monophone HMMs in which every mean and variance is identical is created
- These are then retrained, short-pause models are added and the silence model is extended slightly
- The monophones are then retrained
- The recogniser tool HVite is used to perform a forced alignment, a new phone level MLF is created in which the choice of pronunciations depends on the acoustic evidence
- This new MLF can be used to perform a final re-estimation of the monophone HMMs

Step 6 - Creating Flat Start Monophones

- The first step in HMM training is to define a prototype model
- For phone-based systems, a good topology to use is 3-state left-right with no skips

```
~o <VecSize> 39 <MFCC_0_D_A>
~h "proto"
<BeginHMM>
 <NumStates> 5
 <State> 2
    <Mean> 39
      0.0 0.0 0.0 ...
    <Variance> 39
      1.0 1.0 1.0 ...
 <State> 3
    <Mean> 39
      0.0 0.0 0.0 ...
    <Variance> 39
      1.0 1.0 1.0 ...
 <State> 4
    <Mean> 39
      0.0 0.0 0.0 ...
    <Variance> 39
      1.0 1.0 1.0 ...
 <TransP> 5
  0.0 1.0 0.0 0.0 0.0
  0.0 0.6 0.4 0.0 0.0
  0.0 0.0 0.6 0.4 0.0
  0.0 0.0 0.0 0.7 0.3
  0.0 0.0 0.0 0.0 0.0
<EndHMM>
```

Step 6 - Creating Flat Start Monophones

- The HTK tool HCompV will scan a set of data files, compute the global mean and variance and set all of the Gaussians in a given HMM to have the same mean and variance

```
HCompV -C config -f 0.01 -m -S train.scp -M hmm0 proto
```

- Given this new prototype model stored in the directory hmmo
  - A Master Macro File (MMF) called hmmdefs containing a copy for each of the required monophone HMMs is constructed
  - Relabeling it for each required monophone (including "sil")

Step 6 - Creating Flat Start Monophones

macros

```
~o
   <VecSize> 39
   <MFCC_0_D_A>
~v "varFloor1"
   <Variance> 39
      0.0012 0.0003 ...
```

hmmdefs

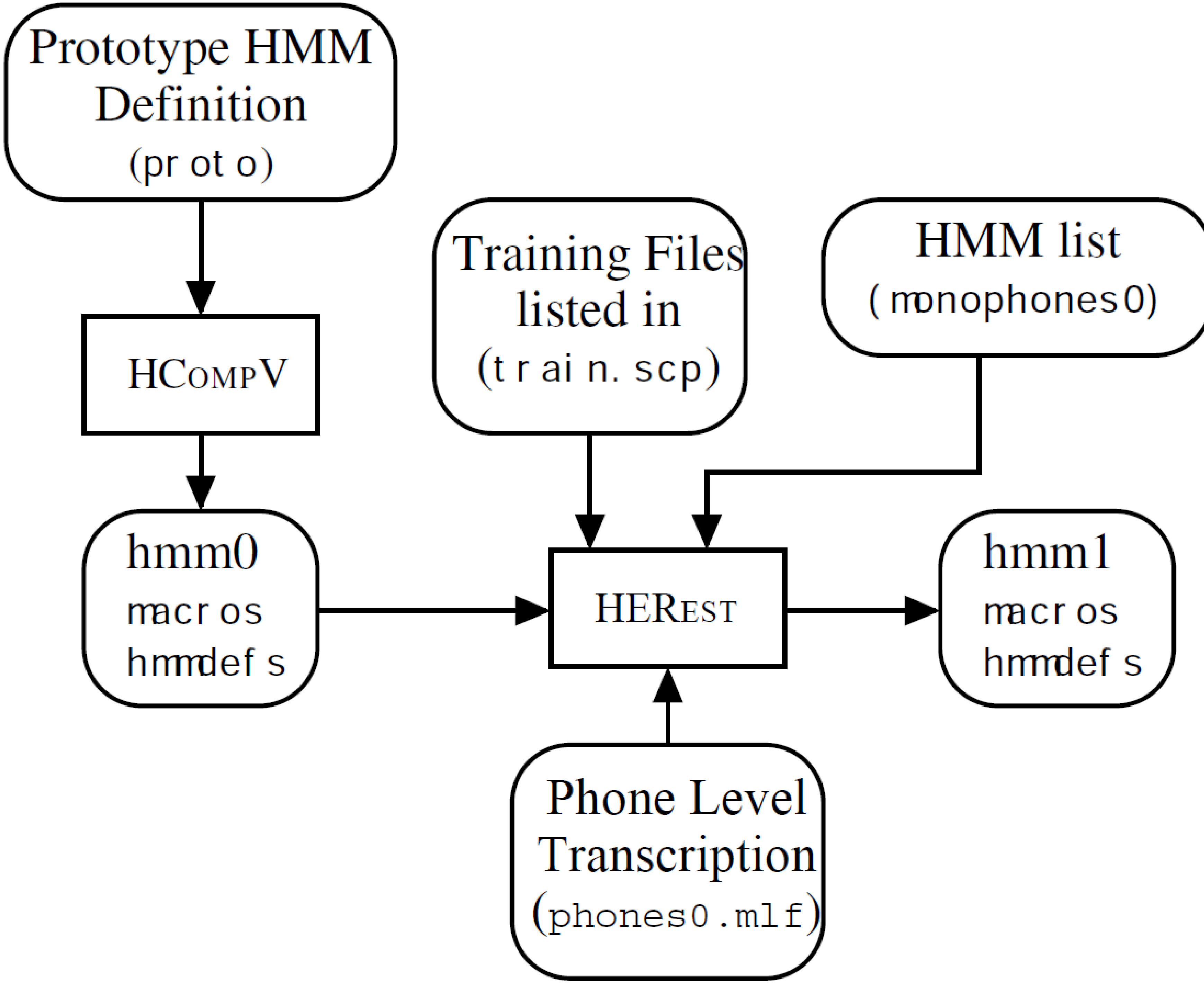
```
~h "aa"
   <BeginHMM> ...
   <EndHMM>
~h "eh"
   <BeginHMM> ...
   <EndHMM>
... etc
```

Step 6 - Creating Flat Start Monophones

- The flat start monophones stored in the directory hmm0 are re-estimated using the embedded re-estimation tool HERest invoked as follows

```
HERest -C config -I phones0.mlf -t 250.0 150.0 1000.0 \
 -S train.scp -H hmm0/macros -H hmm0/hmmdefs -M hmm1 monophones0
```

- This loads all the models in hmm0 which are listed in the model list monophones0
- These are then re-estimated using the data listed in train.scp
- The new model set is stored in the directory hmm1
- -t option sets the pruning thresholds to be used during training
- Pruning limits the range of state alignments that the forward-backward algorithm includes in its summation

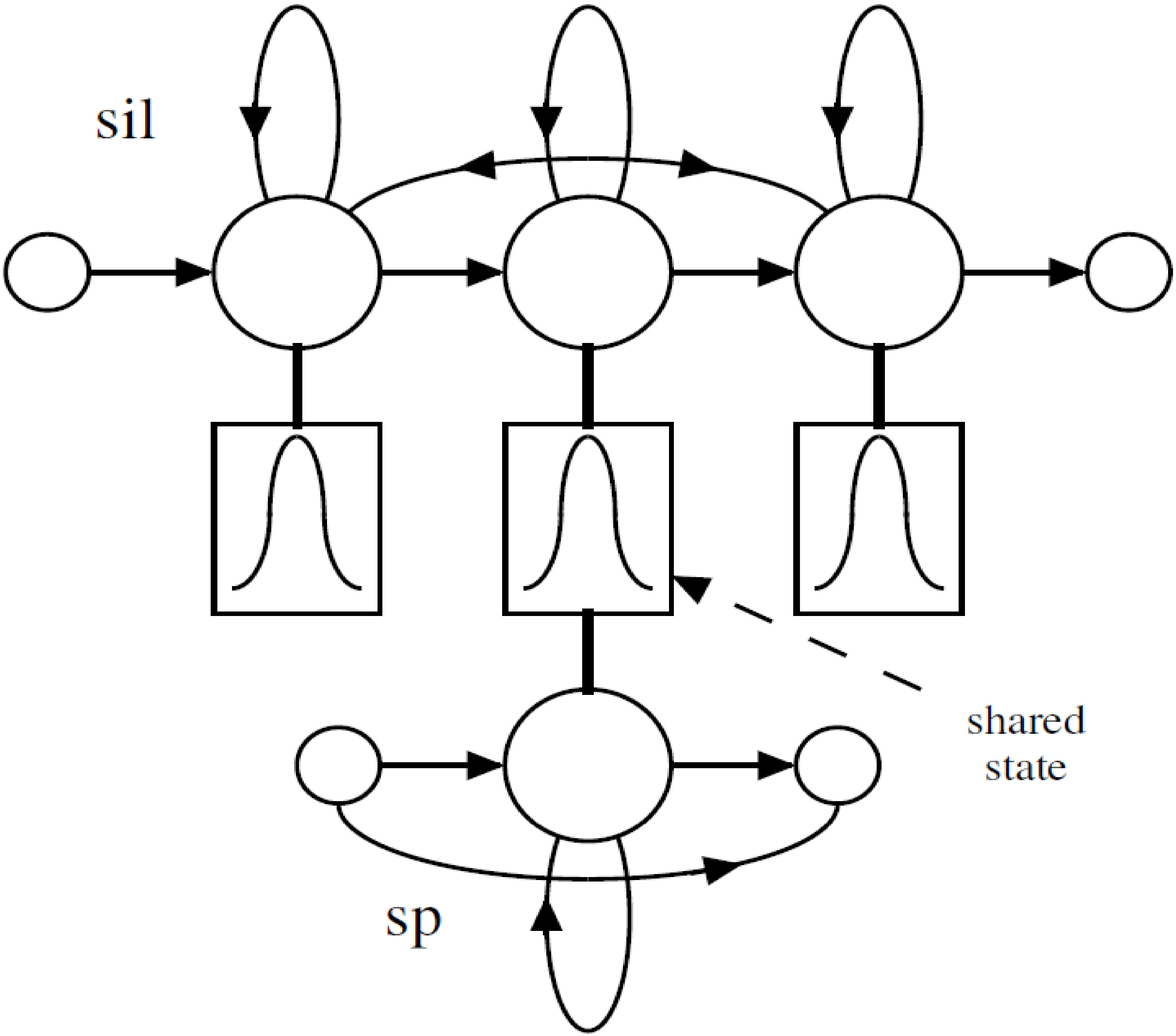

Prototype HMM Definition
(proto)
HCompV
hmm0
macros
hmmdefs
Training Files listed in
(train.scp)
HMM list
(monophones0)
HERest
hmm1
macros
hmmdefs
Phone Level Transcription
(phones0.mlf)

Step 6 - Creating Flat Start Monophones

- Each time HERest is run it performs a single re-estimation
- Each new HMM set is stored in a new directory
- Execution of HERest should be repeated twice more, changing the name of the input and output directories (set with the options -H and -M) each time, until the directory hmm3 contains the final set of initialized monophone HMMs

Step 7 - Fixing the Silence Models

- Extra transitions from states 2 to 4 and from states 4 to 2 in the silence model are added
- The idea is to make the model more robust by allowing individual states to absorb the various impulsive noises in the training data while avoiding transit to the following word
- A 1 state short pause sp model should be created with a direct transition from entry to exit node
- This sp has its emitting state tied to the center state of the silence model


sil
shared
state
sp

Step 7 - Fixing the Silence Models

- These silence models can be created in two stages
  - Use a text editor on the file hmm3/hmmdefs to copy the center state of the sil model to make a new sp model and store the resulting MMF hmmdefs, which includes the new sp model, in the new directory hmm4
  - Run the HMM editor HHEd to add the extra transitions required and tie the sp state to the center sil state
- HHed applies a set of commands in a script to modify a set of HMMs

```
HHEd -H hmm4/macros -H hmm4/hmmdefs -M hmm5 sil.hed monophones1
```

Step 7 - Fixing the Silence Models

```
AT 2 4 0.2 {sil.transP}
AT 4 2 0.2 {sil.transP}
AT 1 3 0.3 {sp.transP}
TI silst {sil.state[3],sp.state[2]}
```

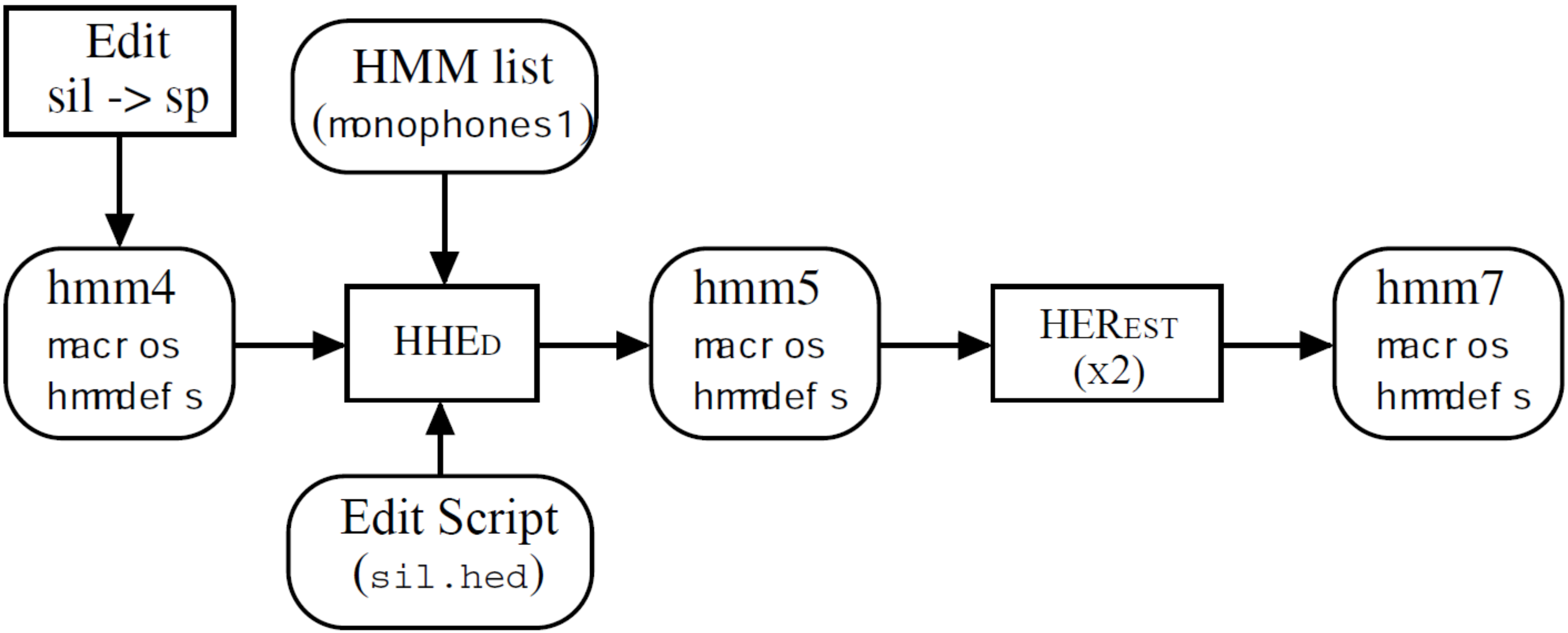
- HHed applies a set of commands in a script to modify a set of HMMs

```
HHEd -H hmm4/macros -H hmm4/hmmdefs -M hmm5 sil.hed monophones1
```

Step 7 - Fixing the Silence Models

- Two passes of HERest are applied using the phone transcriptions with sp models between words
- This leaves the set of monophone HMMs created so far in the directory hmm7

Step 7 - Fixing the Silence Models



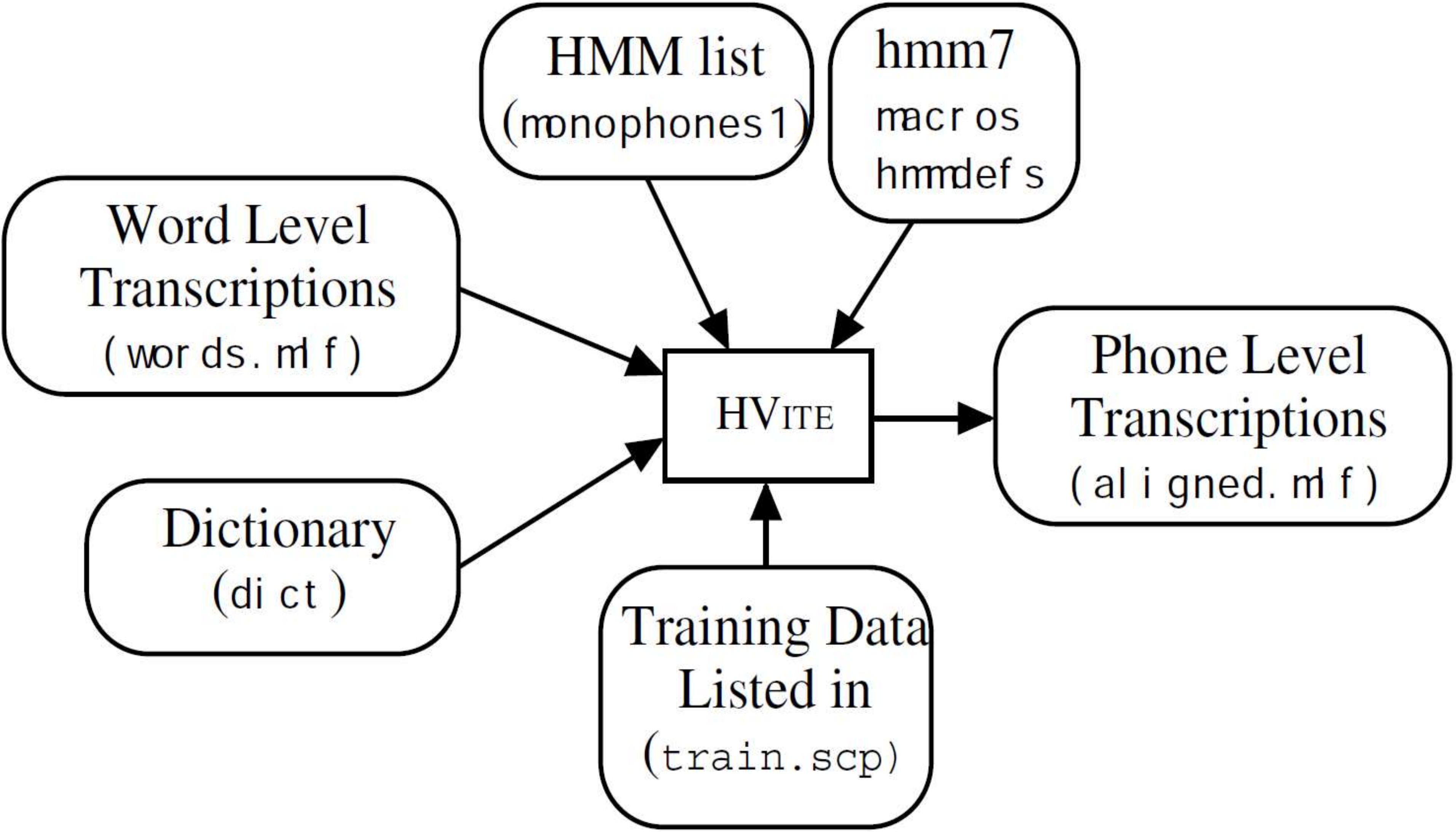
Step 8 - Realigning the Training Data

- The dictionary contains multiple pronunciations for some words, particularly function words
- The phone models created so far can be used to realign the training data and create new transcriptions
- This can be done with a single invocation of the HTK recognition tool Hvite

```
HVite -l '*' -o SWT -b silence -C config -a -H hmm7/macros \
      -H hmm7/hmmdefs -i aligned.mlf -m -t 250.0 -y lab \
      -I words.mlf -S train.scp  dict monophones1
```

- Once the new phone alignments have been created, another 2 passes of HERest can be applied to re-estimate the HMM set parameters again

Step 8 - Realigning the Training Data



Creating Tied-State Triphones

- Given a set of monophone HMMs, the final stage of model building is to create context-dependent triphone HMMs
  - The monophone transcriptions are converted to triphone transcriptions and a set of triphone models are created by copying the monophones and re-estimating
  - Similar acoustic states of these triphones are tied to ensure that all state distributions can be robustly estimated

Step 9 - Making Triphones from Monophones

- Context-dependent triphones can be made by simply cloning monophones and then re-estimating using triphone transcriptions
- To convert the monophone transcriptions in aligned.mlf to an equivalent set of triphone transcriptions in wintri.mlf we use

```
HLEd -n triphones1 -l '*' -i wintri.mlf mktri.led aligned.mlf
```

- At the same time, a list of triphones is written to the file triphones1

```
mktri.led
WB sp
WB sil
TC
```

```
sil th ih s sp m ae n sp ...
sil th+ih th-ih+s ih-s sp m+ae m-ae+n ae-n sp ...
```

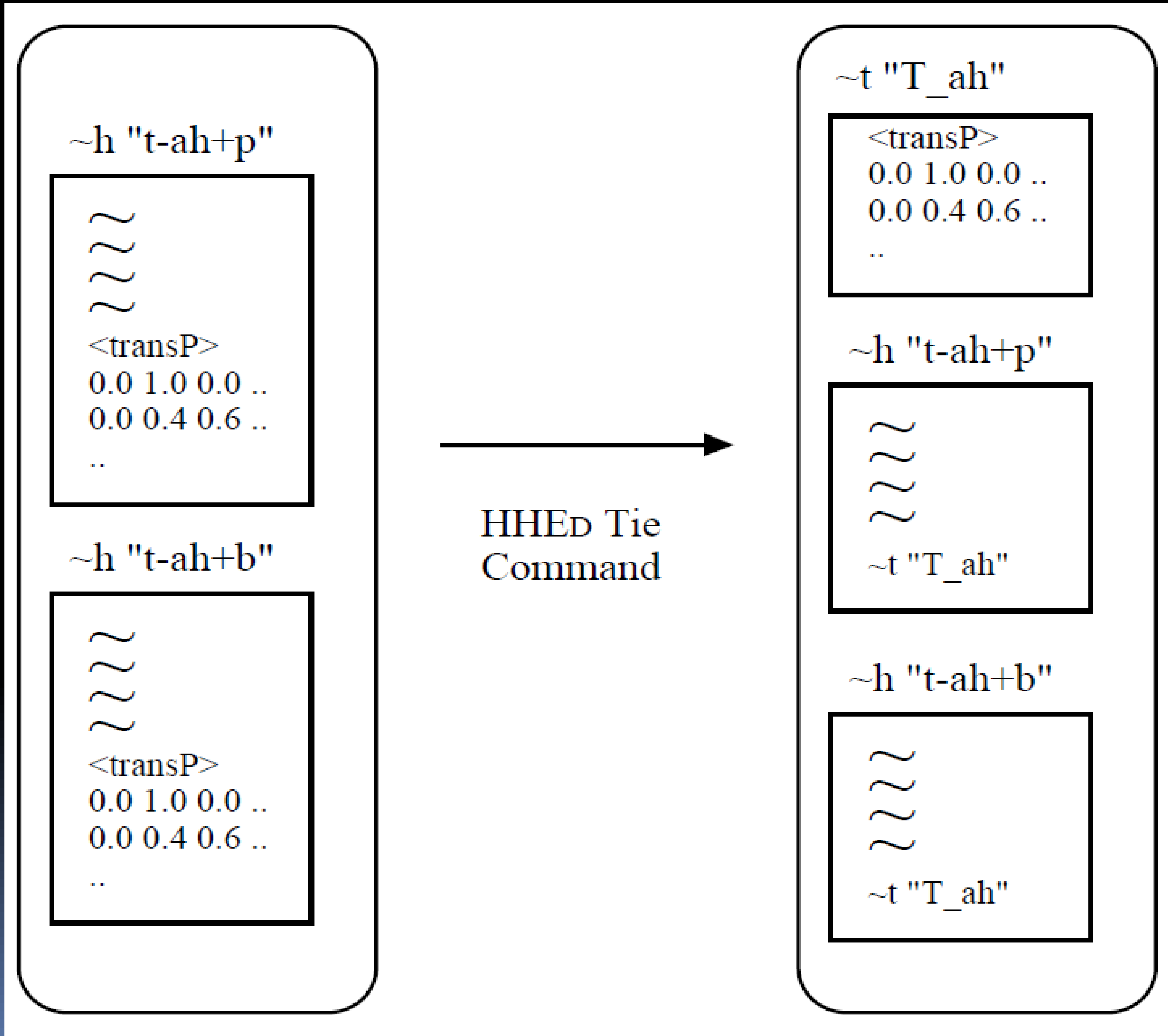
Step 9 - Making Triphones from Monophones

- The cloning of models can be done efficiently using the HMM editor HHEd

```
HHEd -B -H hmm9/macros -H hmm9/hmmdefs -M hmm10
     mktri.hed monophones1
```

- The edit script mktri.hed contains a clone command CL followed by TI commands to tie all of the transition matrices in each triphone set

```
CL triphones1
TI T_ah {(*-ah+*,ah+*,*-ah).transP}
TI T_ax {(*-ax+*,ax+*,*-ax).transP}
TI T_ey {(*-ey+*,ey+*,*-ey).transP}
TI T_b {(*-b+*,b+*,*-b).transP}
TI T_ay {(*-ay+*,ay+*,*-ay).transP}
...
```


~h "t-ah+p"
<transP>
0.0 1.0 0.0 ..
0.0 0.4 0.6 ..
..
~h "t-ah+b"
<transP>
0.0 1.0 0.0 ..
0.0 0.4 0.6 ..
..
HHEd Tie Command
~t "T_ah"
<transP>
0.0 1.0 0.0 ..
0.0 0.4 0.6 ..
..
~h "t-ah+p"
~t "T_ah"
~h "t-ah+b"
~t "T_ah"

Step 9 - Making Triphones from Monophones

- The cloning of models can be done efficiently using the HMM editor HHEd

```
HHEd -B -H hmm9/macros -H hmm9/hmmdefs -M hmm10
     mktri.hed monophones1
```

- The edit script mktri.hed contains a clone command CL followed by TI commands to tie all of the transition matrices in each triphone set

```
CL triphones1
TI T_ah {(*-ah+*,ah+*,*-ah).transP}
TI T_ax {(*-ax+*,ax+*,*-ax).transP}
TI T_ey {(*-ey+*,ey+*,*-ey).transP}
TI T_b {(*-b+*,b+*,*-b).transP}
TI T_ay {(*-ay+*,ay+*,*-ay).transP}
...
```

Step 9 - Making Triphones from Monophones

- The cloning of models can be done efficiently using the HMM editor HHEd

```
HHEd -B -H hmm9/macros -H hmm9/hmmdefs -M hmm10
      mktri.hed monophones1
```

- The edit script mktri.hed contains a clone command CL followed by TI commands to tie all of the transition matrices in each triphone set

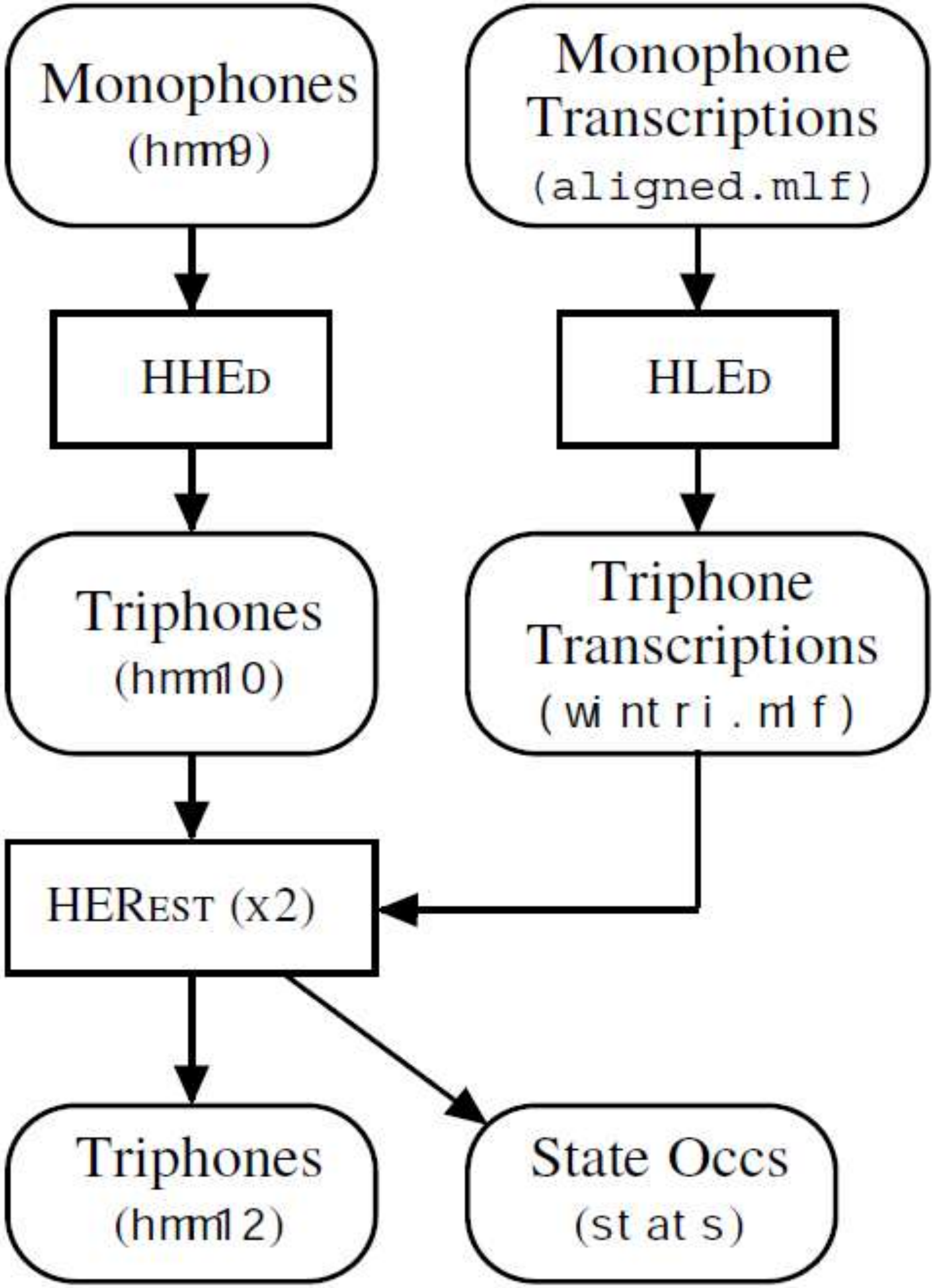
What is the difference between T_ah and ah?

```
CL triphones1
TI T_ah {(*-ah+*,ah+*,*-ah).transP}
TI T_ax {(*-ax+*,ax+*,*-ax).transP}
TI T_ey {(*-ey+*,ey+*,*-ey).transP}
TI T_b {(*-b+*,b+*,*-b).transP}
TI T_ay {(*-ay+*,ay+*,*-ay).transP}
...
```

Step 9 - Making Triphones from Monophones

- The transition parameters do not vary significantly with acoustic context but nevertheless need to be estimated accurately
- Once the context-dependent models have been cloned, the new triphone set can be re-estimated using HERest
- For the final pass of HERest, the -s option should be used to generate a file of state occupation statistics called stats.

```
HERest -B -C config -I wintri.mlf -t 250.0 150.0 1000.0 -s stats \
 -S train.scp -H hmm11/macros -H hmm11/hmmdefs -M hmm12 triphones1
```


Monophones
(hmm9)
Monophone
Transcriptions
(aligned.mlf)
HHEd
HLEd
Triphones
(hmm10)
Triphone
Transcriptions
(wintri.mlf)
HERest (x2)
Triphones
(hmm12)
State Occs
(stats)

Step 10 - Making Tied-State Triphones

- The outcome of the previous stage is a set of triphone HMMs with all triphones in a phone set sharing the same transition matrix
- When estimating these models, many of the variances in the output distributions will have been floored since there will be insufficient data associated with many of the states
- The last step in the model building process is to tie states within triphone sets in order to share data and thus be able to make robust parameter estimates
- The performance of the recognizer depends crucially on how accurate the state output distributions capture the statistics of the speech data

Step 10 - Making Tied-State Triphones

- HHEd provides two mechanisms which allow states to be clustered and then each cluster tied
  - Data-driven: using a similarity measure between states
  - Decision trees: based on asking questions about the left and right contexts of each triphone
- Decision tree state tying is performed by running HHEd as follows

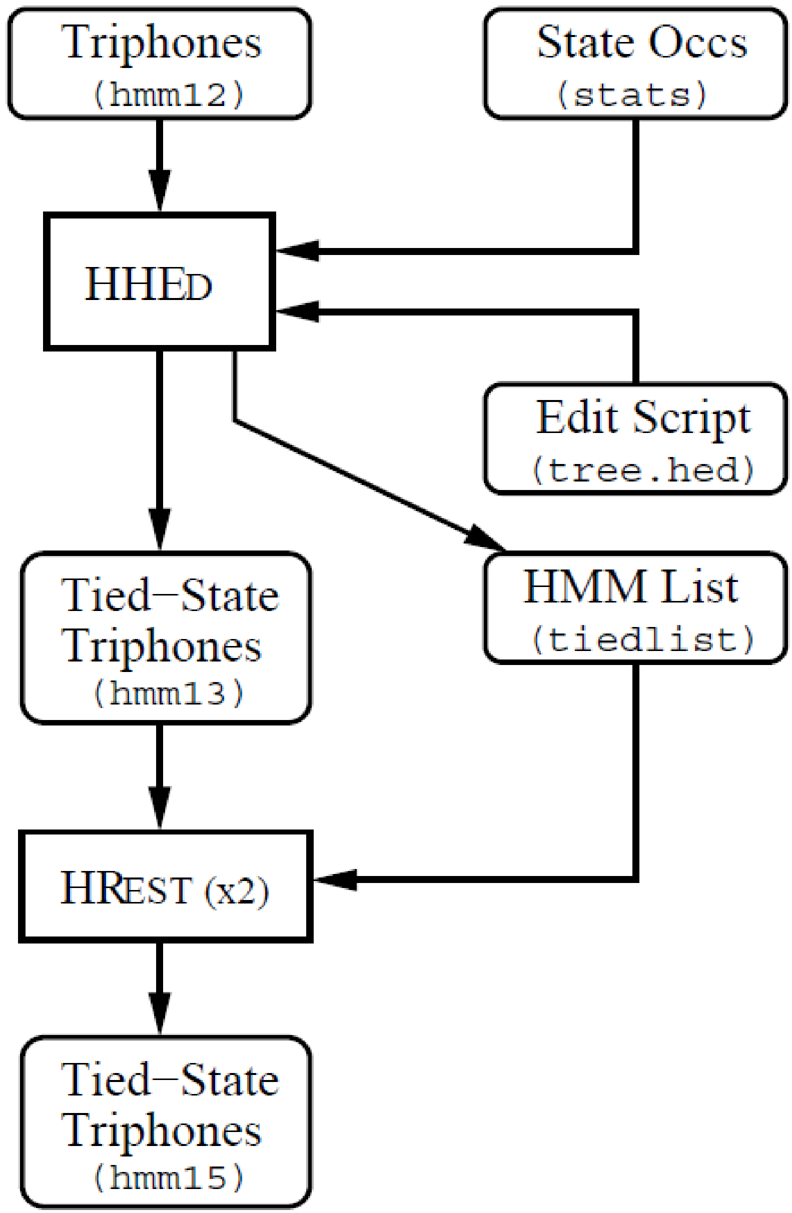
```
HHEd -B -H hmm12/macros -H hmm12/hmmdefs -M hmm13 \
     tree.hed triphones1 > log
```

Step 10 - Making Tied-State Triphones

```
RO 100.0 stats
TR 0
QS "L_Class-Stop" {p-*,b-*,t-*,d-*,k-*,g-*}
QS "R_Class-Stop" {*+p,*+b,*+t,*+d,*+k,*+g}
QS "L_Nasal" {m-*,n-*,ng-*}
QS "R_Nasal" {*+m,*+n,*+ng}
QS "L_Glide" {y-*,w-*}
QS "R_Glide" {*+y,*+w}
....
QS "L_w" {w-*}
QS "R_w" {*+w}
QS "L_y" {y-*}
QS "R_y" {*+y}
QS "L_z" {z-*}
QS "R_z" {*+z}
```

```
TR 2

TB 350.0 "aa_s2" {(aa, *-aa, *-aa+*, aa+*).state[2]}
TB 350.0 "ae_s2" {(ae, *-ae, *-ae+*, ae+*).state[2]}
TB 350.0 "ah_s2" {(ah, *-ah, *-ah+*, ah+*).state[2]}
TB 350.0 "uh_s2" {(uh, *-uh, *-uh+*, uh+*).state[2]}
....
TB 350.0 "y_s4" {(y, *-y, *-y+*, y+*).state[4]}
TB 350.0 "z_s4" {(z, *-z, *-z+*, z+*).state[4]}
TB 350.0 "zh_s4" {(zh, *-zh, *-zh+*, zh+*).state[4]}

TR 1

AU "fulllist"
CO "tiedlist"

ST "trees"
```

Step 10 - Making Tied-State Triphones



Recogniser Evaluation

- The recognizer is now complete and its performance can be evaluated
- The recognition network and dictionary have already been constructed, and test data has been recorded
- All that is necessary is to run the recogniser and then evaluate the results using the HTK analysis tool HResults

Step 11 - Recognizing the Test Data

- Assuming that test.scp holds a list of the coded test files, then each test file will be recognised and its transcription output to an MLF called recout.mlf

```
HVite -H hmm15/macros -H hmm15/hmmdefs -S test.scp \
      -l '*' -i recout.mlf -w wdnet \
      -p 0.0 -s 5.0 dict tiedlist
```

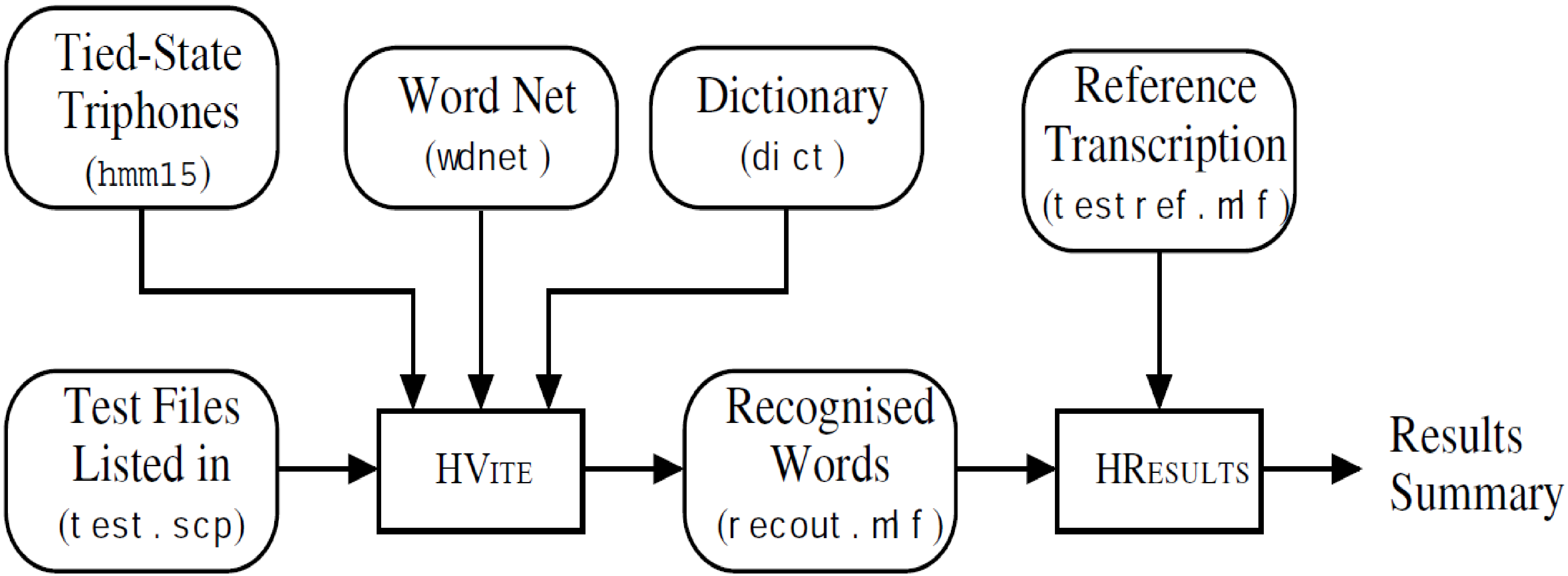
- The options -p and -s set the word insertion penalty and the grammar scale factor
- Assuming that the MLF testref.mlf contains word level transcriptions for each test file

Step 11 - Recognizing the Test Data

```
HResults -I testref.mlf tiedlist recout.mlf

====================== HTK Results Analysis ===============
  Date: Sun Oct 22 16:14:45 1995
  Ref : testrefs.mlf
  Rec : recout.mlf
------------------------ Overall Results -----------------
SENT: %Correct=98.50 [H=197, S=3, N=200]
WORD: %Corr=99.77, Acc=99.65 [H=853, D=1, S=1, I=1, N=855]
==========================================================
```

- Assuming that the MLF testref.mlf contains word level transcriptions for each test file

Step 11 - Recognizing the Test Data



Running the Recogniser Live

- The recognizer can also be run with live input

```
# Waveform capture
SOURCERATE=625.0
SOURCEKIND=HAUDIO
SOURCEFORMAT=HTK
ENORMALISE=F
USESILDET=T
MEASURESIL=F
OUTSILWARN=T
```

```
HVite -H hmm15/macros -H hmm15/hmmdefs -C config2 \
      -w wdnet -p 0.0 -s 5.0 dict tiedlist
```

Thank You