IDIAP COMMUNICATION

# A Weighted Finite State Transducer tutorial

Philip N. Garner [a]

IDIAP–Com 08-01

December 2007

[a] IDIAP Research Institute

**IDIAP Research Institute** **www.idiap.ch**
Av. des Prés–Beudin 20 P.O. Box 592 1920 Martigny – Switzerland
Tel: +41 27 721 77 11 Fax: +41 27 721 77 12 Email: info@idiap.ch

# Contents

# 1 Background

## 1.1 ASR

The object of Automatic Speech Recognition (ASR) is normally to recognise words given some audio signal. This is typically represented probabilistically as a maximisation problem: choose the word sequence, $\hat{\boldsymbol{g}}$, with the highest likelihood given an acoustic sequence $\boldsymbol{a}$. The symbol $\boldsymbol{g}$ is used to suggest more general *grammar* entities, and to fit with notation in the literature.

As the recognition model is based on the acoustics having been generated dependent upon the word sequence (a generative model), the direction of inference is reversed to reflect this using Bayes's theorem thus:

$$\hat{\boldsymbol{g}} = \underset{\boldsymbol{g}\in\mathcal{G}}{\operatorname{argmax}}\, f\left(\boldsymbol{g} \mid \boldsymbol{a}\right) = \underset{\boldsymbol{g}\in\mathcal{G}}{\operatorname{argmax}}\, f\left(\boldsymbol{a} \mid \boldsymbol{g}\right) f\left(\boldsymbol{g}\right) \tag{1}$$

where $\mathcal{G}$ is the set of all possible word sequences, and the denominator of Bayes's theorem is ignored as it just affects the numerical value, not the optimal word sequence. Note that parameters or training data are normally required, but omitted for clarity.

In order to relate the audio signal to the words, the words are broken down into component parts, typically phones, context dependent phones, and ultimately states of an HMM. Noting first that the word sequence can be described in terms of one or more phone sequences, $\boldsymbol{l}$, (or *lexical* sequences),

$$f\left(\boldsymbol{a} \mid \boldsymbol{g}\right) = \sum_{\boldsymbol{l}\in\mathcal{L}} f\left(\boldsymbol{a} \mid \boldsymbol{l}, \boldsymbol{g}\right) f\left(\boldsymbol{l} \mid \boldsymbol{g}\right),$$

where $\mathcal{L}$ is the set of all possible lexical (phone) sequences. As $\boldsymbol{a}$ is conditionally independent of $\boldsymbol{g}$ given $\boldsymbol{l}$, the first term in the summation simplifies to $f\left(\boldsymbol{a} \mid \boldsymbol{l}\right)$.

Next note that phone sequence can be broken down into a sequence $\boldsymbol{c}$ of context dependent phones

$$f\left(\boldsymbol{a} \mid \boldsymbol{l}\right) = \sum_{\boldsymbol{c}\in\mathcal{C}} f\left(\boldsymbol{a} \mid \boldsymbol{c}, \boldsymbol{l}\right) f\left(\boldsymbol{c} \mid \boldsymbol{l}\right),$$

where $\mathcal{C}$ is the set of context dependent phone sequences and the same simplification can be applied by comparison with the previous equation.

Finally, note that the context dependent phone sequence can be represented by a sequence, $\boldsymbol{h}$, of PDFs arranged in a set of HMMs.

$$f\left(\boldsymbol{a} \mid \boldsymbol{c}\right) = \sum_{\boldsymbol{h}\in\mathcal{H}} f\left(\boldsymbol{a} \mid \boldsymbol{h}, \boldsymbol{c}\right) f\left(\boldsymbol{h} \mid \boldsymbol{c}\right),$$

where, as above, $\mathcal{H}$ is the set of all possible PDF sequences, and the conditional independence simplification applies.

Substituting these components into the original expression yields

$$\hat{\boldsymbol{g}} = \underset{\boldsymbol{g}\in\mathcal{G}}{\operatorname{argmax}} \sum_{\boldsymbol{l}\in\mathcal{L}} \sum_{\boldsymbol{c}\in\mathcal{C}} \sum_{\boldsymbol{h}\in\mathcal{H}} f\left(\boldsymbol{a} \mid \boldsymbol{h}\right) f\left(\boldsymbol{h} \mid \boldsymbol{c}\right) f\left(\boldsymbol{c} \mid \boldsymbol{l}\right) f\left(\boldsymbol{l} \mid \boldsymbol{g}\right) f\left(\boldsymbol{g}\right). \tag{2}$$

ASR is not normally expressed in the above terms because the summations either correspond to only a single possible sequence (in the case of context dependency), or are approximated by a maximisation via the Viterbi algorithm. However, the formulation will be useful in the following discussion. In summary, note that each of the sets $\mathcal{G}$, $\mathcal{L}$, $\mathcal{C}$ and $\mathcal{H}$ are capable of representing two distinct things:

1. A 'structural' or 'grammatical' restriction concerning which entities can follow which other entities.

2. A probabilistic restriction that favours some grammatical sequences over others.

## 1.2 Finite State Transducers

It has long been common to represent the allowable word sequences, $\mathcal{G}$, using a grammar. In general, the grammar can be represented by a directed graph with probabilities on the arcs, and words on either the arcs or the vertices. Even N-gram language models, which are typically not thought of as graphs, can be represented in this way [1]. In the probabilistic interpretation, the graph is thought of as a generative model, and is hence a Finite State Machine (FSM), also known as a Finite State Automaton (FSA). In FSM terminology, the arcs are transitions and the vertices are states. There are two types of FSM:

1. In a Moore[1] machine, an output is produced for each state.

2. In a Mealy[2] machine, an output is produced for each transition.

It is always possible to write a Moore machine as a Mealy machine by moving the output on each state either forward to each outgoing transition or backward to each incoming transition. Writing a Mealy machine as a Moore machine is also trivial, but can require the introduction of extra states.

Now consider what happens if we take a FSM, but add an input associated with each output. The resulting automaton can still generate output, but can be constrained by an external input sequence. Such a FSM is known as a Finite State Transducer (FST).

For the purposes of ASR, we define a FST to be a Mealy machine where each transition has an input and an output. Further, each input or output can be labelled with the null label $\epsilon$ (epsilon). Epsilon input labels consume no input, and epsilon output labels produce no output. Epsilon transitions enable FSTs where the input sequence is not the same length as the output sequence.

This type of FST was developed for ASR purposes at AT&T by Mehryar Mohri and colleagues and is documented extensively in a series of papers, a representative summary being Mohri *et al.* [2].

In FST theory, a FSM is known also as a Finite State Acceptor.

## 1.3 FSTs and ASR

Consider an ASR situation where a grammar is used to restrict the possible word sequences to be recognised. If the grammar can be represented as a FSM, and since a FST is a superset of a FSM, we can use a FST, $G$, to represent this grammar. Such a FST is shown in figure 1. The grammar has

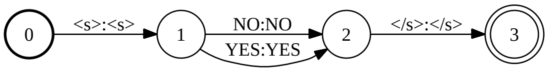


Figure 1: Simple yes no grammar.

start and end symbols, which can be mapped to silence later, and simply allows one or other of the words ‘YES’ or ‘NO’. The grammar has ‘matched’ input and output symbols, the reason for which will become clear shortly.

Now consider the lexicon, $L$. A lexicon is more often known as a dictionary, and not generally represented as a FST. However, figure 2 illustrates how it can be represented as such. Notice that the symbol `<eps>` is used to represent the null label, $\epsilon$. The transition from state 4 back to state 0 is known as a ‘closure’, and allows the lexicon to represent a *sequence* of transductions.

The reason for the representation as a FST becomes clear in the context of FST composition. Composition takes two transducers and finds all possible paths though each transducer such that the output labels of the first and the input labels of the second produce the same sequence. The output is another transducer representing those matching paths, with the input labels of the first transducer

[1]Mnemonic: mOOre, On circles (states)
[2]Mnemonic: mEAly, Every Arc

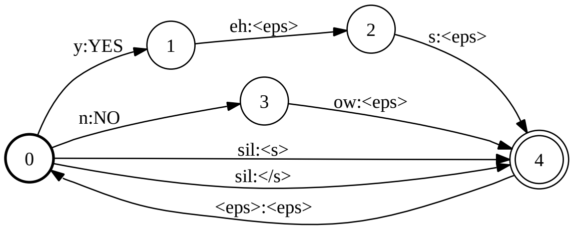


Figure 2: Simple lexicon transducer.

and the outputs of the second. The matching labels are annihilated. Mathematically, the composition of lexicon $L$ and grammar $G$ is written

$$L \circ G. \tag{3}$$

Practically, it results in the transducer shown in figure 3.

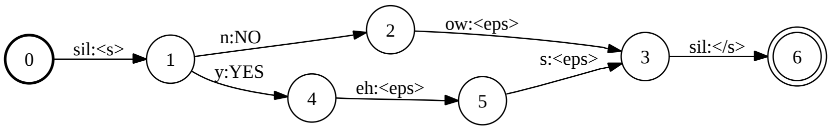


Figure 3: Transducer resulting from composition of grammar and lexicon.

Notice that this is simply a rather obvious substitution of lexical elements for words. This sort of thing can be done in an ASR decoder without great strain on computing resources. The real power of FSTs, however, is illustrated as follows. Imagine that it is desirable to represent an abbreviation of the word 'YES' without the final 's' sound. This might be pronounced 'YEAH', and it might be either a distinct word, or an alternative pronunciation. The extra word and pronunciation can be trivially added to $G$ and $L$ as shown in figure 4, but the composition results in the transducer in

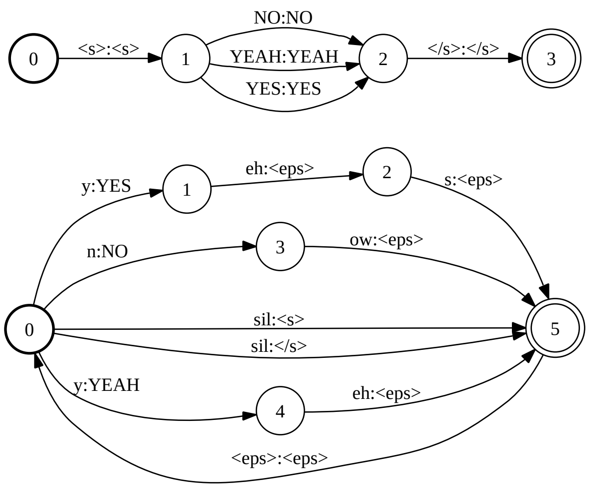


Figure 4: Grammar and lexicon transducers for extra pronunciation.

figure 5. Notice that the result is not just a simple substitution; the composition algorithm has also

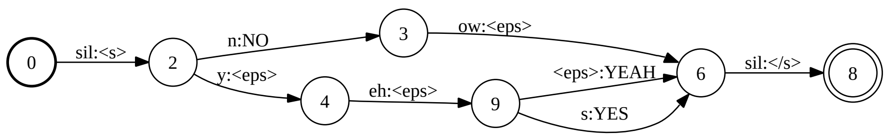


Figure 5: Transducer representing words 'YES', 'NO' and 'YEAH'.

taken advantage of the shared pronunciation of two words by 'pushing' the labels 'YES' and 'YEAH' towards the end of the transducer. This results in compression of the final transducer and (crucially) a reduction of the decoder search space. It should be clear by inspection that if the two pronunciations had been introduced as pronunciations of the same word, then that word could have been left at the beginning of the transducer.

This effect, known as prefixing, is directly analogous to the tree structured lexicon that is typically used in ASR decoders (see for instance Haeb-Umbach and Ney [3], who attribute it to Bahl *et al.*). In addition, the composition generally performs suffixing. Both effects can be enhanced using minimisation algorithms. This type of optimisation has been shown to significantly improve ASR decoder performance (in terms of speed) for marginal increases in memory [4, 5].

Many of the main benefits of the use of FSTs come from the lexicon compression. However, another benefit is the unified approach to representation. This is evident in the fact that both the context dependency and the HMMs themselves can also be represented as transducers. The general ASR grammar composition can then be written

$$H \circ C \circ L \circ G. \tag{4}$$

A discussion of the context dependency transducer, $C$, only makes sense after some discussion of structural optimisation.

# 2 Structural optimisation

## 2.1 Introduction

In this section, the structural aspects of WFSTs are discussed, particularly determinisation.

## 2.2 Determinisation

Composition works by finding common paths through the output labels of one FST and the input labels of another. It follows that reducing the number of such paths will reduce the final size of the composed FST. One way to reduce the number of common paths is known as determinisation. Determinisation is discussed at length in the literature from AT&T in the context of general FST theory [6], and an algorithm is provided to determinise a non-deterministic FST.

A deterministic FSM is one where any given sequence can correspond to only one path through the FSM. One way to achieve this is to make sure that each transition out of any given state has a different label. When FSTs are used instead of FSMs, it is possible to consider three different types of determinism:

1. w.r.t. the input labels.

2. w.r.t. the output labels.

3. w.r.t. both the input and output labels combined.

By convention, determinism in FSTs refers to the first case, i.e., the input labels only. The second case is achieved by defining the inverse of FST $X$ to be $X^{-1}$, where the input and output labels are swapped. A FST can then be said to have a deterministic inverse. The third case is achieved in software by re-coding the FST with output labels merged into the input labels, i.e., encoding as an acceptor.

With the above in mind, it now possible to make a key observation:

> The composition $X \circ Y$ will proceed more easily and to a smaller result if one or both of $X^{-1}$ and $Y$ are deterministic.

Another key point about determinisation is that the composition of two deterministic transducers is determinisable. Hence, in a transducer hierarchy where the composition

$$X \circ (Y \circ Z) \tag{5}$$

is required, if both $Y$ and $Z$ are both deterministic, then $Y \circ Z$ can be determinised and the final transducer is then determinisable.

### 2.3 Auxiliary labels

If a FST is not deterministic, it can be worth (in terms of FST size and decoding speed) going to some trouble to force it to be deterministic. One trick to aid this process is the use of auxiliary labels. These labels are added during the generation of the initial FST, and can either be removed at some stage in the composition process, or left in place to be used by the decoder.

There are two exemplary situations where auxiliary labels are of use:

1. In the back-off transitions of FSTs representing n-gram language models.

2. In the pronunciations of homophones in the lexicon FST.

In each case, the auxiliary labels render a non-deterministic FST either deterministic or determinisable, or ensure that the result of a subsequent composition is determinisable. Their use is illustrated in the composition walk-through below.

## 3 Context dependency

Context dependency is not normally represented as a FST, but it can be done, albeit not trivially. It is discussed by Riley *et al.* [7]. Note that the transducer in figure 4 of that paper is synchronous, i.e., the input phone corresponds to the centre phone of the output triphone. It also requires determinisation.

Typically, a FST is required that is the inverse of the one in [7], with input context dependent phones and output phones. Further, an algorithm exists that produces a FST that does not require determinisation. The output of such an algorithm is illustrated in figure 6. The central state labeled ‘sil’ (silence phone) is a ‘context independent’ state; all transitions leaving this state output the context independent label ‘sil’ and lead to states on a first concentric circle. All transitions leaving the first concentric circle either ‘back-off’ to the context independent state, or lead to context dependent states on another (outer) concentric circle. In general, sequences of context dependent phones are matched by moving around the outer concentric circle only. This figure is a compromise between readability and completeness; many of these outer circle transitions are not shown.

Notice that each state of this example FST has exactly 3 exit transitions, each labelled on the output with one of the 3 phones in the example (x, y and sil). This means that the FST is deterministic w.r.t its output (or, equivalently, has a deterministic inverse).

Notice also that the context dependent input labels are delayed by one phone w.r.t. their corresponding output labels. This is of no concern normally as FST composition tends to push labels out of synchronisation. However, it can cause difficulties with context free phones such as the short pause used in HTK systems.

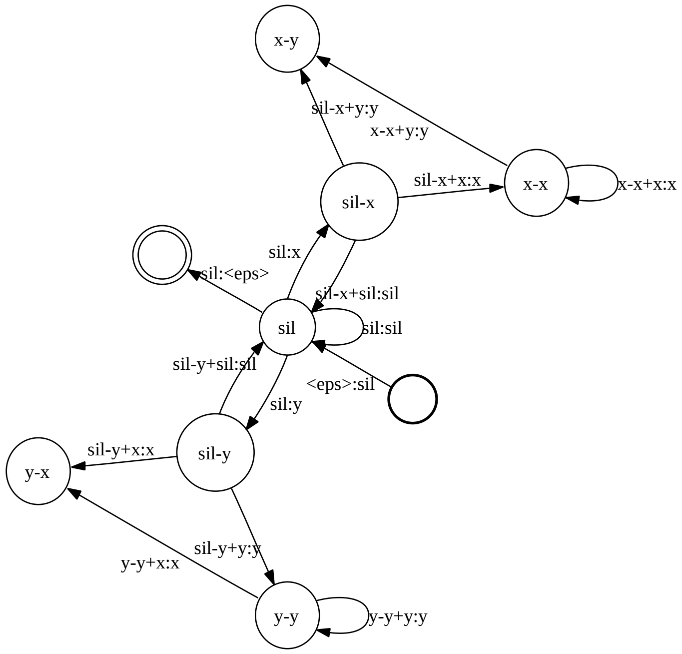


Figure 6: Skeleton of a context dependency FST. Note that many transitions from the outer states are not drawn.

# 4 Weighted Finite State Transducers

## 4.1 Introduction

FSTs as described so far are capable of representing the structural aspects of ASR grammars. The probabilistic aspects are now addressed.

Probability in general is represented rather trivially by associating a weight with each FST transition, the resulting FST being known as a Weighted FST or WFST. The general theory of WFSTs does not specify the meaning of the weights. Rather, it specifies an abstract mathematical concept known as a semi-ring that in turn specifies how weights can be combined during WFST operations, notably composition and determinisation. From the point of view of ASR, the weights are always (natural) logarithms of probabilities, and this confines the discussion to two possible semi-rings:

1. The log semi-ring, where weights are negative log probabilities (i.e., positive numbers) and combine as is usual in ASR software. Weights in series add, corresponding to multiplication of probabilities, and weights in parallel are subject to log-addition (exponential followed by addition and logarithm).

2. The tropical semi-ring, which is identical to the log semi-ring except that parallel weights combine using a 'max' operation. This is based on the assumption that $\log(a+b)$ is approximated by the larger of $\log(a)$ and $\log(b)$. This is related to the score used in Viterbi decoding. The word 'tropical' refers to the fact that its original proponent was Brazilian.

## 4.2 Weight pushing

The addition of weights to a FST introduces difficulties in minimisation. Simply put, if two paths have identical labels but different weights, then they cannot be combined. The solution is to recognise that WFSTs are defined such that the alignment of the weights and labels is not important, just the total weight over a path. Weights can be shifted along a path arbitrarily without changing the function of the WFST.

Weight pushing is the name given to this shifting of weight. It is typically applied in such a way as to make parallel paths identical so they can be combined. Minimisation of WFSTs is hence reliant upon weight pushing.

### 4.3 A probabilistic interpretation

The *Weighted* FST representation has a very useful probabilistic interpretation, aside from that of simply representing probability. In the derivation of equation 2, the lexicon, context and HMM sequences are effectively nuisance variables that are summed over. Although the WFST composition process does not do a complete summation (it only sums over identical labels), it compresses the required summations into a single summation, as shown in equation 6,

$$\sum_{\boldsymbol{l}\in\mathcal{L}}\sum_{\boldsymbol{c}\in\mathcal{C}}\sum_{\boldsymbol{h}\in\mathcal{H}} f(\boldsymbol{a}\mid\boldsymbol{h})\, f(\boldsymbol{h}\mid\boldsymbol{c})\, f(\boldsymbol{c}\mid\boldsymbol{l})\, f(\boldsymbol{l}\mid\boldsymbol{g})\, f(\boldsymbol{g}) = \sum_{\boldsymbol{h}\in\mathcal{H}'} f(\boldsymbol{a}\mid\boldsymbol{h})\, f(\boldsymbol{h}\mid\boldsymbol{g})\, f(\boldsymbol{g}), \tag{6}$$

i.e., the annihilation of labels in composition is entirely analogous to the summation over nuisance parameters.

The meanings of the individual transducers are illustrated in table 1. Notice that the top level $G$

| WFST | Represents |
|---|---|
| $G$ | $f(\boldsymbol{g})$ (or $f(\boldsymbol{g},\boldsymbol{g})$) |
| $L$ | $f(\boldsymbol{l}\mid\boldsymbol{g})$ |
| $C$ | $f(\boldsymbol{c}\mid\boldsymbol{l})$ |
| $H$ | $f(\boldsymbol{h}\mid\boldsymbol{c})$ |
| $L\circ G$ | $f(\boldsymbol{l},\boldsymbol{g})$ or $f(\boldsymbol{l}\mid\boldsymbol{g})\, f(\boldsymbol{g})$ |
| $C\circ L\circ G$ | $f(\boldsymbol{c},\boldsymbol{g})$ or $f(\boldsymbol{c}\mid\boldsymbol{g})\, f(\boldsymbol{g})$ |
| $H\circ C\circ L\circ G$ | $f(\boldsymbol{h},\boldsymbol{g})$ or $f(\boldsymbol{h}\mid\boldsymbol{g})\, f(\boldsymbol{g})$ |

Table 1: Various WFSTs and the probabilities they represent.

WFST represents an unconditional probability, whereas the other WFSTs are conditional densities. Compositions yield joint densities. The joint and conditional densities are particularly appealing in the context of WFSTs as there is a clear relationship with the input and output labels. In this sense, we can think of the WFST $G$ as representing the joint density $f(\boldsymbol{g},\boldsymbol{g})$.

## 5 Stochastic optimisation

### 5.1 Stochasticity

The above probabilistic interpretation of WFSTs suggests how to normalise the probabilities for each of the WFSTs in the composition:

- $G$ represents all possible values of the unconditional probability $f(\boldsymbol{g})$. The sum of the probabilities of *all* distinct paths should be unity.
- $L$ represents all possible values of the conditional probability $f(\boldsymbol{l}\mid\boldsymbol{g})$, with the input and output labels representing $\boldsymbol{l}$ and $\boldsymbol{g}$ respectively. The weights of all distinct paths corresponding to any particular word sequence $\boldsymbol{g}$ should add to unity. This mandates the use of pronunciation probabilities.
- The probabilities represented by $C$ and $H$ are unity as there is a direct mapping from phones to context dependent phones and then to HMM states. In fact, those transducers represent label transductions rather than probabilistic ones.

Generally speaking, in order for the above sum to unity conditions to remain true, the sum of the probabilities of all transitions leaving a particular state should add to unity. This is clear when the FSM is thought of as a generator. At each state a decision is taken about which path to follow, and the sum of the probabilities of each decision should be unity. This is the *stochasticity constraint*. In case of $L$, the stochasticity constraint applies to paths with the same output label, reflecting the conditional probability $f(\boldsymbol{l} \mid \boldsymbol{g})$.

The above also follows trivially for the context dependency transducer, $C$. In this case all probabilities are unity as there is only one possible conditional phone sequence for a given lexical sequence. The HMM transducer, $H$, is trained as a FSM in the first place; all the transition probabilities are normalised naturally.

### 5.2 Stochastic weight pushing

As described above, $G$ and $L$ can certainly be constructed stochastically. However, determinisation, composition and related algorithms do not follow stochasticity constraints. Rather, they make sure that the probability of any particular path is maintained. In this sense, the composition result is not necessarily stochastic. This can be illustrated with reference to figure 5. As drawn, the FST is not weighted. Correct stochastic weighting requires that the transitions $2 \rightarrow 3$ and $2 \rightarrow 4$ have probabilities adding to unity. Notice, however, that WFST theory also allows such weights to be on transitions $3 \rightarrow 6$ or $4 \rightarrow 5$ whilst still maintaining the weight of each path.

The solution to the above is the weight pushing described by Mohri and Riley [8] (also described in their journal articles). Note that weight pushing is integral to optimisation of WFSTs, but this 'stochastic' pushing is distinct. It pushes weights back towards the beginning of the WFST such that stochasticity holds at each state. In this sense, it corrects for the other algorithms being non-stochastic.

Mohri and Riley strongly suggest that stochasticity has important implications for pruning in ASR decoders. It is directly analogous to the language model look-ahead used in conventional decoders [9]. However, in the context of Juicer, on the language models considered so far, we find no speed improvements over conventional weight pushing.

### 5.3 Fudge factors

ASR decoders tend to make use of two so called fudge factors: a language model match factor and an insertion penalty. Together, they define a linear transform that scales and shifts the language model probabilities.

These fudge factors are normally applied to the language model directly, and it is tempting to apply them to the $G$ FST. However, this breaks stochasticity. To maintain stochasticity, they should be applied *after* composition and weight pushing. The multiplicative language model match factor can be applied to all weights. The insertion penalty must be applied to only those transitions with word output labels.

That said, the same comment as for weight pushing applies: In the context of Juicer, on the language models considered so far, we find no speed improvements over applying the fudge factors to the grammar transducer itself.

## 6 Composition walk-through

### 6.1 Introduction

This section presents a walk-through of the construction of a toy stochastic n-gram transducer. The aim is to make explicit some of the techniques summarised earlier, to introduce other techniques that are more explicit by example, and to warn of potential pitfalls.

## 6.2 Grammar

In this example, almost the same ‘YES’ ‘NO’ lexicon as before is used, but this time with a completely hypothetical n-gram grammar:

```
\data\
ngram 1=5
ngram 2=6

\1-grams:
-1.3780 </s>    0.0000
-99.999 <s>     -2.2370
-3.0641 NO      -1.0749
-4.6249 YEAH    -0.6928
-3.6249 YES     -0.6928

\2-grams:
-2.7028 <s> NO
-3.7233 <s> YES
-2.1809 NO </s>
-2.0000 NO NO
-0.7300 YES </s>
-2.3000 YES YES
```

Note that this was generated by taking a real n-gram grammar and removing most of the words. No attempt has been made to normalise the probabilities. The transducer representation of this grammar is shown in figure 7. Note also the slight obfuscation because the ARPA format probabilities are base 10 logarithms whereas the transducer uses negative natural logarithms.

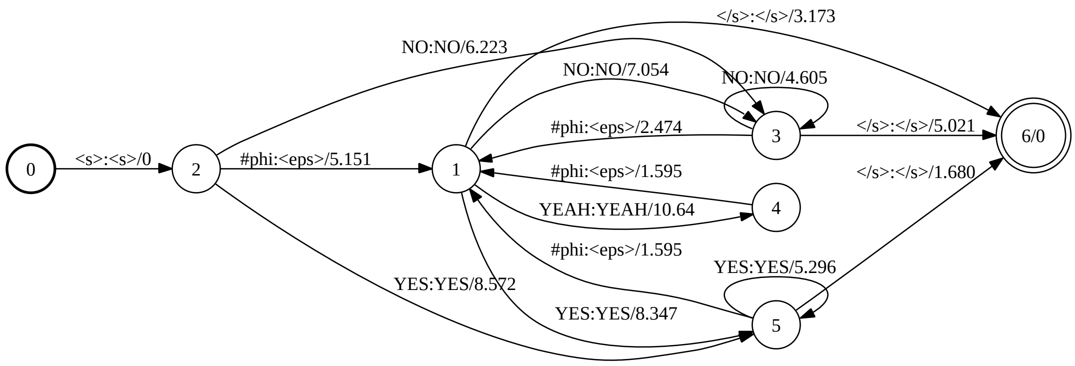


Figure 7: N-gram grammar transducer.

Clearly, the transducer contains as many transitions as there are probabilities in the ARPA format language model. It also contains auxiliary labels; these are to determinise the transducer.

Each back-off transition is associated with an $\epsilon$ output label and a $\phi$ input label.

The function of the $\phi$ labels is illustrated by the following example: In, say, even the simple sequence ‘<s> yes </s>’, the word ‘YES’ could be generated by either of two transitions in the FST in figure 7: $1 \rightarrow 5$ or $2 \rightarrow 5$. The first is a bigram and the second is a back-off ($2 \rightarrow 1$) followed by a unigram.

The $\phi$ label on the back-off transition distinguishes the two alternatives and hence renders the FST deterministic.

## 6.3 Lexicon

A suitable lexicon FST is shown in figure 8. It is essentially like the one in figure 4, except that:

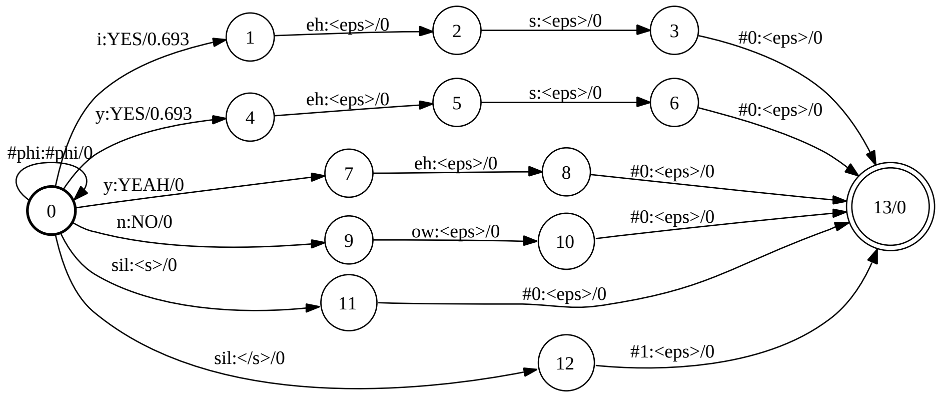


Figure 8: Lexicon transducer with alternative pronunciation.

- There is an extra hypothetical pronunciation of the word 'YES'. This is to illustrate that the weights in such a case should be distributed among the two pronunciations such that the inferred probabilities add to unity ($-\log 0.5 = 0.693$).
- Each pronunciation has a numerical auxiliary symbol. Pronunciations that are otherwise the same are distinguished by different auxiliary symbols. In this case, it is only the input and output tokens, which both have 'silence' pronunciations.
- There is loop on the initial state mapping the auxiliary symbol $\phi$ to itself. This is both to match the $\phi$ transitions in the grammar, $G$, and to propagate them to the composition $L \circ G$.

Note that adding the numerical auxiliary symbols is not as trivial as it may appear. The input dictionary must be indexed by pronunciation rather than just by word in order to count homophones.

In order to be used in the composition, the lexicon must be closed. Figure 9 shows the result of doing this automatically.

Other than closure, no other processing is required. To ease the composition $L \circ G$ it would be helpful if $L$ were deterministic w.r.t. its output labels. However, the existence of multiple pronunciations per word, and the fact that many of the output labels are $\epsilon$ make this impossible. In practise, the non-optimised lexicon is nearly deterministic, and the auxiliary labels are enough to make $L \circ G$ determinisable.

Certainly $L$ should not be determinised w.r.t. its input labels. Although this can drastically reduce its size, the output labels tend to be shifted from their initial positions forcing the composition algorithm to follow many dead-end paths.

## 6.4 Lexicon-Grammar composition

The result of the $L \circ G$ composition is shown in figure 10. This composition includes the steps of

1. Epsilon normalisation. This attempts to remove epsilon transitions, and to group sequences of epsilons together in order to assist later stages.

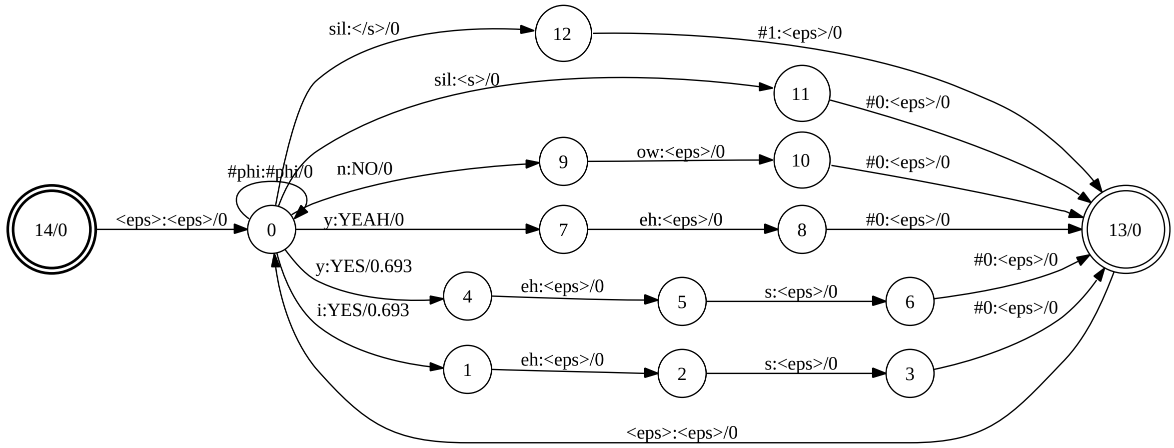


Figure 9: Lexicon transducer with closure transition.

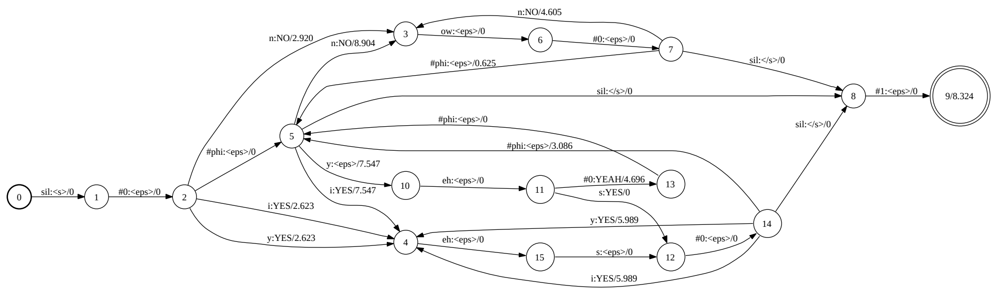


Figure 10: $L \circ G$ with auxiliary symbols.

2. Determinisation.

3. Minimisation. In the case of the AT&T FSM toolkit, this minimisation is applied by first encoding the transducer as an acceptor.

Notice that all the auxiliary labels are still present. At this point they can be removed as described Mohri *et al.* [5]. The reason for this is that the composition

$$C \circ (L \circ G) \tag{7}$$

that follows does not need to result in a deterministic FST. If, however, that FST will be used in the composition

$$H \circ (C \circ (L \circ G)) \tag{8}$$

then the auxiliary labels must be left in place. The result of removing the auxiliary labels is shown in figure 11.

## 6.5 Context dependency

The context dependency FST, $C$, is too complicated to plot in full. Rather, it is almost identical to the one described in figure 6. All weights are set to zero (probability unity) as there is only one possible path for any given phone sequence.

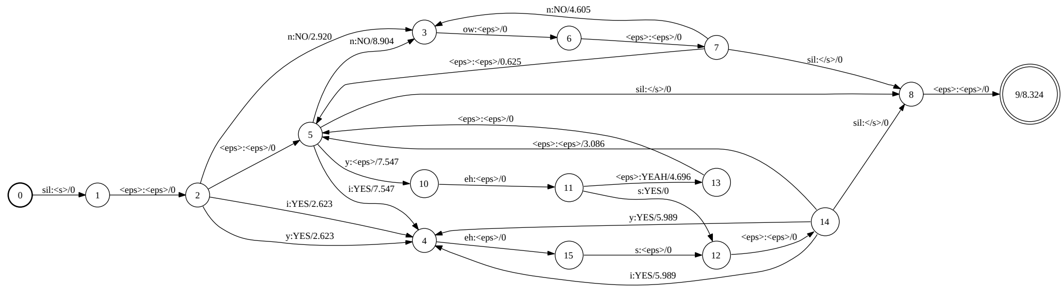


Figure 11: $L \circ G$ with auxiliary symbols replaced by $\epsilon$.

The only difference from the ideal case is, if the auxiliary labels are left in $L \circ G$, then $C$ must also contain transitions to at least match these labels. The easiest way to do this is to put auxiliary label loops on all states in $C$, as illustrated in figure 12.

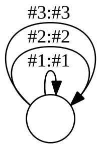


Figure 12: Auxiliary labels on each context dependency FST state.

If it is constructed in the manner described in section 3 then there is no need to determinise the inverse of $C$. It is already deterministic w.r.t. its output labels. However, if the HTK short pause phone is incorporated into $C$ then it may benefit from such determinisation, followed by minimisation as an acceptor.

### 6.6 Context dependency composition

The two input FSTs for the $C \circ (L \circ G)$ composition are now ready and determinised. The composition process depends upon the next stage, and if the auxiliary symbols are still present.

If auxiliary symbols are still present, $C \circ (L \circ G)$ can be determinised and minimised as an acceptor. If, however, the auxiliary symbols were removed at an earlier stage, then the resulting $C \circ (L \circ G)$ is not determinisable.

If this $C \circ (L \circ G)$ is the final FST in the composition process then the weights should also be pushed to re-normalise the FST such that it is once again stochastic.

The resulting FST is shown in figure 13.

## 7 Silence modelling

In an HTK system, as described by Young *et al.* [10], it is normal to have two silence models:

1. A silence model, `sil`, with the same structure as the other phonetic models, and contextually a monophone. i.e., silence acts as a context, but is not itself context dependent.

2. A short pause model, `sp`, that is essentially tied to the silence model, but is context free, and has a 'tee' transition that omits any emitting states.[3]

[3]HTK phonetic models are Moore machines rather than Mealy machines.

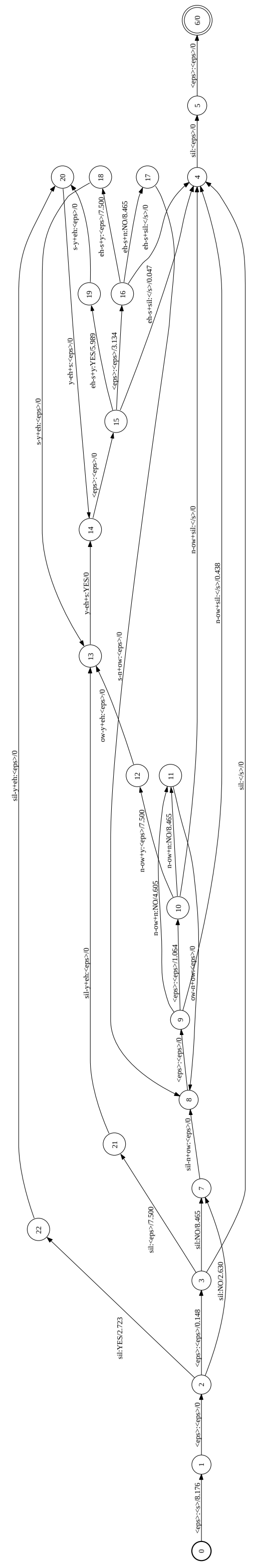


Figure 13: Final $C \circ (L \circ G)$ FST

In general, the silence model is used at the beginning and end of an utterance, or when prescribed in the grammar by a specific token. The short pause model is used in the lexicon at the end of every word to allow optional silence states that do not break context.

The use of the short pause model at the end of each word was revised by Hain *et al.* [11] to advocate the use of both short pause and silence at the word ends. This gives the option of either breaking context between words or not at decode time. In fact, Hain *et al.* distinguish the (third) case where neither silence nor short pause are used, although this is included in the short pause 'tee' structure.

The AT&T team do not mention silence modelling in detail in their WFST publications. However, it is discussed briefly by Allauzen *et al.* [12]; the conclusions can be summarised as:

- Any silence must be associated with a probability, otherwise stochasticity tends to break; notably if silence is included as a free cost loop in the grammar.
- A weighted silence loop can be included after every word using another FST.

The silence class transducer, figure 4 in Allauzen *et al* [12], bears a close similarity to the short pause phone of the HTK system. Both allow zero or more instances of a silence state after each word, where the transition probabilities are learned from data.

It follows from the above that

- Including silence in the grammar is not possible as the grammar does not generally contain probability information for silence.
- Including a short pause model after each word in the lexicon implements both the HTK short pause model as described, and has the same effect as the AT&T silence transducer.
- A silence model can be trivially included after each word in the lexicon in the same manner as the short pause model.

One further issue is the implementation of the 'tee' model. The skip transition can be included either as a transition in the WFST or at the model level in the decoder. In the latter case, the decoder may have to be modified to deal with such transitions.[4]

Three distinct cases are illustrated in figure 14 for just the word 'NO' in the lexicon FST $L$.

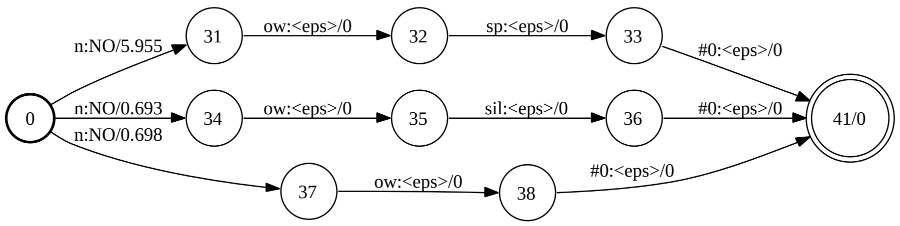


Figure 14: Three pronunciation options accommodating silence for a single word in the lexicon.

## 7.1 Short pause in context dependency

Insertion of the short pause phone in the $C$ is not trivial since the inputs and outputs are not synchronised. The solution is to insert a specific path for each triphone that also delays the short pause. This is illustrated[5] in figure 15.

[4] In Juicer, it's a special case.

[5] Juicer (actually `cdgen`) actually generates the larger of these two representations, so the FST $C$ can be optimised by determinisation of its inverse, contrary to what is written in section 3.

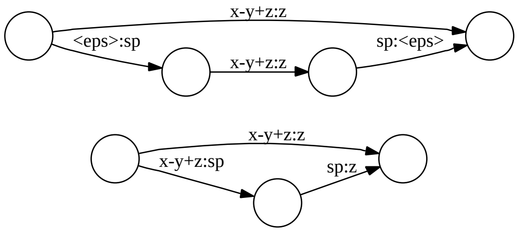


Figure 15: Above: Insertion of sp into context dependent transition in $C$. Below: slightly more compact form.

## 7.2 Experimental evaluation

Experiments were done to test the effects of various silence model strategies. The experimental setup was identical to the WSJ 20K task used by Moore *et al.* [13]. The language model, $G$, is bigram with 1,473,622 transitions.

Table 2 shows the effect of the silence model strategy on the FST size. In particular, the first line shows the size with no silence at all in $L$. The silence overhead is then a percentage measure in excess of this baseline.

Table 2: WFST sizes for various silence model configurations.

| | $L$ | $L \circ G$ | $C \circ L \circ G$ | Silence overhead |
|---|---|---|---|---|
| No silence | 146,244 | 2,434,076 | 2,565,305 | 0% |
| Short pause + FST skip | 312,854 | 2,693,252 | 3,437,116 | 34.0% |
| Silence + FST skip | 312,854 | 2,693,242 | 3,868,745 | 50.8% |
| Short pause | 166,617 | 2,465,654 | 2,800,908 | 9.2% |
| Short pause + silence | 333,327 | 2,676,759 | 4,105,237 | 60.0% |
| Short pause + silence + FST skip | 479,464 | 2,904,369 | 4,762,140 | 85.6% |
| Short pause + silence at start | 333,227 | 2,753,511 | 3,809,777 | 48.5% |
| Short pause at start | 166,617 | 2,453,962 | 2,814,999 | 9.7% |

Mohri *et al.* [5] state that $C \circ L \circ G$ should be just over twice the size of $G$ in terms of transitions for this kind of composition. The results above show some variation around that figure.

Notice that short pause and silence follow different paths in the context dependency transducer, $C$, so their effects in the size of $C \circ L \circ G$ are additive.

Some of the above situations are evaluated in figure 16. To explain the plots in more detail:

1. HDecode was evaluated with 32 tokens per state, a number simply recommended by colleagues, and with acoustic model caching disabled. Acoustic model caching further speeds up HDecode, but is not implemented in Juicer yet. The curve was generated by varying the main beam pruning.

2. Juicer 0.5.0 is the unmodified 0.5.0 decoder with the supplied WFST generation script. Again, the curve is generated by varying main beam pruning. This and the HDecode curves attempt to duplicate those of Moore *et al.* [13].

3. The third plot is the result of various decoder and FST modifications. However, the main beam is optimised at 200 and the phone end beam-width is varied between 75 and 200 to generate the curve. Each pronunciation contains the same 3 silence variants as in the 0.5.0 decoder.

4. The fourth curve is the result of moving the tee model skip into the decoder whilst retaining the silence and short pause pronunciation variants. The phone end beam is varied between 50 and 175.

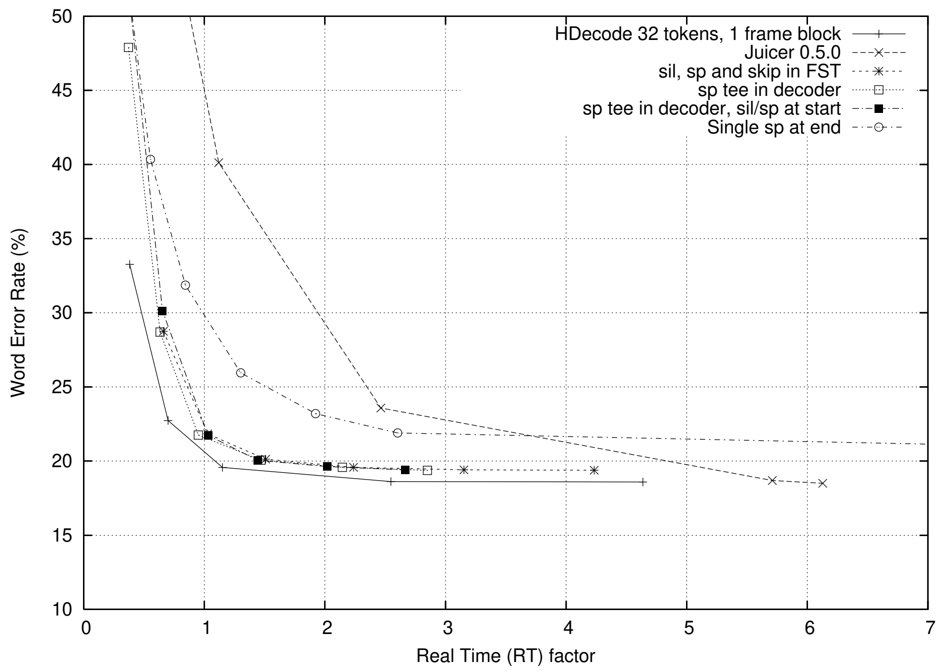


Figure 16: Runtime vs. error rate for various configurations of Juicer, and compared with HDecode.

5. The fifth curve is identical to the fourth except that the silence phones are at the front of each word in the lexicon instead of at the end.

6. The final curve is the performance of the smallest FST in table 2, that is, with just a single short pause at the end of each word. Notice that the performance asymptote is worse as there is now a mismatch between the models and data, but with this taken into account the speed is not significantly different.

## 7.3 Asymptotic performance

Notice that in figure 16 the Juicer 0.5.0 and HDecode curves asymptote below the others (18.59 for HDecode and 18.50 for Juicer 0.5.0). There are three main reasons for this:

1. The language model is changed by adding pronunciation probabilities via the lexicon normalisation.

2. Moving the fudge factors from the grammar FST to the composed FST effectively applies them on top of the pronunciation probabilities.

3. The widest beamwidth is a little tighter (200 instead of 250) for the later Juicer experiments.

In testing Juicer, the 20,000 word open vocabulary language model supplied with the WSJ1 data was used. This language model contains the unknown word token, plus just 2 other out of vocabulary words:

```
<UNK>
'EM
'N
```

The above words were removed, and the language model renormalised, using the HTK `LNorm` utility. Basically, the experiments were performed using the normalised model. However, it is possible to build a WFST for Juicer using the unnormalised model. In this case, the out of vocabulary words are just omitted by the composition process resulting in a non-stochastic WFST. This was the way the 0.5.0 curve was (perhaps erroneously) generated.

# 8 Conclusions and recommendations

## 8.1 Summary

This report has presented a summary of WFST construction techniques. Most techniques are already available in the literature, but the probabilistic interpretation may appear novel to some readers (obvious to others).

Experiments have shown that silence modelling can account for a large part of the WFST size, but does not have a large effect on recognition speed.

Various optimisations have increased the speed of the Juicer WFST based recogniser.

## 8.2 Future work

Future work could take any of several directions:

1. The decoder speed is probably limited by the search space being the whole transducer. HDecode, on the other hand, works by having only one lexicon sized search space and using multiple tokens per state. The multiple tokens allow paths with different histories to exist at the same state. This implicitly performs a type of phone-dependent pruning, limiting the number of hypotheses that can exist for a partucular acoustic state regardless of its position in the grammar.

2. So far, the 'On The Fly' technique of Cheng *et al.* [14] has not been considered. This technique uses multiple tokens per state and could be harmonised with the speed-up suggestion above.

3. Juicer is not an online decoder, i.e., it cannot read directly from an audio card or produce partial hypotheses. It would form a much better basis for demonstration if it could work online.

4. So far, Juicer does not use the $H$ transducer. If experiments showed that composing $H \circ C \circ L \circ G$ rather than $C \circ L \circ G$ were promising, then the decoder could be altered to deal with a state level grammar.

# References


[1] G. Riccardi, R. Pieraccini, and E. Bocchieri. Stochastic automata for language modelling. *Computer Speech and Language*, 10:265–293, 1996.

[2] Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, 16(1):69–88, 2002.

[3] Reinhold Haeb-Umbach and Hermann Ney. Improvements in beam search for 10000-word continuous-speech recognition. *IEEE Transactions on Speech and Audio Processing*, 2(2):353–356, April 1994.

[4] Stephan Kanthak, Hermann Ney, Michael Riley, and Mehryar Mohri. A comparison of two LVR search optimization techniques. In *Proceedings of the International Conference on Spoken Language Processing*, 2002.

[5] Mehryar Mohri, Michael Riley, Don Hindle, Andrej Ljolje, and Fernando Pereira. Full expansion of context dependent networks in large vocabulary speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 1998.

[6] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2), 1997.

[7] Michael Riley, Fernando Pereira, and Mehryar Mohri. Transducer composition for context-dependent network expansion. In *Proceedings of EUROSPEECH*, 1997. Rhodes, Greece.

[8] Mehryar Mohri and Michael Riley. A weight pushing algorithm for large vocabulary speech recognition. In *Proceedings of EUROSPEECH*, 2001. Aalborg, Denmark.

[9] S. Ortmanns, H. Ney, and A. Eiden. Language-model look-ahead for large vocabulary speech recogntiion. In *Proceedings of the International Conference on Spoken Language Processing*, pages 2095–2098, 1996. Philadelphia.

[10] Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying (Andrew) Lui, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, Valtcho Valtchev, and Phil Woodland. *The HTK Book*. Cambridge University Engineering Department, version 3.4 edition, December 2006.

[11] T. Hain, P. C. Woodland, G. Evermann, and D. Povey. The CU-HTK March 2000 hub5e transcription system. In *Proceedings of the Speech Transcription Workshop*, 2000. College Park.

[12] Cyril Allauzen, Mehryar Mohri, Michael Riley, and Brian Roark. A generalized construction of integrated speech recognition transducers. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2004.

[13] Darren Moore, John Dines, Mathew Magimai Doss, Jithendra Vepa, Octavian Cheng, and Thomas Hain. Juicer: A weighted finite-state transducer speech decoder. In *Proceedings of the 3rd Joint Workshop on Multimodal Interaction and Related Machine Learning Algorithms*, 2006.

[14] Octavian Cheng, John Dines, and Mathew Magimai-Doss. A generalized dynamic composition algorithm of weighted finite state transducers for large vocabulary speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 1997. Honolulu.

# A Improvements to Juicer

## A.1 Introduction

This tutorial was written after spending some time evaluating Juicer version 0.5.0. It details some pitfalls that were not completely obvious from reading the literature from AT&T, or at least not reading all of it. This process led to improvements in the speed of Juicer. Most of these improvements (i.e., other than the silence modelling) are not really novel; they fix mistakes and oversights. They are summarised here in the three different categories in which they were addressed.

## A.2 FST structural changes

- Auxiliary $\phi$ labels enabled in the grammar.
- Auxiliary $\phi$ loops enabled in the lexicon.
- Lexicon determinisation and optimisation removed.

- Auxiliary $\phi$ loops enabled in the context dependency transducer.
- Auxiliary labels removed after $L \circ G$ composition.
- Short pause skip removed from lexicon.

### A.3 FST probabilistic changes

- Language model match factor and insertion penalty removed from grammar.
- Lexicon alternative pronunciations normalised.
- Weight pushing explicitly included after composition.

### A.4 Decoder changes

Decoder optimisation focussed on reducing memory usage and avoiding data cache misses. Aside, the 0.5.0 decoder threw errors if word end tokens were not encountered. As the composition phase was altered to remove these, the requirement was also removed from Juicer.

- Avoid reseting hypotheses on pruning.
- Reduce size of buffer for words without end tokens (and this could be completely removed).
- Localise allocation for new tokens in emitting states.
- Add code to follow tee models to next model (basically to handle short pause in the decoder).

## B Drawing the graphs

Graphs are generated using GraphViz, `http://www.graphviz.org`. The basic tool is `dot -Tps` run on the output of `fsmdraw`.

Back-off language models and context dependencies can be drawn using `twopi`, which lays out the nodes in concentric circles. The root node is the context node to which the monophones back off. In practice it's tricky to lay out the graph well, but these runes help:

```
overlap = scale;
nodesep = "0.5"
ranksep = "2.0"
```