OpenFst: a General and Efficient Weighted Finite-State Transducer Library

Part II. *Applications*

Common Elements of FST Applications

- Models
  - Constructed
    - Direct automata specification
    - Regular expressions
    - Rewrite rules
    - Context-dependency transducer
    - Rule-based grammars
  - Learned
    - n-gram language models
    - Pair n-gram language models
    - EM-estimated models (known topology)
- Cascades and Search
  - Composition and intersection
  - Shortest path and shortest distance
  - FST optimization

Example Applications

(These are drawn from www.openfst.org/FstExamples)

Example Domain and Data

The common data files used in the examples are all available from `www.openfst.org`.

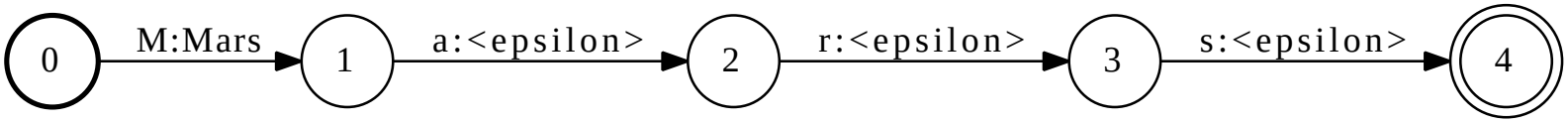
wotw.txt	text of H.G. Well's *War of the Worlds*
wotw_lm.fst	5-gram language model FST for wotw.txt
wotw.syms	FST symbol table file for wotw_lm.fst
ascii.syms	FST symbol table file for ASCII letters

- With these files and the descriptions that follow, the reader should be able to repeat the examples.
- With about 340,000 words in *The War of the Worlds*, it is a small, public-domain corpus that allows non-trivial examples.
- The n-gram model was built with *OpenGrm* (soon to be available at www.opengrm.org).

Tokenization - I

- Converts sequence of ASCII characters into word tokens with punctuation and whitespace stripped.
- Create a *lexicon transducer* that maps from letters to their corresponding word tokens.
- Begin with a lexicon transducer of a single word *Mars*

```
$ fstcompile --isymbols=ascii.syms --osymbols=wotw.syms >Mars.fst <<EOF
0 1 M Mars
1 2 a <epsilon>
2 3 r <epsilon>
3 4 s <epsilon>
4
EOF
```

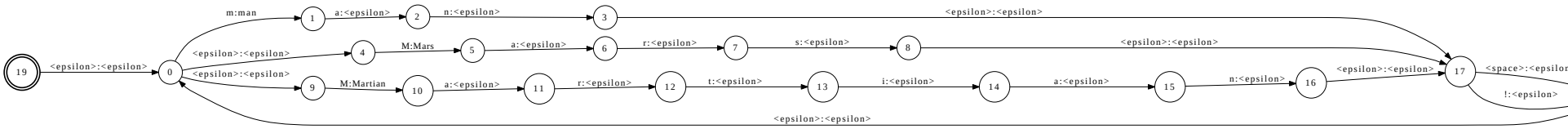


Tokenization - II

Suppose `Martian.fst` and `man.fst` are similarly created, then:

```
$ fstunion man.fst Mars.fst | fstunion - Martian.fst | fstclosure >lexicon.fst
```

produces a finite-state lexicon that transduces zero or more spelled-out word sequences into to their word tokens:

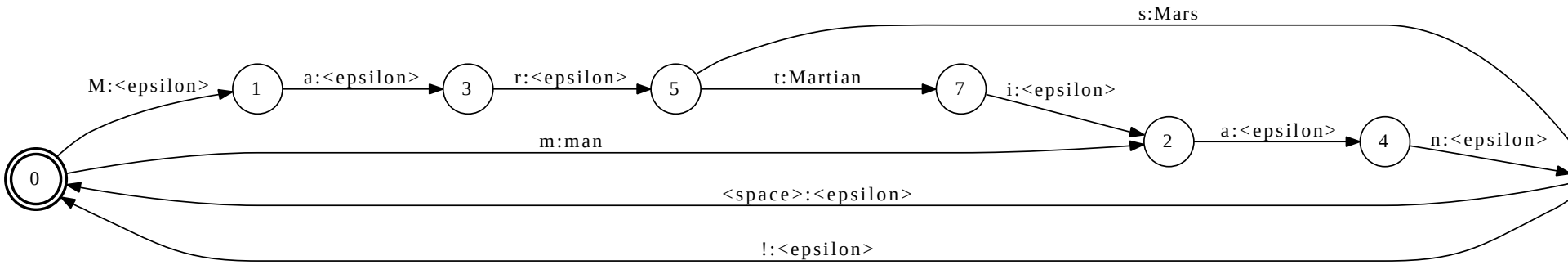


Tokenization - III

The non-determinism and non-minimality in the lexicon transducer can be removed with:

```
$ fstrmepsilon lexicon.fst | fstdeterminize | fstminimize >lexicon_opt.fst
```

resulting in the equvialent, deterministic and minimal:



Tokenization - IV

To handle punctuation symbols, we change the lexicon construction to:

```
$ fstunion man.fst Mars.fst | fstunion - Martian.fst |
fstconcat - punct.fst | fstclosure >lexicon.fst
```

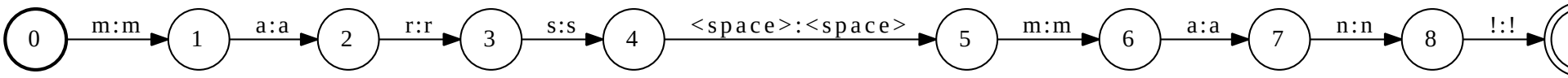
where:

```
$ fstcompile --isymbols=ascii.syms --osymbols=wotw.syms >punct.fst <<EOF
0 1 <space> <epsilon>
0 1 .  <epsilon>
0 1 , <epsilon>
0 1 ?  <epsilon>
0 1 !  <epsilon>
1
EOF
```

is a transducer that deletes common punctuation symbols.

Tokenization - V

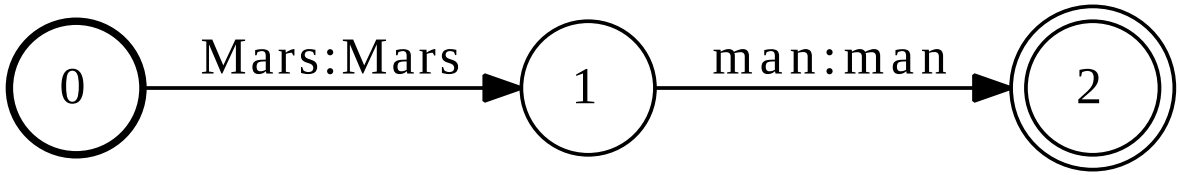
The tokenizaton of the example string Mars man encoded as an FST:



can be accomplished with:

```
$ fstcompose Marsman.fst lexicon_opt.fst | fstproject -- project_output |
fstrmepsilon >tokens.fst
```

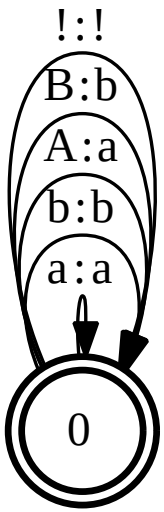
giving:



Downcasing Text - I

- Convert case-sensitive input to all lowercase output.
- Create a *flower* transducer of the form:

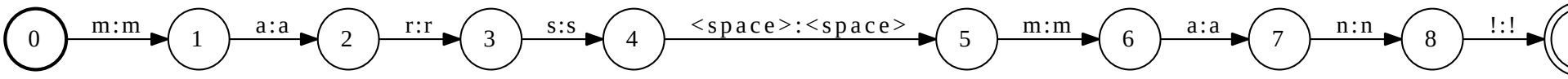
```
$ fstcompile --isymbols=ascii.syms --osymbols=ascii.syms >downcase.fst <<EOF
0 0 !  !
0 0 A a
0 0 B b
0 0 a a
0 0 b b
0
EOF
```



Downcasing Text - II

This transducer can be applied to the Mars men automaton from the previous example with:

```
$ fstproject Marsman.fst | fstcompose - downcase.fst |
fstproject --project_output >marsman.fst
```



Advantages:

- Downcases any automaton - e.g. the lexicon from the previous example:

  ```
  $ fstinvert lexicon_opt.fst | fstcompose - downcase.fst |
  fstinvert >lexicon_opt_downcase.fst
  ```

- Inverse FST can be used to restore case – next example.

Case Restoration - I

- Create a transducer that attempts to restore the case of downcased input.
- No error-free way to do this, in general.
- Use 5-gram LM of *The War of the Worlds* corpus in *OpenFst* format:

```
$ fstrandgen --select=log_prob wotw_lm.fst|
fstprint --isymbols=wotw.syms --osymbols=wotw.syms | cut -f3 | tr '\n' ' '
The desolating cry <epsilon> worked <epsilon> upon my mind once I \
<epsilon> <epsilon> slept <espilon> little
```

Epsilons are used to represent backoff transitions (as will be described in the ASR section).

Case Restoration - II

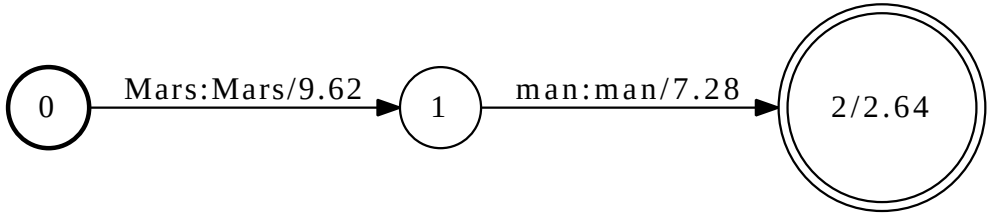
Given this language model and using the lexicon and downcasing transducers from the previous examples, a case restoration solution is:

```
$ fstcompose lexicon_opt.fst wotw_lm.fst| fstarcsort --sort_type=ilabel >wotw.fst
$ fstinvert downcase.fst | fstcompose - wotw.fst >case_restore.fst
```

- wotw.fst: maps from letters to tokens following the probability distribution of the language model.
- case_restore.fst: as above, but uses only downcased letters.

Case prediction can then be performed with:

```
$ fstcompose marsman.fst case_restore.fst | fstshortestpath |
fstproject --project_output | fstrmepsilon | fsttopsort >prediction.fst
```



Case Restoration - III

Unfortunately, the first composition above is extremely expensive since the output labels in the lexicon are pushed back in determinization, delaying matching.

One solution: use the input to restrict the composition chain:

```
$ fstcompose downcase.fst marsman.fst | fstinvert |
fstcompose - lexicon_opt.fst | fstcompose - wotw_lm.fst| fstshortestpath |
fstproject -project_output | fstrmepsilon | fsttopsort >prediction.fst
```

Works but has the disadvantages that it requires:

- a multiple transduction cascade for application
- inputs be strings or otherwise small

Case Restoration IV

A second solution:

- Replace *transducer* determinization and minimization of the lexicon with *automata* determinization and minimization
- Apply *transducer* determinization and minimization to the result of composition with the language model.

```
$ fstencode --encode_labels lexicon.fst enc.dat | fstdeterminize |
fstminimize | fstencode --decode - enc.dat >lexicon_compact.fst

$ fstcompose lexicon_compact.fst wotw_lm.fst| fstdeterminize | fstminimize >wotw.fs
$ fstinvert downcase.fst | fstcompose - wotw.fst >case_restore.fst
```

A natural solution but has the disadvantage that the (much larger) transducer determinization and minimization steps are expensive.

Case Restoration V

A third solution: use an FST representation that allows lookahead matching to avoid composition matching delays:

```
$ fstconvert --fst_type=olabel_lookahead --save_relabel_opairs=relabel.prs \
lexicon_opt.fst >lexicon_lookahead.fst
# Relabels the language model input (required by lookahead implementation)
$ fstrelabel --relabel_ipairs=relabel.prs wotw_lm.fst|
fstarcsort --sort_type=ilabel >wotw_relabel.lm
```

This can now be used efficiently with the original proposed solution:

```
$ fstcompose lexicon_lookahead.fst wotw_relabel.lm >wotw.fst
$ fstinvert downcase.fst | fstcompose - wotw.fst >case_restore.fst
```

Edit Distance - I

- Predictions (as in the previous example) may not always be correct
- Error can be measured when the reference answers available
- To measure the error, align the hypothesis with the reference and define:

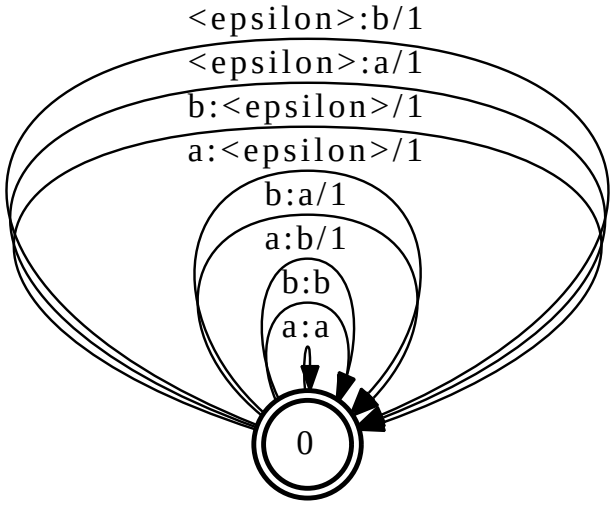
$$\begin{aligned} edit\_distance &= \#_{substitutions} + \#_{deletions} + \#_{insertions} \\ error\_rate &= \frac{edit\_distance}{\#_{reference\_symbols}} \end{aligned}$$

Given reference ref.fst, (unweighted) hypothesis, hyp.fst, and an edit transducer, edit.fst, the edit distance can be computed with:

```
$ fstcompose ref.fst edit.fst | fstcompose - hyp.fst |
# Returns shortest distance from final states to the initial (first) state
$ fstshortdistance --reverse | head -1
```

Edit Distance - II

The edit transducer, edit.fst, for two letters a and b, is the flower transducer:



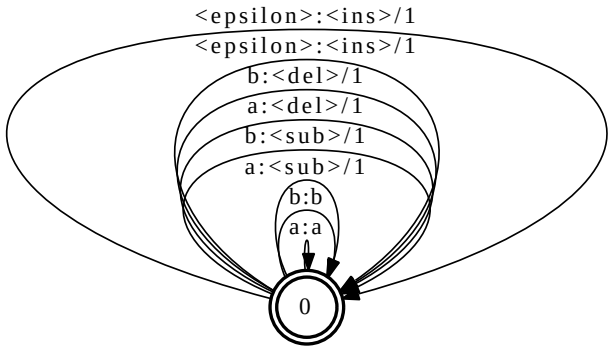
- Counts a substitution, insertion, or deletion as 1 edit
- Counts a match as zero edits.
- Generalizations of string-to-string Levenshtein distance:
  - Allow arbitrary weights per pairs.
  - Limit the number of contiguous insertions or deletions
  - Specify merge, split, or transposition weights
  - Use multiple hypothesis set (lattice) input: oracle edit distance

Edit Distance – III

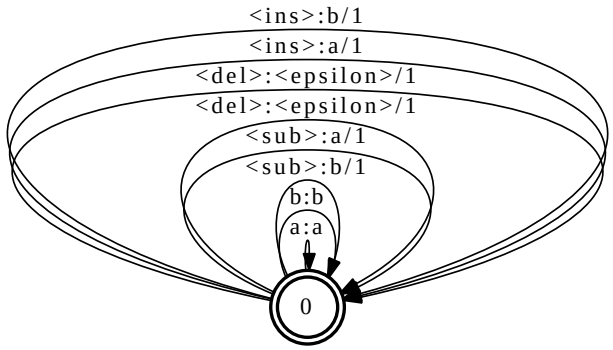
The edit transducer has:

- 9215 transitions for $|V| = 95$ letters (ascii.syms)
- 50,438,403 transitions for $|V| = 7101$ words (word.syms)

Levenshtein-case solution: factor the edit transducer as:



edit1.fst



edit2.fst

- New symbols <sub>, <del>, and <ins>
- Factors compose into the original edit transducer
- Original edit transducer: $(|V| + 1)^2 - 1$ transitions
- Factored transducers: $4|V|$ transitions

Edit Transducer - IV

Given these factors, compute:

```
$ fstcompose ref.fst edit1.fst >ref_edit.fst
$ fstcompose edit2.hyp hyp.fst >hyp_edit.fst
$ fstcompose ref_edit.fst hyp_edit.fst | fstshortdistance --reverse | head -1
```

Search space is quadratic in the input length, so if large:

- Use inadmissable pruning
- Limit the number of contiguous insertions or deletions

With more general, unfactorable, edit transducers, use:

- specialized edit transducer representation
- three-way composition
- specialized composition filter

ASR Applications

ASR Problem Definition

Given an utterance, find its most likely written transcription.

Fundamental ideas:

- Utterances are built from sequences of units
- Acoustic correlates of a unit are affected by surrounding units
- Units combine into higher level units — phones → syllables → words
- Relationships between levels can be modeled by weighted graphs
- Recognition: find the best path in a suitable product graph

Maximum-Likelihood Decoding

Overall analysis:

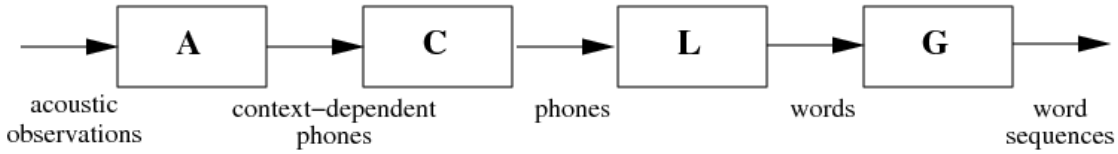
- Acoustic observations: parameter vectors derived by local spectral analysis of the speech waveform at regular (e.g. 10msec) intervals
- Observation sequence $\mathbf{o}$
- Transcriptions $\mathbf{w}$
- Probability $P(\mathbf{o}|\mathbf{w})$ of observing $\mathbf{o}$ when $\mathbf{w}$ is uttered
- Maximum-likelihood decoding:

$$\begin{aligned} \hat{\mathbf{w}} &= \operatorname*{argmax}_{\mathbf{w}} P(\mathbf{w}|\mathbf{o}) = \operatorname*{argmax}_{\mathbf{w}} \frac{P(\mathbf{o}|\mathbf{w})P(\mathbf{w})}{P(\mathbf{o})} \\ &= \operatorname*{argmax}_{\mathbf{w}} \underbrace{P(\mathbf{o}|\mathbf{w})}_{\substack{\text{channel} \\ \text{model}}} \underbrace{P(\mathbf{w})}_{\substack{\text{language} \\ \text{model}}} \end{aligned}$$

Generative Models of Speech

Typical decomposition of $P(\mathbf{o}|\mathbf{w})$ into conditionally-independent mappings between levels:

- Acoustic model $P(\mathbf{o}|\mathbf{p})$: phone sequences $\rightarrow$ observation sequences. Detailed model:
  - $P(o|d)$: distributions $\rightarrow$ observation vectors — *symbolic* $\rightarrow$ *quantitative*
  - $P(\mathbf{d}|m)$: context-dependent phone models $\rightarrow$ distribution sequences
  - $P(\mathbf{m}|\mathbf{p})$: phone sequences $\rightarrow$ model sequences
- Pronunciation model $P(\mathbf{p}|\mathbf{w})$: word sequences $\rightarrow$ phone sequences
- Language model $P(\mathbf{w})$: word sequences



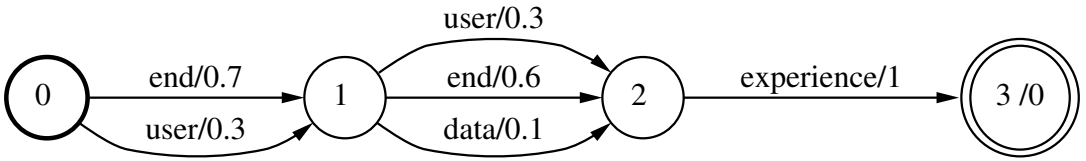
Speech Recognition Problems

- Modeling: how to describe accurately the relations between levels ⇒ *modeling errors*
- Search: how to find the best interpretation of the observations according to the given models ⇒ *search errors*

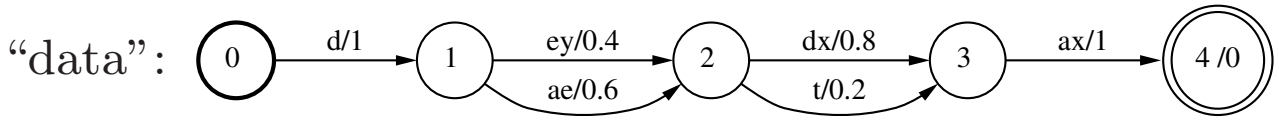
This talk will emphasize the latter topic.

Simple ASR Search I – Network Representation

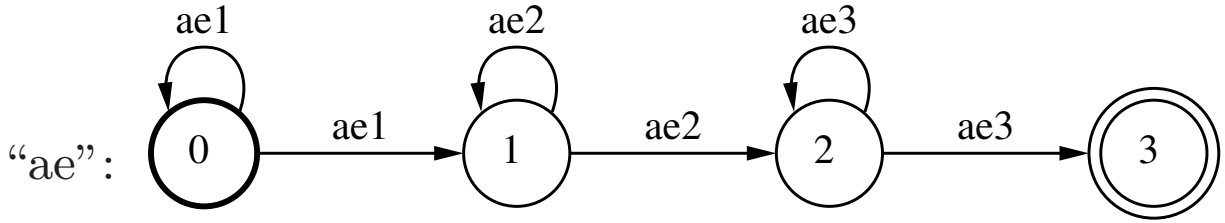
- Grammar: word network



- Lexicon: mapping from word label to phonetic network



- Phone model: mapping from phone label to HMM network



Simple ASR Search II – Network Substitution and Viterbi Search

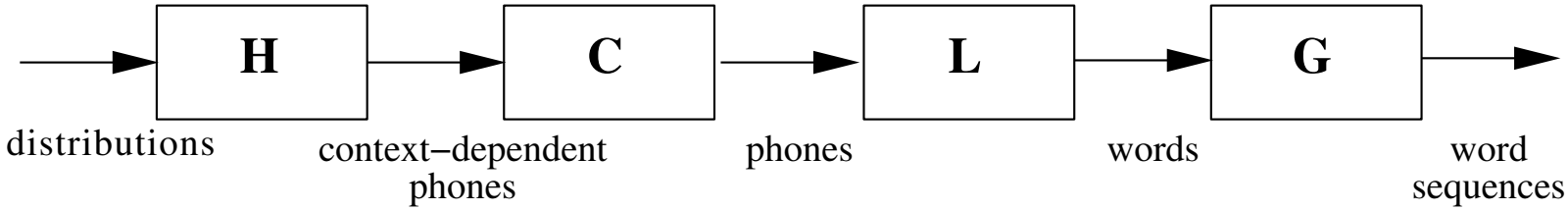
- Above networks are recursively substituted, either offline or dynamically during recognition, to form a single large network.
- The combined network is time-synchronously (Viterbi) matched to the incoming speech and searched for the best (lowest total cost) matching path which is returned as the hypothesized word string. For improved speed, partial paths that score less than the best path so far (outside the so-called beam) can be pruned, but this potentially creates search errors.

Problems with Simple ASR Search

The problems with this simple approach for large vocabulary speech recognition include:

1. Context-Dependent Modeling: Context-dependent models are awkward to represent by network substitution. (*Solution:* finite-state transducers and composition.)
2. Network Redundancy and Size: Networks can be highly redundant and very large. (*Solution:* finite-state optimizations – determinization and minimization.)
3. Network Weight Distribution: The distribution of the grammar and pronunciation weights strongly affect pruning efficiency. What is the optimal way to distribute them? (*Solution:* weight pushing)

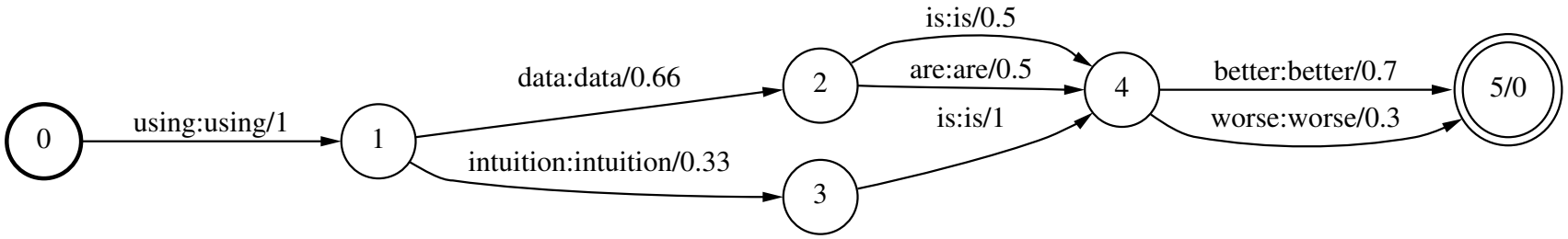
Context-dependent Recognition Transducer



- H: HMM transducer, closure of the union of all HMMs used in acoustic modeling,
- C: context-dependency transducer mapping context-dependent phones to phones,
- L: pronunciation dictionary transducer mapping phonemic transcriptions to word sequences,
- G: language model weighted automaton.

$H \circ C \circ L \circ G$: mapping from sequences of distribution names to word sequences.

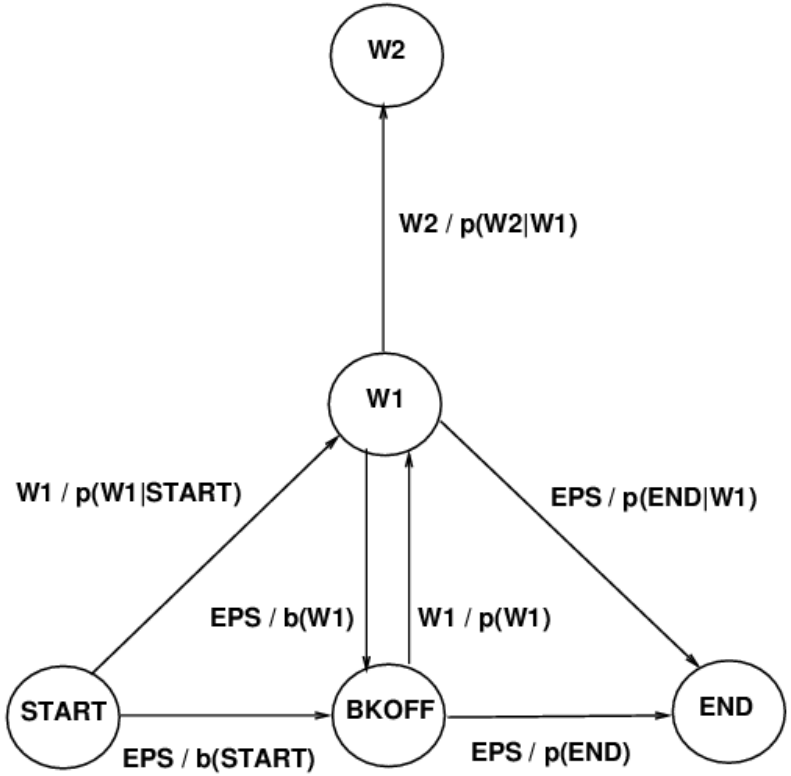
Grammar Acceptor



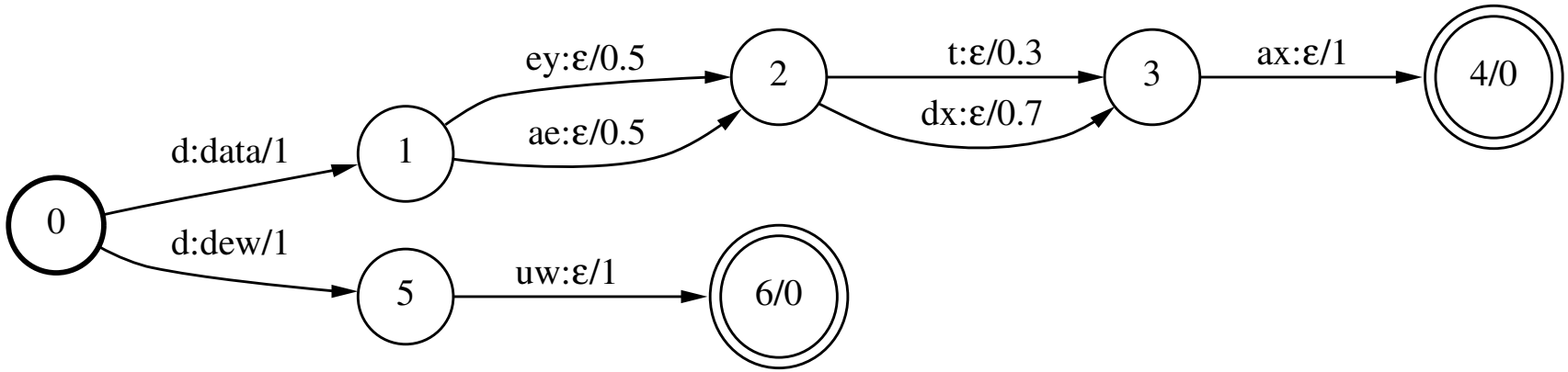
N-Gram Language Models

- Want to estimate the probability of a sentence $w_1, w_2, w_3, w_4, \ldots$
- By the chain rule of probability,
  $\Pr(w_1, w_2, \ldots) = \Pr(w_1) \times \Pr(w_2|w_1) \times \Pr(w_3|w_1, w_2) \times \Pr(w_4|w_1, w_2, w_3) \times \ldots$
- Bigram approximation, plus conventional handling of first word:
  $\Pr(w_1, w_2, \ldots) \approx \Pr(w_1 \mid \$) \times \Pr(w_2 \mid w_1) \times \Pr(w_3 \mid w_2) \times \Pr(w_4 \mid w_3) \times \cdots$
- Has a straightforward representation as an FST, where states encode conditioning histories.
- Backoff models can be represented inexactly with epsilon transitions or exactly with failure transitions.

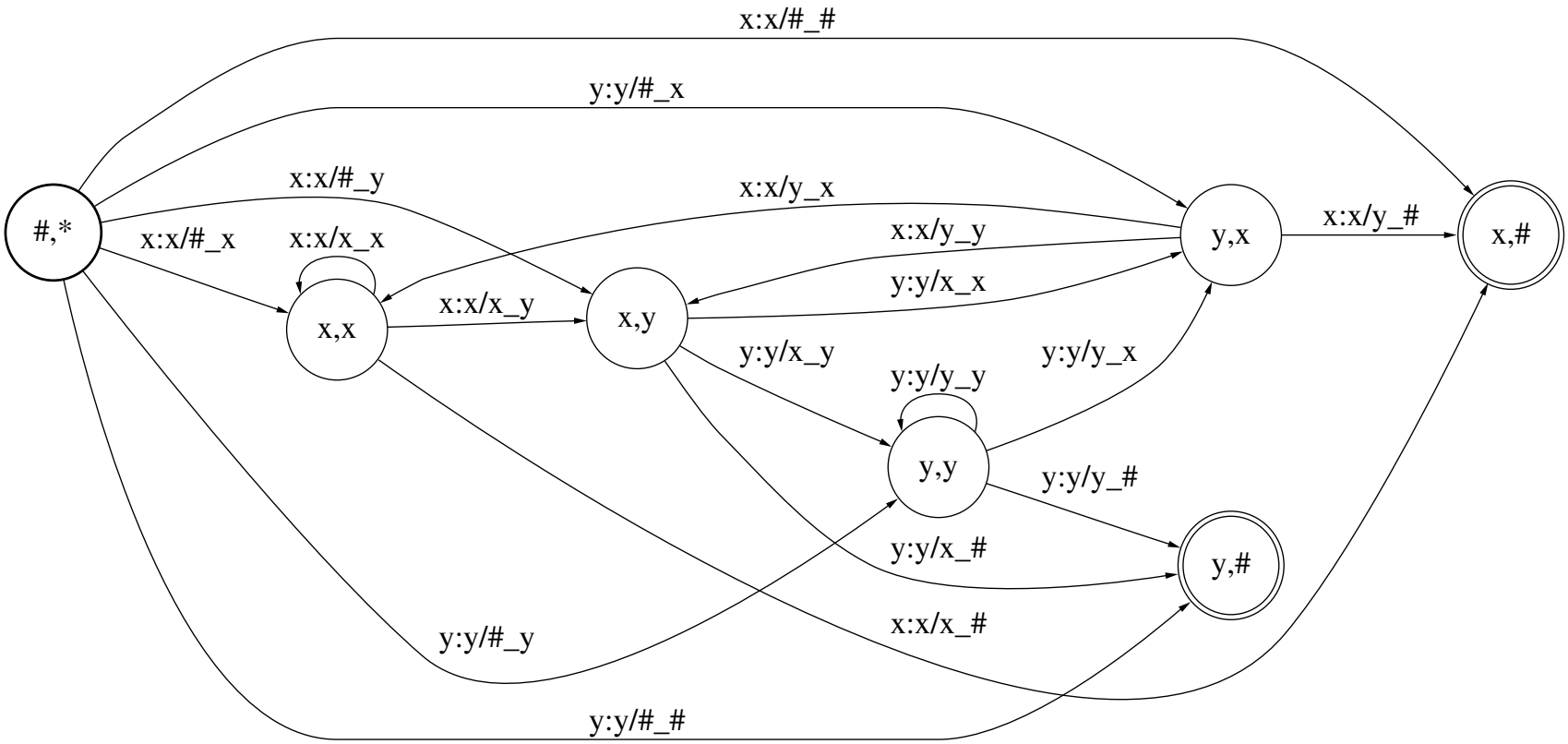
Bigram Grammar



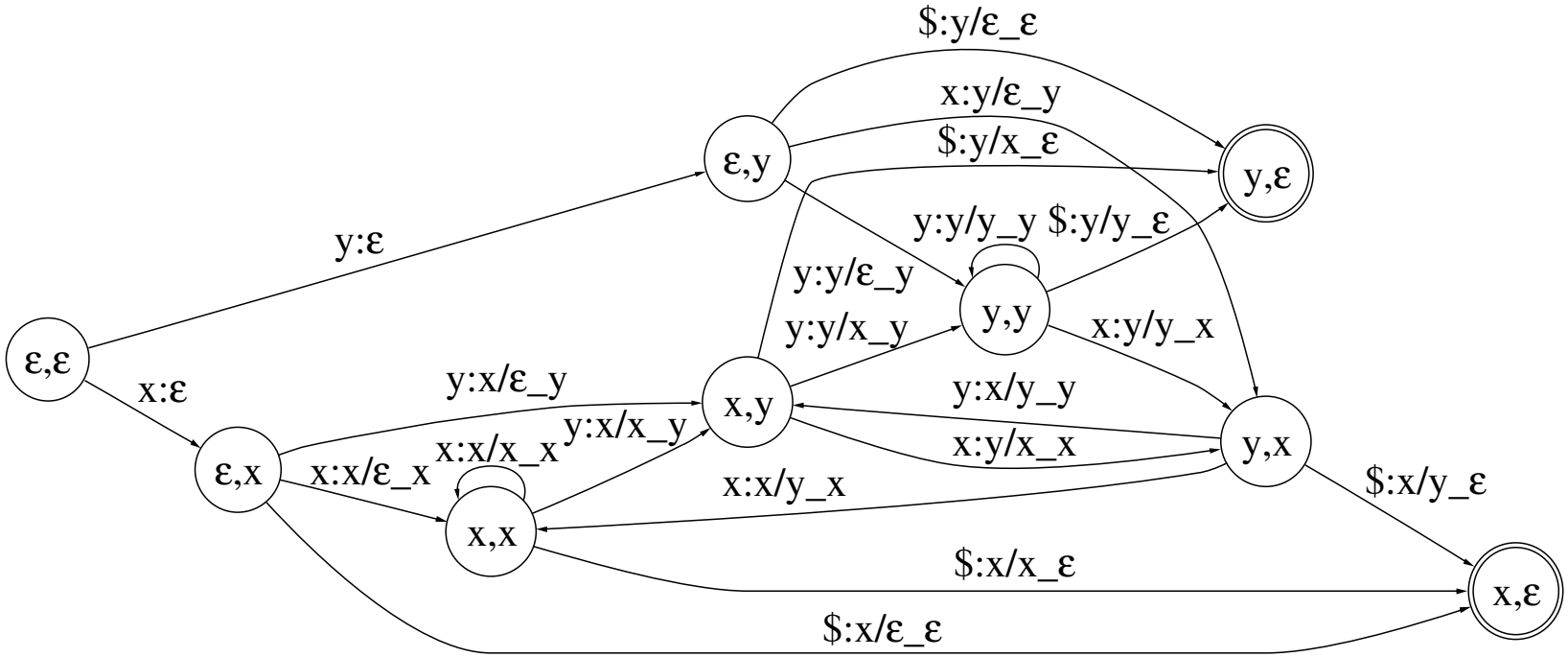
Pronunciation Lexicon Transducer



Context-Dependent Triphone Transducer



Deterministic Context-Dependent Triphone Transducer



Recognition Transducer Construction I: Disambiguation

1. $L \rightarrow \tilde{L}$, auxiliary symbols used to make $L \circ G$ determinizable (homophones, transduction's unbounded delay):

   $$\texttt{r eh d } \#_0 \quad \textit{read}$$
   $$\texttt{r eh d } \#_1 \quad \textit{red}$$

2. $C \rightarrow \tilde{C}$, self-loops used for further determinizations at the context-dependent level,

3. $H \rightarrow \tilde{H}$, self-loops at initial state, auxiliary context-dependent symbols mapped to new distinct distribution names.

Recognition Transducer Construction II: Combination/Optimization

1. Composition:

$$N = \pi_\epsilon(\tilde{H} \circ \tilde{C} \circ \tilde{L} \circ G)$$

2. Determinization:

$$N = \pi_\epsilon(det(\tilde{H} \circ det(\tilde{C} \circ det(\tilde{L} \circ G))))$$

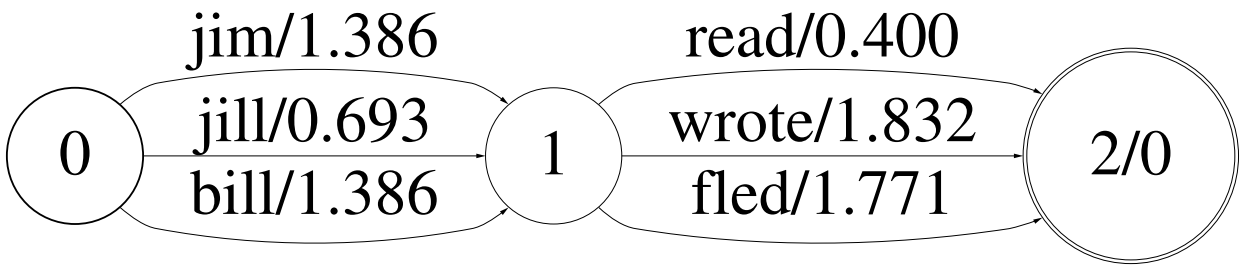
3. Minimization:

$$N = \pi_\epsilon(min(det(\tilde{H} \circ det(\tilde{C} \circ det(\tilde{L} \circ G)))))$$

4. Weight Pushing:

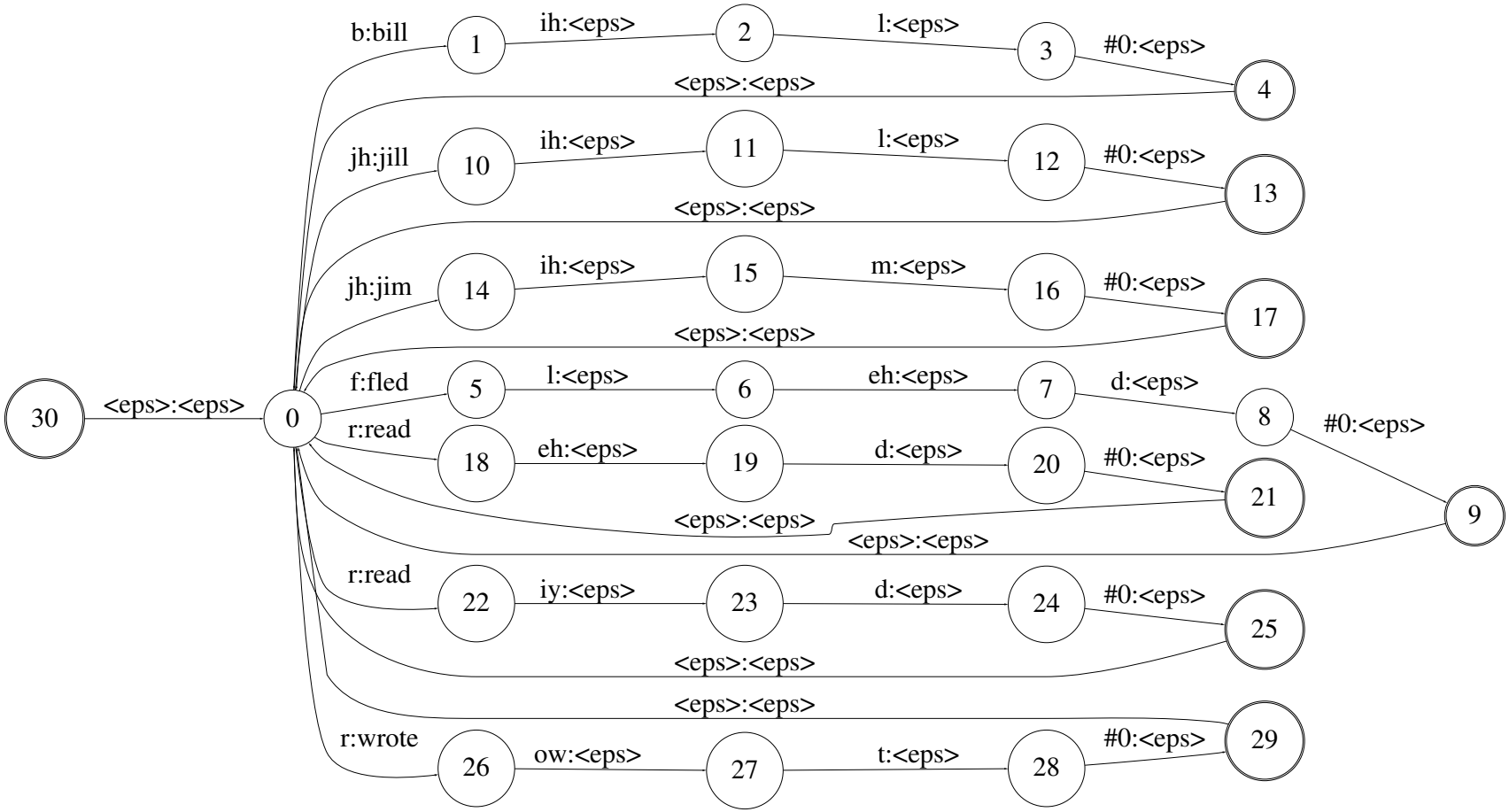
$$N = push(\pi_\epsilon(min(det(\tilde{H} \circ det(\tilde{C} \circ det(\tilde{L} \circ G))))))$$

Recognition Transducer Construction Example I



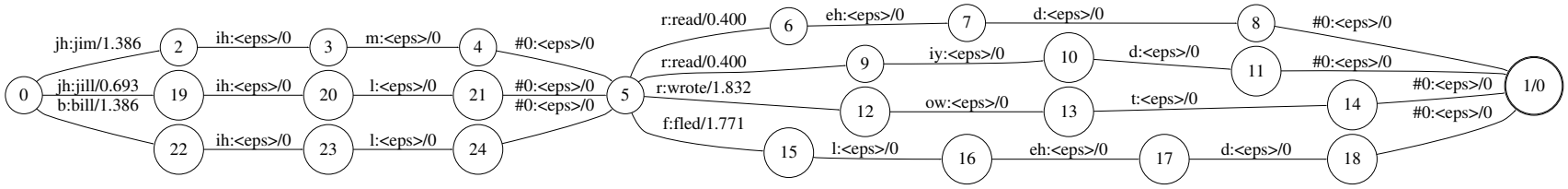
G

Recognition Transducer Construction Example II



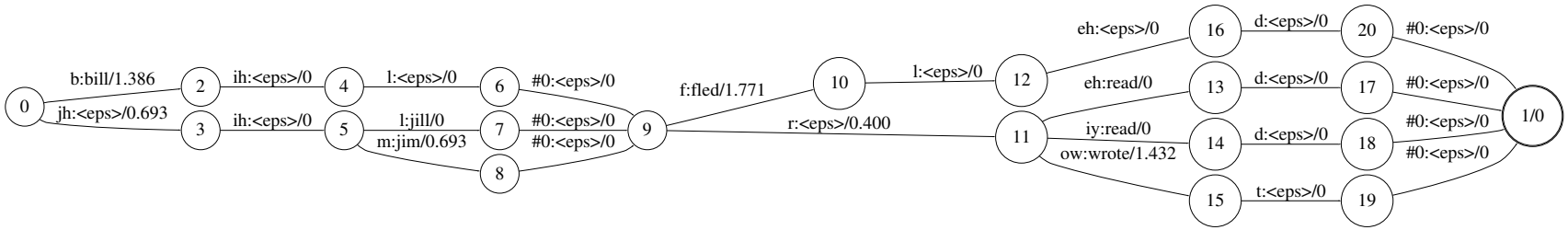
$\tilde{L}$

Recognition Transducer Construction Example III



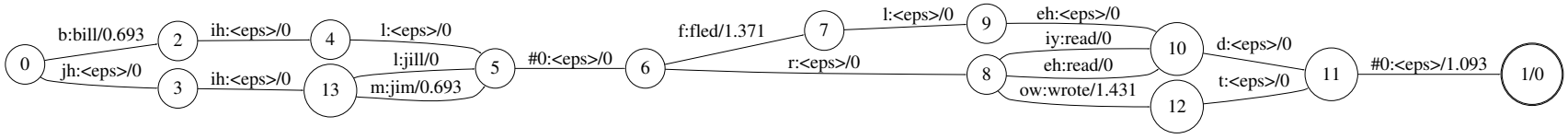
$$\tilde{L} \circ G$$

Recognition Transducer Construction Example IV



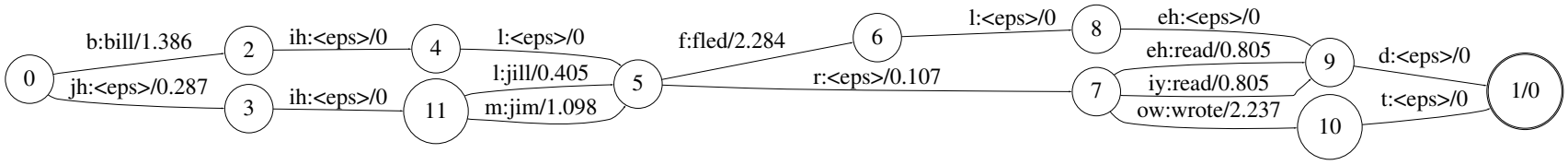
$$det(\tilde{L} \circ G)$$

Recognition Transducer Construction Example V



$$min(det(\tilde{L} \circ G))$$

Recognition Transducer Construction Example VI

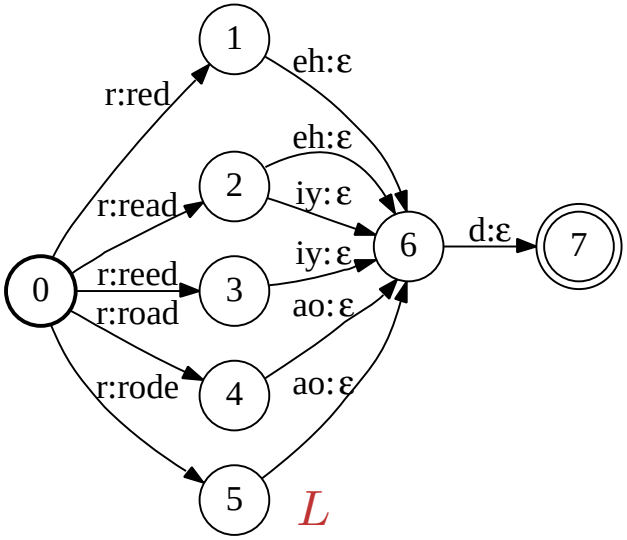


$$push(\pi_\epsilon(min(det(\tilde{L} \circ G))))$$

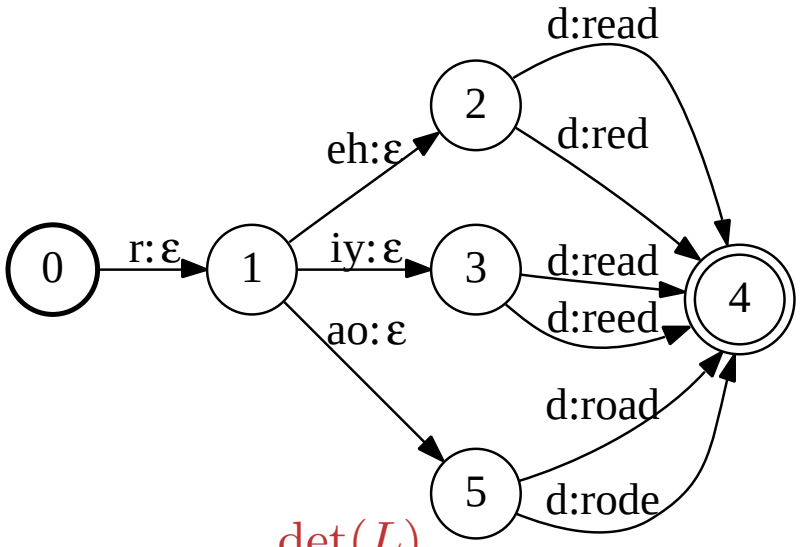
Recognition Transducer Construction – Alternatives

- $C \circ det(L \circ G)$
  Grammar G compiled into optimized transducer offline, cannot exchange it at runtime.
- $det(C \circ L) \circ G$
  Outermost composition with algorithm of Part 1 badly behaved - many no-coaccessible paths created.
  ▷ Solution: generalized composition (with lookahead).

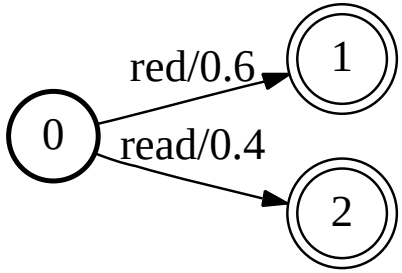
Alternative constructions – Illustration



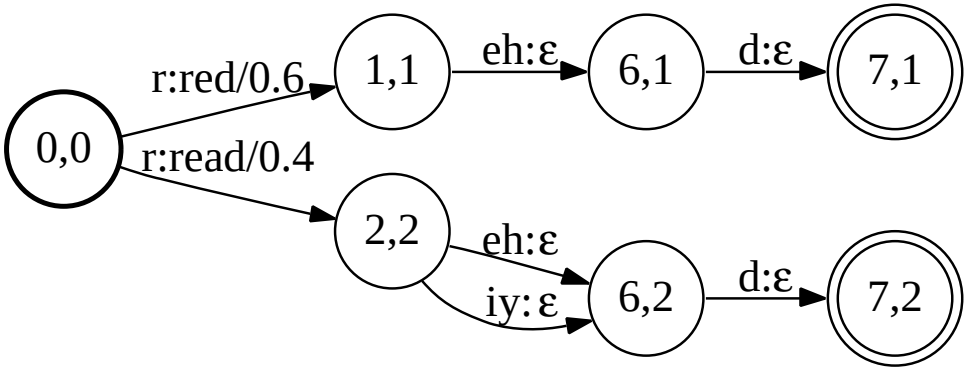
L



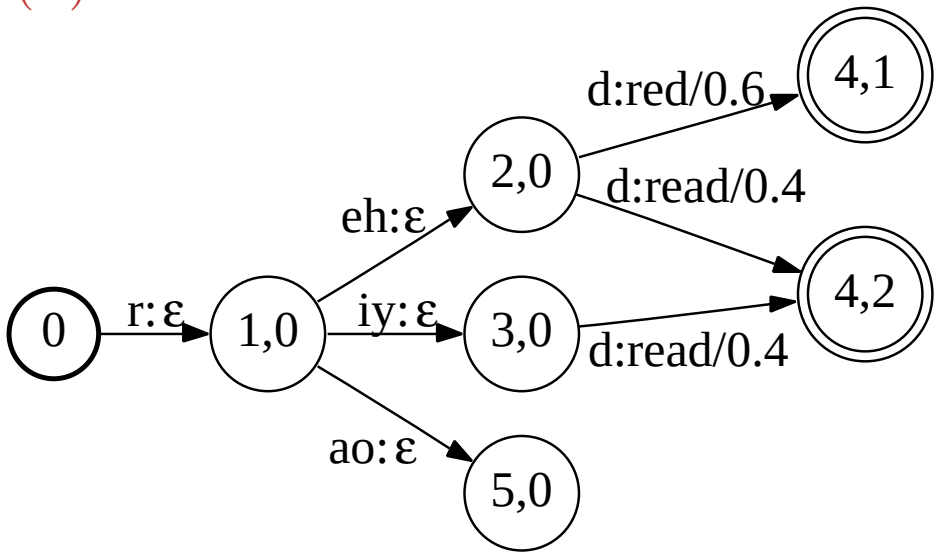
$\det(L)$



G



$L \circ G$



$\det(L) \circ G$

1st-Pass Recognition Transducers – 40K NAB Task

transducer	states	transitions
G	1,339,664	3,926,010
$L \circ G$	8,606,729	11,406,721
$det(L \circ G)$	7,082,404	9,836,629
$C \circ det(L \circ G))$	7,273,035	10,201,269
$det(H \circ C \circ L \circ G)$	18,317,359	21,237,992

1st-Pass Recognition Speed - 40K NAB Eval '95

transducer	x real-time
$C \circ L \circ G$	12.5
$C \circ det(L \circ G)$	1.2
$det(H \circ C \circ L \circ G)$	1.0

Recognition speed of the first-pass transducers in the NAB $40,000$-word vocabulary task at 83% word accuracy.

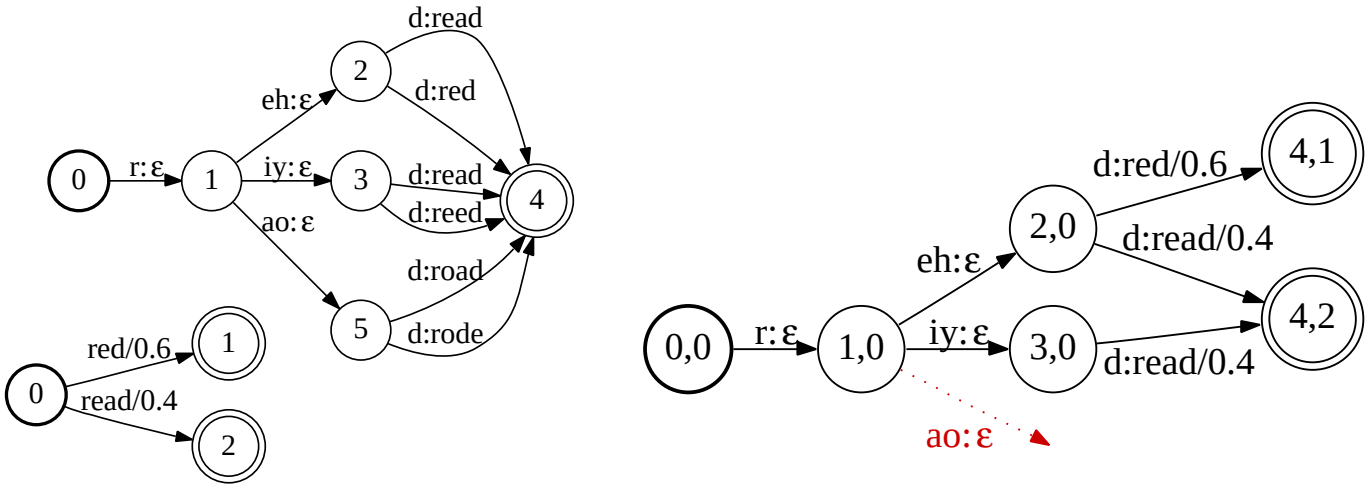
2nd-Pass Recognition Speed - 160K NAB Eval '95

transducer	x real-time
$C \circ L \circ G$	.18
$C \circ det(L \circ G)$	.13
$C \circ push(min(det(L \circ G)))$	.02

Recognition speed of the second-pass transducers in the NAB $160,000$-word vocabulary task at 88%.

Generalized Composition with Lookahead

- Goal: Efficient (static or dynamic) computation of $det(C \circ L) \circ G$
- Algorithm: Generalizes composition by generalizing the notion of composition filter. More than just epsilon filtering.
- Lookahead filters: Use some information about the input transducers to prevent the creation of non-coaccessible states in composition
- Label-reachability filter:



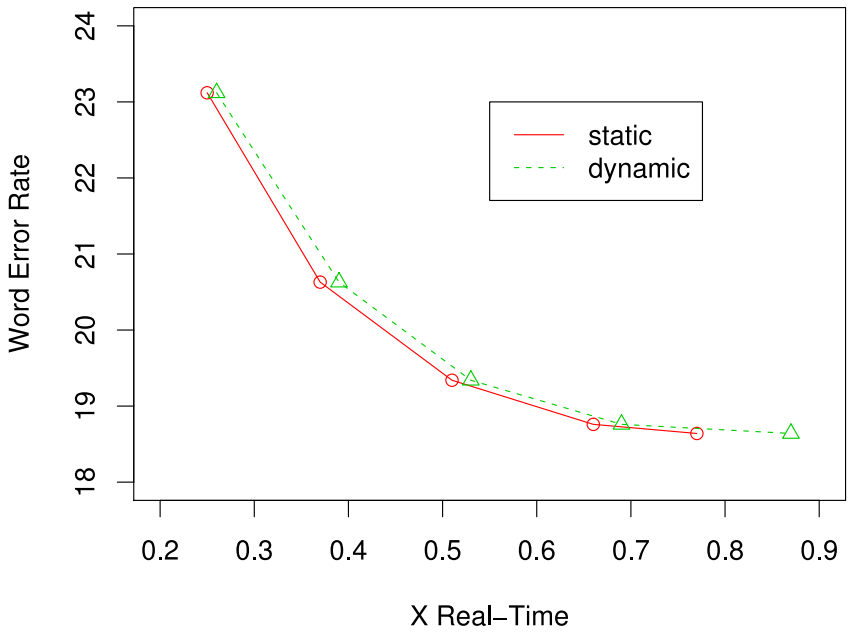
Recognition Experiments

Broadcast News	Spoken Query Task
Acoustic Model	
• Trained on 96 and 97 DARPA Hub4 AM training sets. • PLP cepstra, LDA analysis, STC • Triphonic, 8k tied states, 16 components per state • Speaker adapted (both VTLN + CMLLR)	• Trained on $>$ 1000hrs of voice search queries • PLP cepstra, LDA analysis, STC • Triphonic, 4k tied states, 4 - 128 components per state • Speaker independent
Language Model	
• 1996 Hub4 CSR LM training sets • 4-gram language model pruned to 8M n-grams	• Trained on $>$ 1B words of google.com and voice search queries • 1 million word vocabulary • Katz back-off model, pruned to various sizes

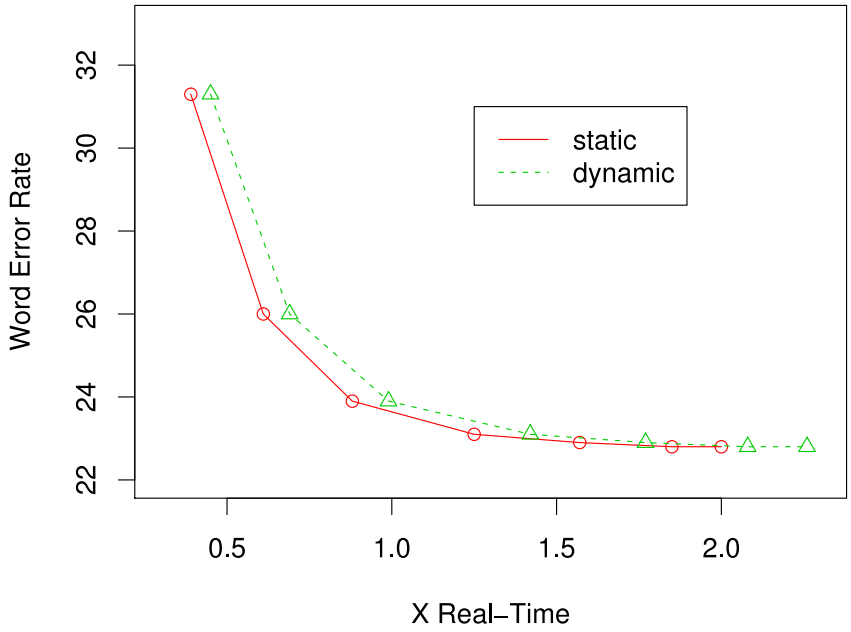
Recognition Experiments

Precomputation before recognition	Broadcast News			Spoken Query Task		
Construction method	Time	RAM	Result	Time	RAM	Result
Static						
(1) with standard composition	7 min	5.3G	0.5G	10.5 min	11.2G	1.4G
(2) with generalized composition	2.5 min	2.9G	0.5G	4 min	5.3G	1.4G
Dynamic						
(2) with generalized composition	none	none	0.2G	none	none	0.5G

Broadcast News



Spoken Query Task

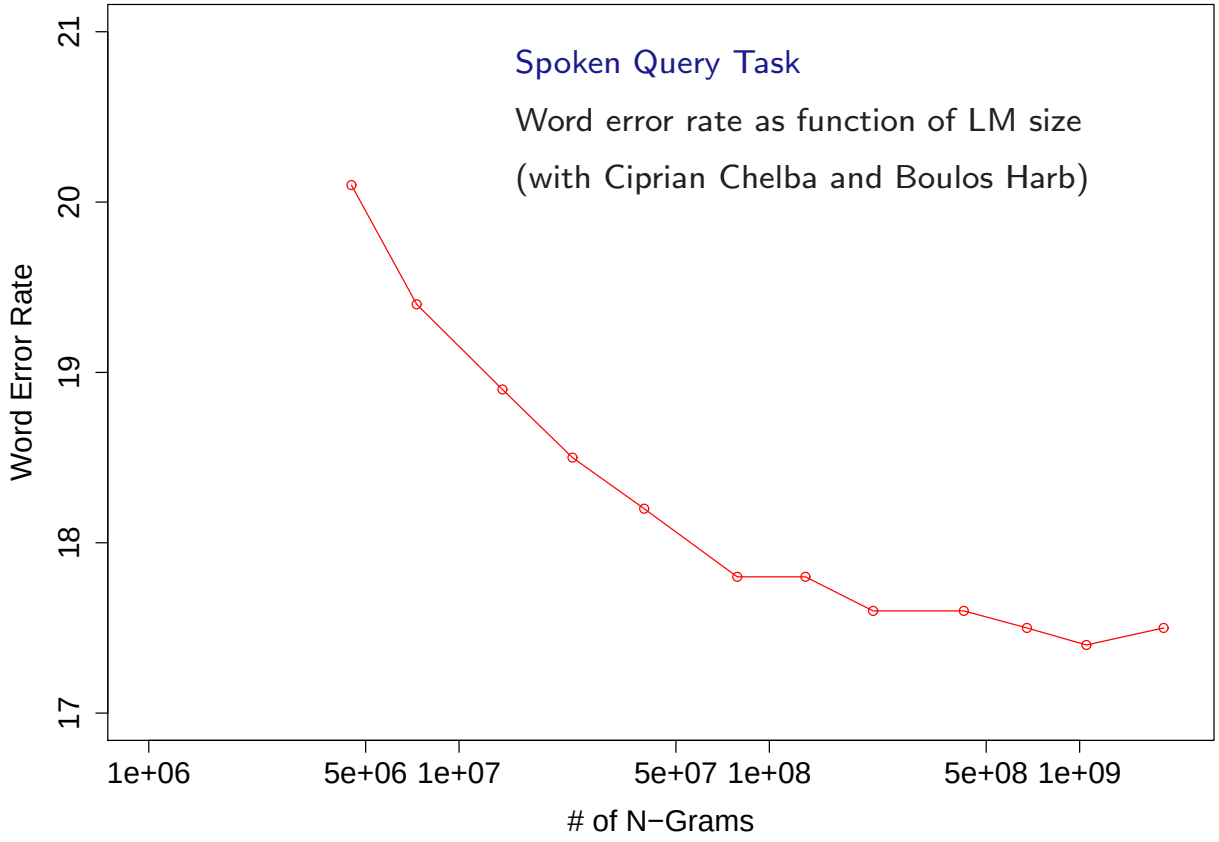


Recognition Experiments

- A small part of the recognition transducer is visited during recognition:

Spoken Query Task	Static	Number of states in recognition transducer	25.4M
	Dynamic	Number of states visited per second	8K

- Very large language models can be used in first-pass:



Conclusion

- More information available at `http://www.openfst.org`:
  - full documentation,
  - slides from this presentation and previous tutorials, and
  - links to relevant papers and books.
- If you have questions later,
  - post your questions in our forum at `http://forum.openfst.org`, or
  - contact us at {`allauzen,riley`}`@google.com`