INTERSPEECH 2019
September 15–19, 2019, Graz, Austria

Contextual Recovery of Out-of-Lattice Named Entities in Automatic Speech Recognition

Jack Serrino[1], *Leonid Velikovich*[2], *Petar Aleksic*[2], *Cyril Allauzen*[2]

[1]MIT CSAIL
[2]Google

jserrino@mit.edu, {leonid,apetar,allauzen}@google.com

Abstract

As voice-driven intelligent assistants become commonplace, adaptation to user context becomes critical for Automatic Speech Recognition (ASR) systems. For example, ASR systems may be expected to recognize a user's contact names containing improbable or out-of-vocabulary (OOV) words.

We introduce a method to identify contextual cues in a first-pass ASR system's output and to recover out-of-lattice hypotheses that are contextually relevant. Our proposed module is agnostic to the architecture of the underlying recognizer, provided it generates a word lattice of hypotheses; it is sufficiently compact for use on device. The module identifies subgraphs in the lattice likely to contain named entities (NEs), recovers phoneme hypotheses over corresponding time spans, and inserts NEs that are phonetically close to those hypotheses. We measure a decrease in the mean word error rate (WER) of word lattices from 11.5% to 4.9% on a test set of NEs.

Index Terms: speech recognition, contextual speech recognition, named entity recognition

1. Introduction

Recognizing named entities (NEs) such as personal names, locations or events is a particularly challenging task for ASR systems due to sparsity in training data and, often, an abundance of homophones. Contextual signals offer a remedy in the form of dynamic clues as to which words the user is likely to say.

Many techniques have been developed to take advantage of contextual signals. For example, in language model (LM)-based ASR systems, rescoring methods are described that dynamically adjust LM weights; some reweigh n-grams appearing in the user's context on the fly [1, 2, 3], while others dynamically expand the LM via class grammars [4]. Contextual ASR (biasing) methods for end-to-end systems have also been proposed [5, 6]. A wide variety of contextual signals have been studied, including the user's location, past actions, and device state [7, 8].

In this paper, we use context to improve the recognition of contextually-relevant NEs that have otherwise low prior probability or are out-of-vocabulary (OOV) for an underlying ASR system. Our proposed system can use context from a wide variety of sources, and can add contextually relevant NE hypotheses to the output of any ASR recognizer capable of generating a word lattice (an acyclic graph over words of recognition hypotheses). Our system uses finite-state transducers (FSTs) and the OpenFST library [9] to process lattices efficiently.

The rest of the paper is organized as follows. Section 2 describes how our module fits into a larger ASR system. We provide descriptions of our system's three core components in Section 3 and present experimental results in section 4.

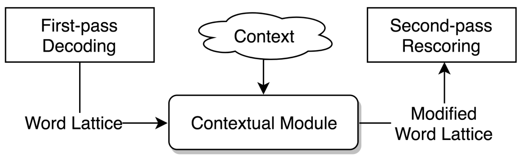

Figure 1: *Our proposed module in a ASR system.*

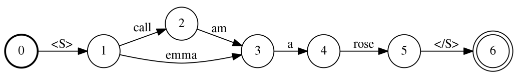

Figure 2: *An input lattice L_{in} from the first-pass decoder.*

2. System Design

In this section, we describe how our contextual module fits into a larger ASR system comprising two passes: a first-pass decoder generates a set of hypotheses, and a second-pass rescorer selects the best hypothesis among these options, as described in [10] and [11]. The decoder may be an FST or a sequence-to-sequence neural network [12], as long as it produces a set of hypotheses representable as a word lattice.

Our module operates between these two passes; it looks for a contextual signal and adds new contextually-relevant hypotheses to the lattice (see Fig. 1) produced by the first pass and sends it to the second pass.

Our approach is inspired by the empirical observations that (1) NEs often occur in common word contexts that are easily recognized by a first-pass decoder (e.g., "call X mobile"); (2) when an NE is misrecognized, the incorrect hypothesis still approximates the ground-truth phonemes spoken by the user. For example, the utterance "call Goudzwaard" may be misrecognized as "call god's word", but the presence of "call" indicates a possible NE, while the phonetics are sufficiently close to match the missing NE based on knowledge of user state (e.g., the name "Goudzwaard" is in the user's contact list).

3. Methodology

This section describes how our module adds contextual hypotheses to an input word lattice L_{in}. In this section, we show our modifications on an example word lattice, shown in Fig. 2.

3.1. Module overview

Our module adds contextual information and hypotheses to word lattices in a three-step process: (1) semantic pattern tagging, (2) phonetic recovery, and (3) hypothesis recovery.

In step 1, we identify word spans likely to contain contextually relevant NEs and mark them in the word lattice with opening and closing tags (e.g., `<contacts> ... </contacts>`). In step 2, we attempt to reconstruct phoneme hypotheses corresponding to the tagged time intervals. In step 3, we search

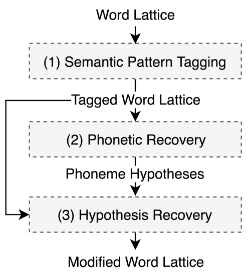

Figure 3: *Block diagram of our contextual module used to modify word lattices.*

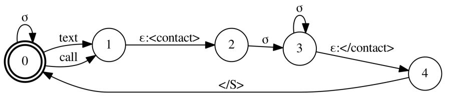

Figure 4: *An example pattern tagging FST with two patterns: "text $CONTACT </S>" and "call $CONTACT </S>"*

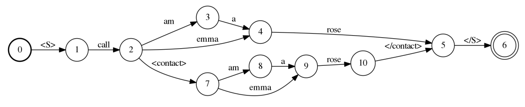

Figure 5: *The tagged lattice L_{tagged} produced composing the pattern tagger in Fig. 4 with the input lattice in Fig. 2*

for NEs phonetically similar to extracted phoneme hypotheses. These NEs are scored and inserted as new hypotheses to the word lattice. A diagram of this process is shown in Fig. 3.

3.2. Semantic pattern tagging

During the semantic pattern tagging step, spans in the ASR word lattice likely to contain contextually relevant NEs are identified. Each tagged span is associated with one or more class of hypothesized NE, such as `song`, `contact`, or `playlist`. Given an input word lattice L_{in}, we produce an augmented word lattice L_{tagged} where each identified span is duplicated, and each new subgraph is surrounded by opening and a closing tags (e.g., `<song>`, `</song>`).

To perform tagging, for each class c, we aggregate a list of patterns P_c in the form of carrier phrases that signal the presence of an NE. A carrier phrase is a sequence of anchoring words and placeholder nonterminal tokens representing the locations of a hypothesized NE. Consider the following example carrier phrase: "play *$FILM* `</S>`". Here, the left anchor is "play", the entity type is `film`, and the right anchor is the end of sentence (`</S>`).

Given a list of semantic patterns P_c for a class c, we construct a tagging FST T_c, see example shown in Fig. 4. When an ASR word lattice is composed with a T_c, a new path is inserted, where words corresponding the nonterminal in the carrier phrase are surrounded with class tags. For example, given the ASR word lattice - "`<S>` Play the beetles `</S>`" - composing with the song tagging FST T_{song} would create an additional path: "`<S>` Play `<song>` the beetles `</song>` `</S>`".

We tag L_{in} by applying all of the T_cs that are relevant to the user's context as follow ($\circ$ denoting FST composition):

$$L_{\text{tagged}} \leftarrow L_{\text{in}} \circ T_{c_1} \circ T_{c_2} \circ \ldots . \tag{1}$$

Note that the presence of semantic patterns may be an indicator that certain user actions or entity types are likely to be found in this utterance. In addition, more complex methods can be used to identify spans containing NEs (such as [13]). Fig. 5 shows the L_{tagged} generated by step 1 from the L_{in} in Fig. 2.

3.3. Phoneme recovery

In the phoneme recovery step, we start with a tagged lattice L_{tagged} and construct hypotheses of spoken phonemes over time intervals corresponding to the tagged spans. The purpose of these hypotheses, represented as phoneme lattices, is to search for phonetically close NEs during the hypothesis recovery step. Although some ASR architectures can generate a phoneme lattice at decoding time, we assume no such lattice is available; this limitation is particularly relevant to sequence-to-sequence decoders, as they may not explicitly model phonemes or other intermediate forms as done by classical recognizers [12].

A naive way to go from a word path to a phoneme lattice is to use a pronunciation lexicon on each of the tagged paths of L_{tagged} as follows:

$$\text{phonemes}_i \leftarrow \Uparrow (\text{LEX} \circ \text{path}_i) \tag{2}$$

where path_i is the tagged word path, LEX maps phoneme sequences to word sequences, and $\Uparrow$ projects FSTs on input labels.

This method is problematic in several ways. By restricting each path to the pronunciations of words originally proposed by the ASR decoder along that specific path – a decision partly guided by prior word probabilities [10] – it may fail to recover the ground-truth phonemes spoken by the user. Worse, because a lattice has a path count bounded exponentially in lattice size, this approach is intractable for thick lattices. We explore three solutions that simplify this problem.

3.3.1. Time interval extraction

Our approach makes use of the chronological time data associated with each state in ASR word lattices. Instead of examining words inside a tagged path, we consider the *time interval* bounded by the path's endpoints. Any word sequences in the lattice that fill this time interval are used to generate phoneme hypotheses:

$$\text{phonemes}_{t_1,t_2} \leftarrow \bigoplus_{\forall p \in \text{interval\_arcs}_{t_1,t_2}} \Uparrow (\text{LEX} \circ p) \tag{3}$$

where $\bigoplus$ is the FST union operation and $\text{interval\_arcs}_{t_1,t_2}$ is the set of all non-overlapping subgraphs in L_{tagged} that fill the time interval $[t_1, t_2)$.

The resulting phoneme lattice allows far more paths than the naive approach: it may admit phoneme sequences from words outside of any tagged span; or even sequences through words that are topologically disconnected in the original ASR lattice. In addition, this approach reduces an exponential computation time to polynomial, as $\text{interval\_arcs}_{t_1,t_2}$ can be computed in polynomial time:

$$L_{\text{flat}} \leftarrow \text{FLATTEN}(L) \tag{4}$$

$$\text{interval\_arcs}_{t_1,t_2} \leftarrow \text{SLICE}(L_{\text{flat}}, t_1, t_2) \tag{5}$$

where FLATTEN collapses all states s at time t into a single state s_t, and SLICE extracts the subgraph of paths starting at s_{t_1} and ending at s_{t_2}. The number of (t_1, t_2) pairs is at most $\frac{|t_U|(|t_U|-1)}{2}$, where $|t_U|$ is the number of unique times associated with states in the lattice. Therefore, the number of SLICE operations is bounded by $\frac{|V|*(|V|-1)}{2}$, where $|V|$ is the total number of states in the word lattice, as each lattice state

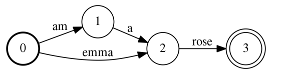

Figure 6: *The word sub-lattice extracted from L_{tagged} in Fig. 5.*

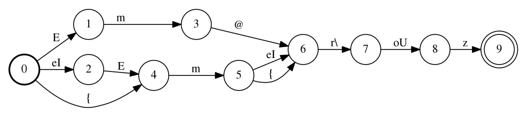

Figure 7: *The word sub-lattice from Fig. 6 transformed into a phoneme lattice.*

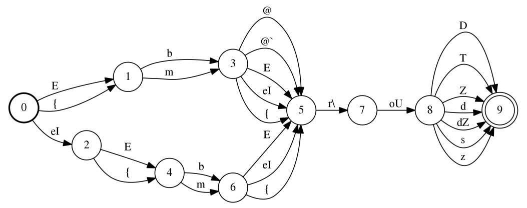

Figure 8: *The phoneme lattice from Fig. 7 after similar phoneme substitution.*

contributes at most one unique time. The number of hypothesis insertion points also becomes polynomial; all tagged paths of a given class from state s_1 to s_2 are reduced to a single task, so the total number of inserted hypotheses is bounded by $|c|\frac{|V|*(|V|-1)}{2}$, where $|c|$ is the number of classes. Fig. 6 shows the words extracted from the example L_{tagged}, and Fig. 7 shows the extracted phonemes.

3.3.2. Substituting similar phonemes

So far, our phoneme hypotheses remain limited to pronunciations of words from the original word lattice, which may not contain the ground-truth phonemes uttered by the user. This is particularly likely with sparse lattices, noisy utterances, or foreign NEs that lack near homophones in the target language. One way to address this is by using an FST that allows acoustically close phonemes to be substituted for one another. We use a phoneme distance matrix computed over phonemes from the X-SAMPA phonetic alphabet [14] from acoustic feature vectors hand-written by linguists (features include place and manner of articulation, height, position, and length). L_2 distance is computed between vectors and low-distance pairs (e.g., /g/ $\sim$ /k/) below a cutoff are used as substitutes. Substitutions are also added for composing or decomposing diphthongs into their component vowel phonemes (e.g., /OI/ $\sim$ /O//I/), and affricates into their component plosive and fricative phonemes (e.g., /tS/ $\sim$ /t//S/). Fig. 8 shows the phonetic lattice from Fig. 7 after similar phoneme substitution.

3.3.3. Phoneme edit distance expansion

In practice, even similar phoneme substitution may not recall all ground-truth phonemes. A more drastic measure is to admit phoneme sequences that are k edits - insertions, deletions, or substitutions - away from the phoneme lattice. We modify the edit distance algorithm in [15] to efficiently limit matching depth; our *edit-distance-k expander* FST E_k with states $s_0, \ldots, s_k$ is constructed as follows: self-loop from s_d to s_d with labels $\sigma : \sigma$ (exact match), transitions from s_d to s_{d+1} with labels $\epsilon : \rho$ (insertion), $\sigma : \epsilon$ (deletion) and $\sigma : \rho$ (substitution).

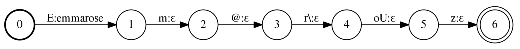

Figure 9: *The* `contacts` *phoneme-to-entity lexicon $L_{contacts}$.*

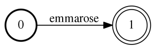

Figure 10: *The NE is recovered by entity matching. We compose the phoneme FST in Fig. 8 with the lexicon FST in Fig. 9.*

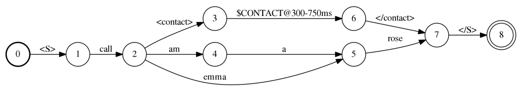

Figure 11: *L_{tagged} from Fig. 5 after replacing the tagged time interval with a replacement arc.*

The phoneme lattice is composed with E_k using a matcher that allows σ labels in E_k to match and consume any label:

$$\text{phonemes\_edit}_{t_1,t_2,k} \leftarrow \text{phonemes}_{t_1,t_2} \circ_\sigma E_k \qquad (6)$$

where $\circ_\sigma$ represents composition using this special matcher. This approach overcomes poor recall by expanding the phoneme space indiscriminately in all directions. As k increases, however, the number of possible phonetic paths grows exponentially, burdening the later hypothesis recovery step with many low-quality phonetic hypotheses. For this reason, we limit our experiments to edit distances of at most 4.

3.4. Hypothesis recovery

The final step in our module is hypothesis recovery, which aims to suggest contextually relevant NEs for given phoneme hypotheses. This step involves searching for NEs in agreement with the phoneme hypotheses; scoring the matched NEs; and augmenting the ASR word lattice with new NEs.

3.4.1. NE matching

Matching is performed by composing each phoneme lattice with a specialized class-based lexicon FST.

We begin with a list of contextual NEs W_c belonging to a class. For example, given the `contacts` class for a given user, the NEs might include the first, last, and first+last names of the contacts in the user's contact list. We construct a lexicon FST L_c from W_c that transduces the pronunciation of each NE in the list to the NE itself. Then for each tagged time interval a of L_{tagged} with class c, we perform phoneme recovery to produce a phoneme hypothesis lattice P_a. A match lattice $M_{a,c}$ containing NEs of class c for time interval a is computed as:

$$M_{a,c} \leftarrow\Downarrow (P_a \circ L_c) \qquad (7)$$

where L_c is the FST for class c corresponding to the tags. Or, when using edit distance expansion:

$$M_{a,c} \leftarrow\Downarrow ((P_a \circ_\sigma E_k) \circ_\rho L_c) \qquad (8)$$

where $\Downarrow$ projects FSTs on output labels, $\circ_\sigma$ allows σ labels in the left operand to consume any label in its right operand, and $\circ_\rho$ allows ρ labels in the left operand to consume any *unmatched* label in its right operand. Fig. 8 (from step 2) and Fig. 9 show an example P_a and L_c for this procedure, respectively. The result (Fig. 10) is a list of new hypotheses for each tagged area in FST form that can be added to the final output lattice.

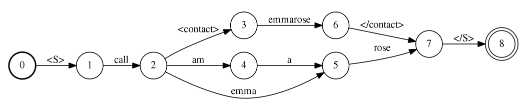

Figure 12: *The final output lattice, L_{out}.*

3.4.2. Hypothesis scoring

Each new NE hypothesis is assigned a probabilistic score so it can be ranked with other NEs and existing hypotheses in the word lattice. Note this step also acts as contextual biasing of first-pass LM probabilities, as NEs already present in the lattice may get suggested again with a better score. To compute the score, observe that the noisy channel ASR model uses the un-normalized pseudo-probability $P'(w|a) \propto P(a|w)P(w)$ to compute argmax$w$ as the recognition result, where w is a word hypothesis and a are the acoustic observations. Assuming that scores in the ASR word lattice track this quantity, we introduce the following compatible variant:

$$P'(h|a,c) \propto \sum_{w} P(a|w)P(w|h)P(h|c) \qquad (9)$$

where h is the hypothesized NE; a are acoustic observations over h's time interval; w is any word hypothesis in this timespan; and c is contextual information including matched semantic patterns from 3.3 and knowledge of user/device state. These quantities are computed as follows: $P(a|w)$ are the decoder acoustic scores, if available, or else estimated from generic word scores; $P(w|h)$ is estimated as the probability of confusing the phoneme sequences of h and w – a distance function based on the operations applied from 3.3.2; $P(h|c)$ is the prior likelihood of this NE getting invoked in the given context.

3.4.3. Hypothesis addition

We take the matched lattices $M_{a,c}$ for each class c corresponding to a tagged span for time interval a and add them to L_{tagged} to give a final output lattice L_{out}, augmented with new hypotheses. We achieve this by creating a placeholder arc in the original lattice and expanding it with new NEs using the Replace operation defined in [16] (Fig. 11 and 12).

4. Experimental results

In this section, we evaluate our module's effect on word error rate (WER), sentence accuracy (SACC) and system latency.

4.1. Corpus

We show results on a test set designed to simulate human use of personal contact names for voice actions. 2,000 anonymized human-spoken utterances were chosen whose transcript contains one of 84 grammatical patterns such as "call `$CONTACT` mobile", "hi `$CONTACT`", "send `$CONTACT` text", etc., where `$CONTACT` may contain a known first name and/or known last name, for a combined length of one to two words.

For each utterance, the provided context is a "phonebook" of 200 contact names. One of the names is mentioned in the utterance and the other 199 are chosen randomly from the same test set; they act as distractors.

4.2. Performance measurements

Table 1 shows the performance of the NE recovery module. Here and onwards, 'E' refers to maximum edit distance and

Table 1: *The WER and SACC of our module for various edit distances (E) and with/without similar phoneme substitution*

E	SPS	SACC [%]	WER [%]
w/o context	w/o context	54.90	15.06
baseline	baseline	64.85	11.48
0	No	86.30	6.72
0	Yes	87.05	6.37
2	No	89.20	5.14
2	Yes	89.35	5.10
4	No	89.70	4.92
4	Yes	89.70	4.92

Table 2: *Examples of hypotheses recovered using our module.*

Param.		*Best Scoring Hypothesis*	
E	SPS	Baseline	After NE Recovery
0	No	"who is ryan"	"who is ryne"
0	Yes	"name is a free"	"name is avrie"
1	Yes	"tell him you're a joke"	"tell amyrah a joke"

'SPS' to similar phoneme substitution being enabled. The baseline in our experiments is a state-of-the-art contextual biasing system described in [1] provided with the same contexts. We also include test results with no context.

With no SPS or edit distance, SACC improves from 54.9% to 86.3%. This means that about 70% of misrecognized utterances already recall the ground-truth phonemes in the word lattice, and the module is able to recover the correct sequence; and match, insert (if not present) and score the correct NE over other distractors. The remaining 30% of misrecognitions are harder due to ground-truth phonemes missing from the lattice, or distractors being scored higher than the correct NE.

SPS handles some cases of the missing phonemes, but its influence diminishes rapidly when edit distance is turned up. We observed that edit distance rendered SPS unnecessary at higher values. It may be worth repeating this experiment with data-driven phoneme distances. E=4 provided our best results of SACC=89.7%. Higher values provided negligible benefit at a cost of significant latency. Examples of improvements to word lattices are shown in Table 2.

4.3. Latency measurements

To compute the effect our module has on latency in an end-to-end ASR pipeline, we measure the runtime of our module on utterances from the test set using dual Intel Xeon E5-2690 2.60GHz CPUs with 24 cores and 64 GB of RAM. We found that with E$\leq$3 and SPS disabled, median latency was in the 2-5 ms range with 90th percentiles below 15ms. Higher values of E and enabling SPS increased these figures, and latency generally increases with the size of the search space.

5. Conclusion

We described a method of using out-of-lattice phoneme and NE recovery to decrease the WER by up to 57.1% relative in an ASR system. Our module performs tagging of relevant subgraphs, phoneme recovery, and NE matching to add missing contextually relevant NEs to word lattices. Our system is agnostic to the design of the underlying first-pass ASR system, handles OOV words, and allows for a trade-off between latency and search accuracy through the adjustment of hyper-parameters.

6. References

[1] P. S. Aleksic, M. Ghodsi, A. H. Michaely, C. Allauzen, K. B. Hall, B. Roark, D. Rybach, and P. J. Moreno, "Bringing contextual information to google speech recognition," in *INTERSPEECH*, 2015.

[2] J. Scheiner, I. Williams, and P. Aleksic, "Voice search language model adaptation using contextual information," in *2016 IEEE Spoken Language Technology Workshop (SLT)*, Dec 2016, pp. 253–257.

[3] K. B. Hall, E. Cho, C. Allauzen, F. Beaufays, N. Coccaro, K. Nakajima, M. Riley, B. Roark, D. Rybach, and L. Zhang, "Composition-based on-the-fly rescoring for salient n-gram biasing," in *INTERSPEECH*, 2015.

[4] L. Vasserman, B. Haynor, and P. Aleksic, "Contextual language model adaptation using dynamic classes," in *2016 IEEE Spoken Language Technology Workshop (SLT)*, Dec 2016, pp. 441–446.

[5] I. Williams, A. Kannan, P. Aleksic, D. Rybach, and T. N. Sainath, "Contextual speech recognition in end-to-end neural network systems using beam search," in *Proc. of Interspeech*, 2018.

[6] Z. Chen, M. Jain, Y. Wang, M. L. Seltzer, and C. Fuegen, "End-to-end contextual speech recognition using class language models and a token passing decoder," *arXiv preprint arXiv:1812.02142*, 2018.

[7] P. Aleksic, C. Allauzen, D. Elson, A. Kracun, D. M. Casado, and P. J. Moreno, "Improved recognition of contact names in voice commands," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2015, pp. 5172–5175.

[8] A. Michaely, J. Scheiner, M. Ghodsi, P. Aleksic, and Z. Wu, "Unsupervised context learning for speech recognition," in *Spoken Language Technology (SLT) Workshop*, 2016.

[9] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "Openfst: A general and efficient weighted finite-state transducer library," 2007.

[10] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," 2001.

[11] G. Saon, D. Povey, and G. Zweig, "Anatomy of an extremely fast lvcsr decoder," September 2005. [Online]. Available: https://www.microsoft.com/en-us/research/publication/anatomy-of-an-extremely-fast-lvcsr-decoder/

[12] C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, K. Gonina, N. Jaitly, B. Li, J. Chorowski, and M. Bacchiani, "State-of-the-art speech recognition with sequence-to-sequence models," *CoRR*, vol. abs/1712.01769, 2017. [Online]. Available: http://arxiv.org/abs/1712.01769

[13] C. Parada, M. Drezde, and F. Jelinek, "OOV sensitive named-entity recognition in speech," in *INTERSPEECH*, 2011, pp. 2085–2088.

[14] J. Wells, "Computer-coding the ipa: a proposed extension of sampa," 04 1995.

[15] M. Mohri, "Edit-distance of weighted automata: General definitions and algorithms," *International Journal of Foundations of Computer Science*, vol. 14, no. 06, pp. 957–982, 2003. [Online]. Available: https://doi.org/10.1142/S0129054103002114

[16] "OpenFST: ReplaceFST Documentation," http://www.openfst.org/twiki/bin/view/FST/ReplaceDoc.