# Step of Decoding-graph Creation on Test Time by Kaldi Toolkit

- STEP 1 Preparing the initial symbol table words.txt and phones.txt

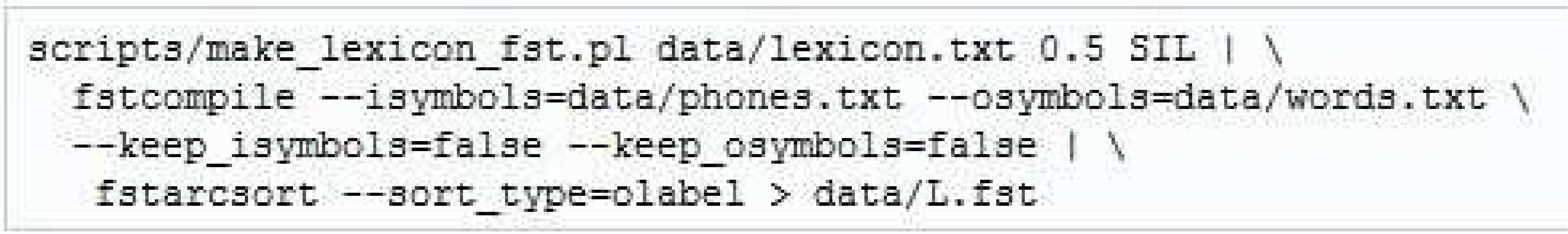
(1)words.txt contains ƹ “#0”

(2)phones.txt doesn’t contain ƹ,but after create L.fst, ƹ in phones_disambig.txt

- STEP 2 Preparing the lexicon L

(1)Lexicon will be used to create L.fst which used in training(No disambiguation symbols );lexcion created with disambiguation symbols used in decoding-graph creation

(2)Convert the lexicon without disambiguation symbols into an FST.



```
scripts/make_lexicon_fst.pl data/lexicon.txt 0.5 SIL | \
  fstcompile --isymbols=data/phones.txt --osymbols=data/words.txt \
  --keep_isymbols=false --keep_osymbols=false | \
   fstarcsort --sort_type=olabel > data/L.fst
```

The output of silence with probability 0.5

(3)Structure of lexicon

Final: one state(“loop state”)

Start:two transition to loop(silence & no silence)

Loop state:input –the first phone of a word

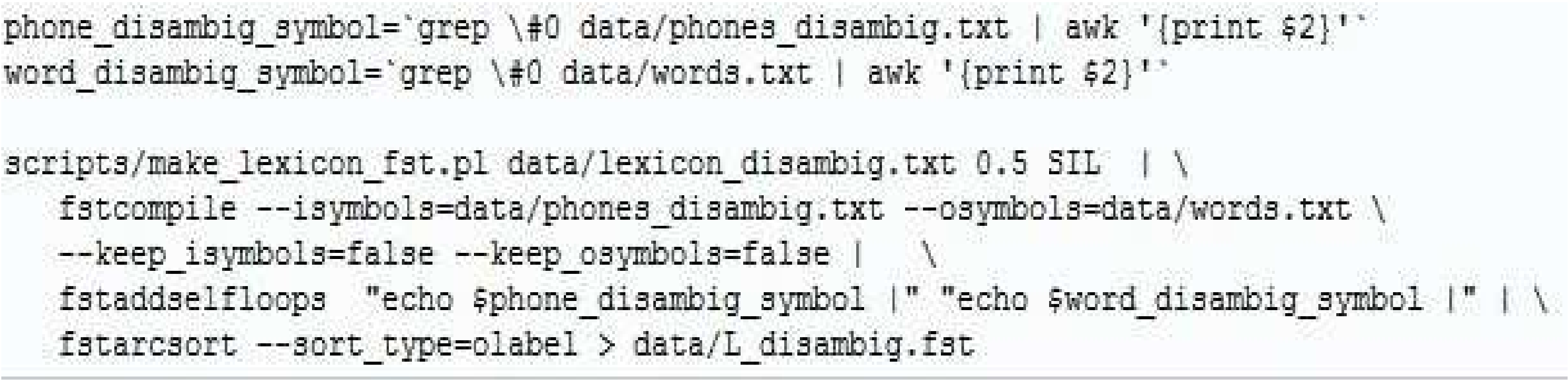
output—the word

## (4)Create lexicon with disambiguation symbols

Add self-loops to the lexicon so disambiguation symbols #0 from G.fst can be passed through the lexicon.

Two ways: program fstaddselfloops

script make_lexicon_fst.pl



```
phone_disambig_symbol=`grep \#0 data/phones_disambig.txt | awk '{print $2}'`
word_disambig_symbol=`grep \#0 data/words.txt | awk '{print $2}'`

scripts/make_lexicon_fst.pl data/lexicon_disambig.txt 0.5 SIL  | \
  fstcompile --isymbols=data/phones_disambig.txt --osymbols=data/words.txt \
  --keep_isymbols=false --keep_osymbols=false |   \
  fstaddselfloops  "echo $phone_disambig_symbol |" "echo $word_disambig_symbol |" | \
  fstarcsort --sort_type=olabel > data/L_disambig.fst
```

- STEP 3 Preparing the grammar G

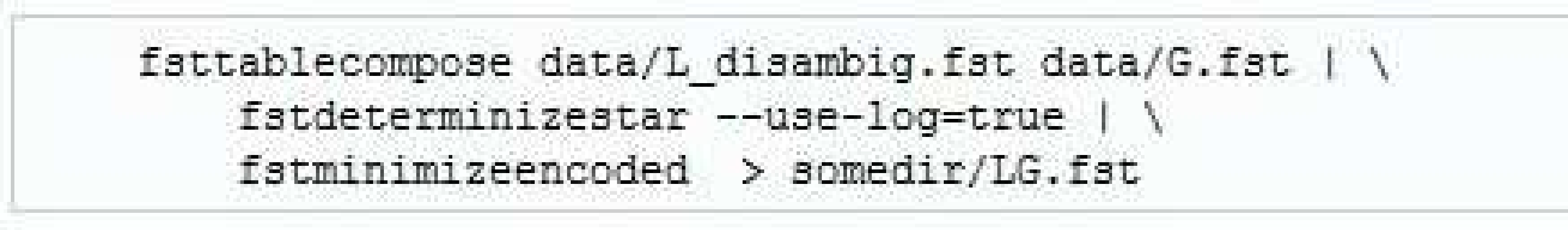
The grammar G is for the most part an acceptor (i.e. input and output symbols are identical on each arc) with words as its symbols.

Exception--the disambiguation symbol #0 only appears on the input side

steps running arpa2fst:

- remove the embedded symbols from the FST
- make sure there are no out-of-vocabulary words in the language model
- remove "illegal" sequences of the start and end-of-sentence symbols
- replace epsilons on the input side with the special disambiguation symbol #0.

- STEP 4 Preparing LG



```
    fsttablecompose data/L_disambig.fst data/G.fst | \
        fstdeterminizestar --use-log=true | \
        fstminimizeencoded  > somedir/LG.fst
```

(1) composing L with G

(2)remove ε

(3)minimization: the same as minimization algorithm that applies to weighted acceptors; the only change relevant here is that it avoids pushing weights, hence preserving stochasticity

- STEP 5 Preparing CLG

  Prepare an FST called CLG to get a transducer whose inputs are context-dependent phones.

(1)Making the context transducer.

The basic structure of C is that it has states for all possible phone windows of size N-1.

Beginning of utterance

Suppose: state <eps>/<eps> output symbol a

so the input is <eps>/<eps>/a

when P=1,the central element is <eps>

so , let input of arc be #-1

End of utterance :The context FST has, on the right (its output side), a special symbol $ that occurs at the end of utterances.

e.g. a/b/<eps>     <eps> represents undefined context

Natural way: have a transition with

input          a/b/<eps>

output        <eps>

from state a/b to final state.

Instead:(1) use $ as the end-of-utterance symbol

(2) make sure it appears once at the end of each path in LG

(3) replace <eps> with $ on the output of C and the number of repetitions of $ is equal to N-P-1.

Achieved by: function AddSubsequentialloop( )

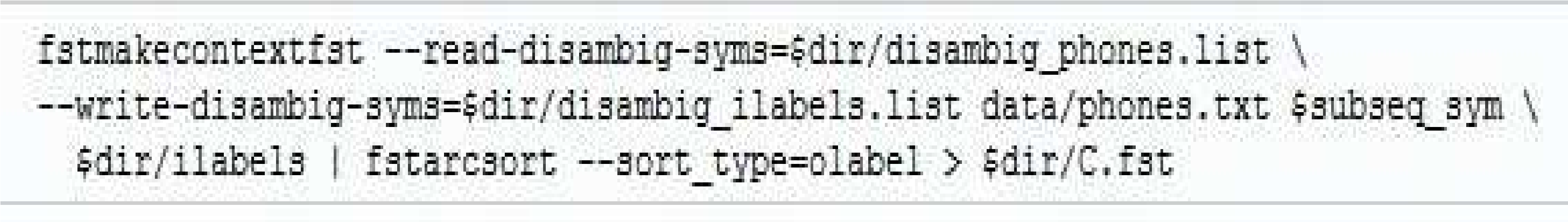
program fstaddsubsequentialloop

If we wanted C on its own, need:

(1)a list of disambiguation symbols;

(2)work out an unused symbol id use for the subsequential symbol

We could then create C with the following command



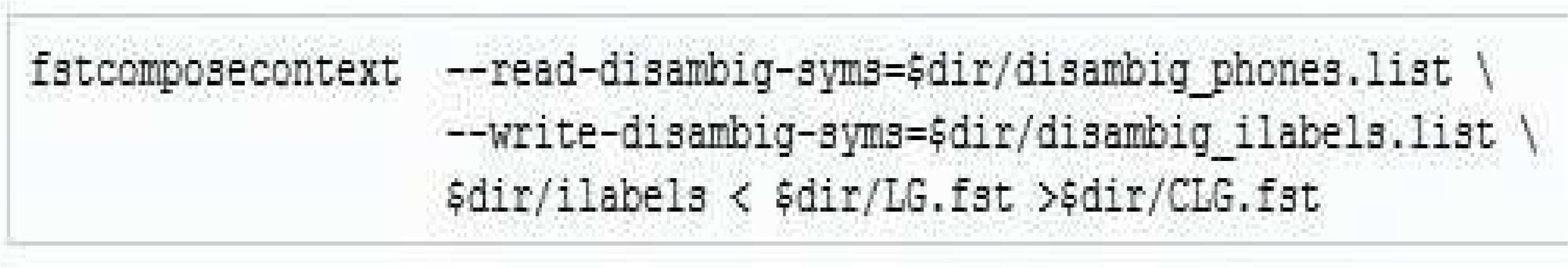
```
fstmakecontextfst --read-disambig-syms=$dir/disambig_phones.list \
--write-disambig-syms=$dir/disambig_ilabels.list data/phones.txt $subseq_sym \
  $dir/ilabels | fstarcsort --sort_type=olabel > $dir/C.fst
```

Need: a list of phones;

a list of disambiguation symbols;

id of the subsequential symbols.

(2)Composing with C dynamically-- use program fstcomposecontext



```
fstcomposecontext  --read-disambig-syms=$dir/disambig_phones.list \
                   --write-disambig-syms=$dir/disambig_ilabels.list \
                   $dir/ilabels < $dir/LG.fst >$dir/CLG.fst
```

(3) Reducing the number of context-dependent input symbols.

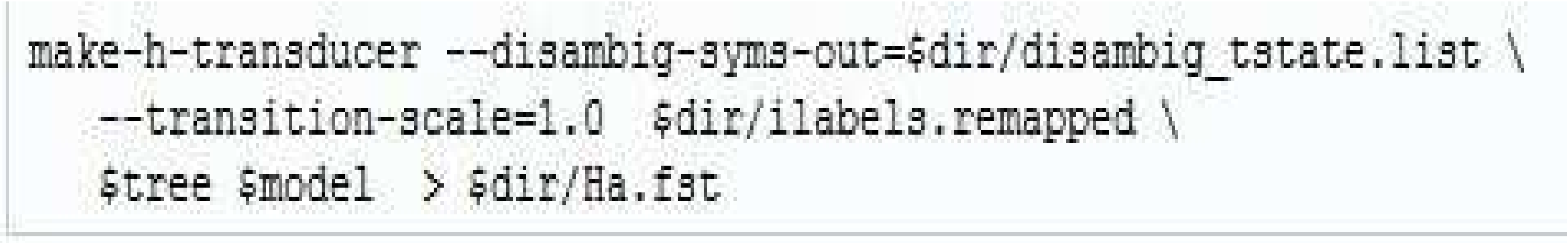
After creating CLG.fst, there is an optional graph creation stage that can reduce its size. Use program make-ilable-transducer and output a new ilable_info(5%-20% reduction).

- STEP 6 Making the H transducer

H:input transition-id(encodes the pdf-id plus some other information including the phone).

output context-dependent phones

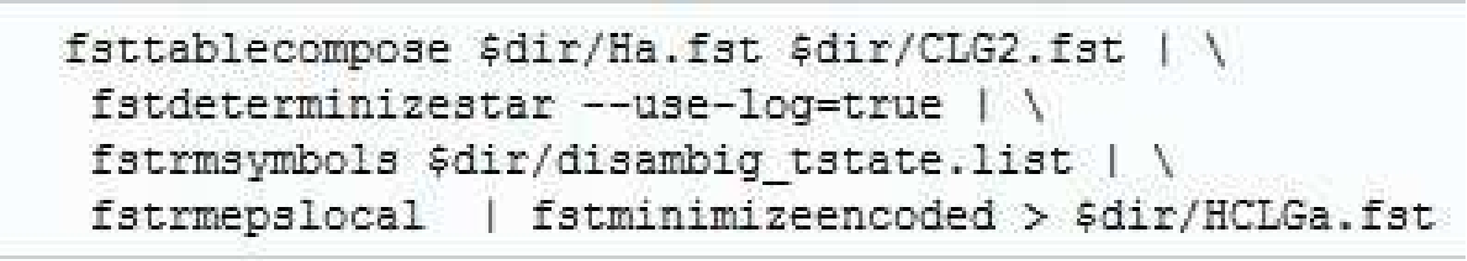
Script that makes the H transducer



```
make-h-transducer --disambig-syms-out=$dir/disambig_tstate.list \
   --transition-scale=1.0  $dir/ilabels.remapped \
   $tree $model  > $dir/Ha.fst
```

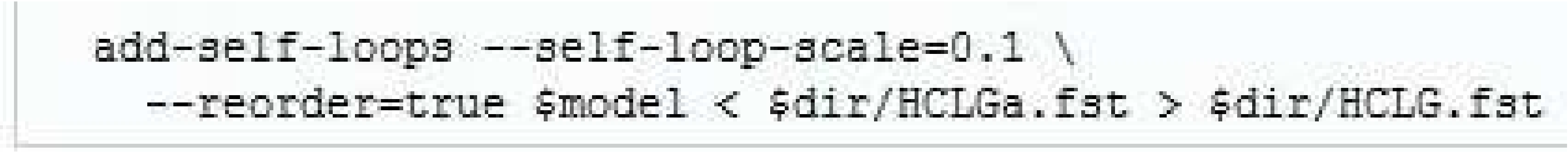
Called Ha.fst because it lacks self-loops.

- STEP 7 Make the HCLG that lacks self-loops.



```
  fsttablecompose $dir/Ha.fst $dir/CLG2.fst | \
   fstdeterminizestar --use-log=true | \
   fstrmsymbols $dir/disambig_tstate.list | \
   fstrmepslocal  | fstminimizeencoded > $dir/HCLGa.fst
```

- STEP 8 Adding self-loops to HCLG



```
  add-self-loops --self-loop-scale=0.1 \
    --reorder=true $model < $dir/HCLGa.fst > $dir/HCLG.fst
```

The self-loop scale is the scale that we apply to the self-loops add a self-loop with log-probability self-loop-scale * log(p), and add (self-loop-scale * log(1-p)) to all the other log transition probabilities out of that state