Finite-state text processing

Kyle Gorman

Outline

- Motivations
- State machines
- Formalization
- Rational relations
- Finite-state transducers
- Break (?)
- Composition
- Rewrites
- OpenFst and friends
- Some new(ish) algorithms

Motivations

Kleene 1956

Initially, the study of

- abstract computational devices known as *state machines* and
- formal languages

were considered independent of one another. Kleene (1956) was one of the first to unify these two areas of study. Kleene wished to characterize the properties of *nerve nets* (McCulloch and Pitts, 1943), a primitive form of artificial neural network. In doing so, Kleene introduced the regular languages and established strong connections between regular languages and the *finite acceptors*, a type of state machine.

Introducing state machines

State machines

A *state machine* is hardware or software whose behavior can be described solely in terms of a set of *states* and *arcs*, transitions between those states. In this formalism, states roughly correspond to “memory” and arcs to “operations” or “computations”. A *finite-state machine* is merely a state machine with a finite, predetermined set of states and labeled arcs.

As directed graphs

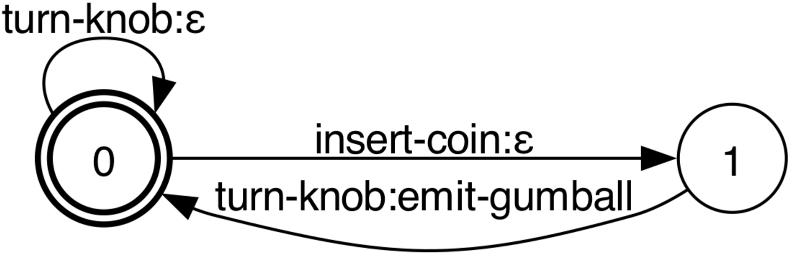
State machines are examples of what computer scientists call *directed graphs*. These are “directed” in the sense that the existence of an arc from state q to state r does not imply an arc from r to q. In *state diagrams*, we indicate this directionality using arrows.

(image: credit: Wikimedia Commons)

The humble gumball machine

One familiar example of a state machine—encoded in hardware, rather than software—is the old-fashioned gumball machine. Each state of the gumball machine is associated with actions such as

- turning the knob,
- inserting a coin, or
- emitting a gumball.


turn-knob:ε
insert-coin:ε
turn-knob:emit-gumball
0
1

Formalization

Sets

A set is an abstract, unordered collection of distinct objects, the *members* of that set. By convention capital Italic letters denote sets and lowercase letters to denote their members. Set membership is indicated with the $\in$ symbol; e.g., $x \in X$ is read "x is a member of X". The empty set is denoted by $\varnothing$.

Subsets

A set X is said to be a *subset* of another set Y just in the case that every member of X is also a member of Y. The subset relationship is indicated with the $\subseteq$ symbol; e.g., $X \subseteq Y$ is read as "X is a subset of Y". Every set is a subset of itself; e.g., $X \subseteq X$.

Union and intersection

- The *union* of two sets, $X \cup Y$, is the set that contains just those elements which are members of X, Y, or both.

$$X \cup Y = \{x \mid x \in X \vee x \in Y\}$$

- The *intersection* of two sets, $X \cap Y$, is the set that contains just those elements which are members of both X and Y.

$$X \cap Y = \{x \mid x \in X \wedge x \in Y\}$$

Strings

Let Σ be an *alphabet* (i.e., a finite set of symbols). A *string* (or *word*) is any finite ordered sequence of symbols such that each symbol is a member of Σ. By convention typewriter text is used to denote strings. The empty string is denoted by ϵ (*epsilon*). String sets are also known as *languages*.

Concatenation and closure

- The *concatenation* of two languages, $X\ Y$, consists of all strings formed by concatenating a string in X with a string in Y.

$$X\ Y = \{xy \mid x \in X \wedge y \in Y\}$$

- The *closure* of a language, X^*, is an infinite language consisting of zero or more “self-concatenations” of X with itself.

$$\begin{aligned} X^* &= \{\epsilon\} \cup X^1 \cup X^2 \cup X^3 \ldots \\ &= \{\epsilon\} \cup X \cup XX \cup XXX \ldots \end{aligned}$$

Regular languages

- The empty language $\emptyset$ is a regular language.
- The empty string language $\{\epsilon\}$ is a regular language.
- If $s \in \Sigma$, then the singleton language $\{s\}$ is a regular language.
- If X is a regular language, then its closure X^* is a regular language.
- If X, Y are regular languages, then:
  - their concatenation XY is a regular language, and
  - their union $X \cup Y$ is a regular language.
- Other languages are not regular languages.

Regular languages in the 20th century

- Regular languages were popularized in part by discussion of the *Chomsky*(-*Schützenberger*) hierarchy (e.g., Chomsky and Miller, 1963).
- Regular languages were used by Thompson (1968) to create the `grep` regular expression matching utility.
- Finite acceptors are used to compactly store morphological dictionaries.
- Finite acceptors are used to compactly represent *language models*, particularly in speech recognition engines.

It now seems that an enormous amount of linguistically-interesting phenomena can be described in terms of regular languages (and the closely-related *rational relations*).

Negative results

At the same time, there were two important negative results:

- Syntactic grammars belong to a higher-classes of formal languages, the *mildly context-sensitive languages* (Vijay-Shanker et al., 1987).
- The class of regular languages are not “learnable” from positive data under Gold’s (1967) notion of *language identification in the limit*.

In practice, this means that regular languages and finite acceptors are somewhat limited as models of syntax, though they are still well-suited as models of phonology and morphology.

Cross-product

A *pair* or *two-tuple* is a sequence of two elements; e.g., (a, b) is the pair consisting of a then b. The *cross-product* (or *Cartesian product*) of two sets, $X \times Y$, is the set that contains all pairs (x, y) where x is an element of X and y is an element of Y.

$$X \times Y = \{(x, y) \mid x \in X \wedge y \in Y\}$$

Relations

A (*two-way* or *binary*) *relation* over sets X and Y is a subset of the cross-product $X \times Y$. By convention lowercase Greek letters indicate relations, and the *domain*—set of inputs—and *range*—the set of outputs—are usually provided upon first definition. For example, the "less than" relation might be written $\lambda \subseteq \mathbb{R} \times \mathbb{R} = \{(x, y) \mid x < y\}$ where $\mathbb{R}$ is the set of real numbers.

Functions

A *function* is a relation for which every element of the domain is associated with exactly one element of the range.

Problem

Let $\mathbb{R}$ be the set of real numbers, and $\mathbb{N}$ be the set of natural numbers. Then, are the following relations functions?

- The "less than" relation $\lambda \subseteq \mathbb{R} \times \mathbb{R} = \{(x, y) \mid x < y\}$?
- The "successor" relation $\sigma \subseteq \mathbb{N} \times \mathbb{N} = \{(x, x + 1) \mid x \in \mathbb{N}\}$?

Solution

- λ is not a function because there are an infinitude of real numbers that are greater than any other real number.
- σ is a function because each natural number has just one successor.

N-ary relations

Three-, four- and five-way relations, and so on, are all well-defined, though there is no such generalization for functions, since n-way relations where $n > 2$ lack well-defined domain and range. However, one can redefine any n-way relation into a two-way relation by grouping the various sets into domain and range; for instance, a four-way relation over $A \times B \times C \times D$ can be redefined as a two-way relation (and possibly, a function) with domain $A \times B$ and range $C \times D$.

Application

The application of an input argument to a relation or function is indicated using square brackets. For instance given the successor function σ, then $\sigma[3] = \{4\}$ because $(3, 4) \in \sigma$.

Finite-state acceptors

An *finite-state acceptor* (FSA) is a 5-tuple defined by:

- a finite set of states Q,
- a *start* or *initial* state $s \in Q$,
- a set of *final* or *accepting* states $F \subseteq Q$,
- an *alphabet* Σ, and
- a *transition relation* $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$.

Note that, as formalized here, there is exactly one start state but may be multiple final states, and that the start state may also be a final state.

Acceptance

An FSA is said to *accept*, *match*, or *recognize* a string if there exists a path from the initial state to some final state, and the labels of the arcs traversed by that state correspond to the string in question. The set of all strings so accepted are called the FSA's language.

Paths

Given two states $q, r \in Q$ and a symbol $z \in \Sigma \cup \{\epsilon\}$, $(q, z, r) \in \delta$ implies that there is an arc from state q to state r with label z. A *path* through a finite acceptor is a pair of

- a state sequence $q_1, q_2, \ldots, q_n \in Q^n$ and a
- a string $z_1, z_2, \ldots, z_n \in (\Sigma \cup \{\epsilon\})^n$,

subject to the constraint that $\forall i \in [1, n] : (q_i, z_i, q_{i+1}) \in \delta$; that is, there exists an arc from q_i to q_{i+1} labeled z_i.

Complete paths

A path is said to be *complete* if

- $(s, z_1, q_1) \in \delta$ and
- $q_n \in F$.

That is, a complete path must also begin with an arc from the initial state s to q_1 labeled z_1 and terminate at a final state. Then, an FSA accepts string $z \in (\Sigma \cup \{\epsilon\})^*$ if there exists a complete path with string z.

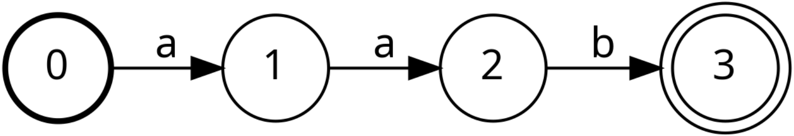
Kleene's theorem

Kleene's theorem holds that any regular language is accepted by an FSA, and any language accepted by an FSA is a regular language. This implies that because regular languages are closed under closure, concatenation, and union, so are FSAs.

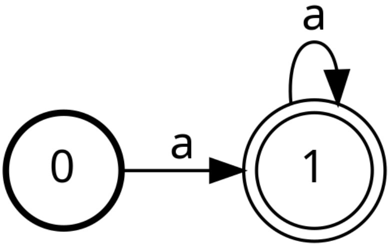
Reading the state diagrams

- States are indicated by circles.
- The initial state is indicated by a bold circle.
- Final states are indicated by double-struck circles.
- Labeled arrows indicate arcs.

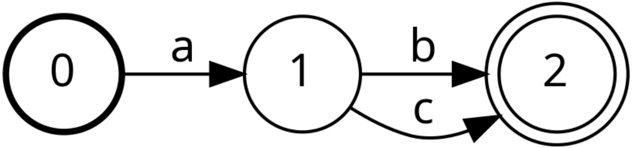
$\{aab\}$



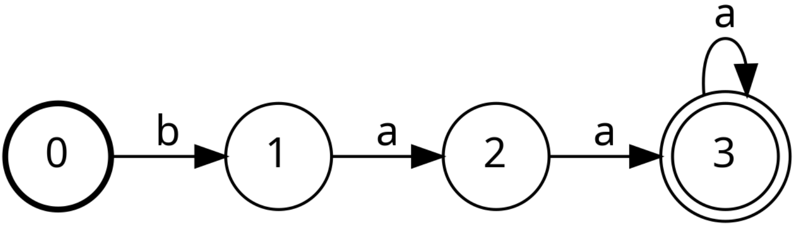
$\{a\}^+$



$\{a\}(\{b\} \cup \{c\})$



$\{ba\}\{a\}^{+}$



http://wellformedness.com/courses/fstp/

The sheep language

- $Q = \{0, 1, 2, 3\}$
- $s = 0$
- $F = \{3\}$
- $\Sigma = \{\mathrm{a}, \mathrm{b}\}$
- $\delta = \{(0, \mathrm{b}, 1), (1, \mathrm{a}, 2), (2, \mathrm{a}, 3), (3, \mathrm{a}, 3)\}$

All about ϵ

The ϵ symbol is a special one which does not match/consume any other symbol. Every ϵ-FSA has an equivalent ϵ-free (or “e-free”) FSA that can be found using the epsilon-removal algorithm (Mohri, 2002a).

Rational relations

Cross-product (redux) and rational relations

Recall that a *cross-product* (or *Cartesian product*) of two sets, $X \times Y$, is the set that contains all pairs (x, y) where x is an element of X and y is an element of Y.

$$X \times Y = \{(x, y) \mid x \in X \wedge y \in Y\}$$

Then, a rational relation is a subset of the cross-product of two regular languages (e.g., $\gamma \subseteq A \times B$).

Example: state abbreviations

$$
\begin{aligned}
\gamma = \{&(\mathsf{AK}, \texttt{Alaska}),\\
&(\mathsf{AL}, \texttt{Alabama},\\
&(\mathsf{AR}, \texttt{Arkansas}),\\
&(\mathsf{AZ}, \texttt{Arizona}),\\
&(\mathsf{CA}, \texttt{California}),\\
&(\mathsf{CO}, \texttt{Colorado}),\\
&(\mathsf{CT}, \texttt{Connecticut}),\\
&(\mathsf{DE}, \texttt{Delaware}),\\
&\ldots\}
\end{aligned}
$$

Interpretation

Regular languages are *languages*, or sets of strings. Rational relations, in turn, can either be thought of as

- sets of pair of (input and output) strings, or
- mappings between input and output strings.

Thus, we might say either that

- $(\mathrm{OH}, \texttt{Ohio}) \in \gamma$, or
- $\gamma[\{\mathrm{OH}\}] = \{\texttt{Ohio}\}$.

Finite-state transducers

Finite-state transducers

Finite-state transducers (FSTs) are generalizations of finite-state acceptors which correspond to the rational relations. An FST is a 6-tuple defined by

- a finite set of states Q,
- a *start* or *initial* state $s \in Q$,
- a set of *final* or *accepting* states $F \subseteq Q$,
- an ***input alphabet*** Σ,
- an ***output alphabet*** Φ, and
- a ***transition relation*** $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Phi \cup \{\epsilon\}) \times Q$.

Transduction

An FST is said to *transduce* or *map* from $x \in (\Sigma \cup \{\epsilon\})^*$ to $y \in (\Phi \cup \{\epsilon\})^*$ so long as a complete path with input string x and output string y exists.

Paths

Given two states $q, r \in Q$, input symbol $x_i \in \Sigma \cup \{\epsilon\}$, and output symbol $y_i \in \Phi \cup \{\epsilon\}$, $(q, x_i, y_i, r) \in \delta$ implies that there is an arc from state q to state r with input label x_i and output label y_i. A *path* through a finite transducer is a triple consisting of

- a state sequence $q_1, q_2, q_3, \ldots \in Q^n$ and a
- a input string $x_1, x_2, x_3, \ldots \in (\Sigma \cup \{\epsilon\})^n$,
- a output string $y_1, y_2, y_3, \ldots \in (\Phi \cup \{\epsilon\})^n$,

subject to the constraint that $\forall i \in [1, n] : (q_i, x_{i+1}, y_{i+1}, q_{i+1}) \in \delta$; that is, there exists an arc from q_i to q_{i+1} labeled $x_{i+1} : y_{i+1}$.

Complete paths

A path is said to be *complete* if

- $(s, x_1, y_1, q_1) \in \delta$ and
- $q_n \in F$.

That is, a complete path must also begin with an arc from the initial state s to q_1 labeled $x_1 : y_1$ and terminate at a final state. Then, an FST transduces input string $x \in (\Sigma \cup \{\epsilon\})^n$ to output string $y \in (\Phi \cup \{\epsilon\})^n$ if there exists a complete path with input string x and output string y.

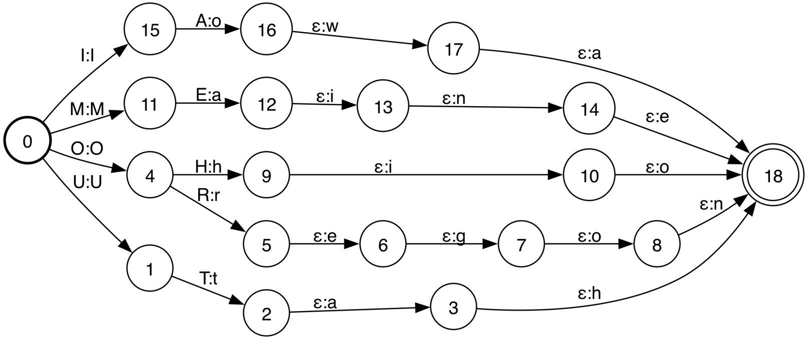
FSAs as FSTs

FSAs can be thought of as a special case of FSTs where every transition has the same input and output label. This is why, in Pynini and friends, FSAs are instance of a class called `Fst`.

Even more about ϵ

FSTs can map between strings of different lengths, but one must use ϵs to "pad out" the shorter string. Thus, whereas every FSA has an equivalent "e-free" FSA, not all ϵ-FSTs have an equivalent "e-free" form. Thus, when one applies the ϵ-removal algorithm (e.g., Pynini's `rmepsilon` method) to FSTs, it simply removes $\epsilon : \epsilon$ arcs.

State abbreviations (fragment)



Rational operations over FSTs

Rational relations—and thus FSTs—are closed under closure, concatenation, and union, and the Thompson (1968) constructions for these operations are also appropriate to FSTs.

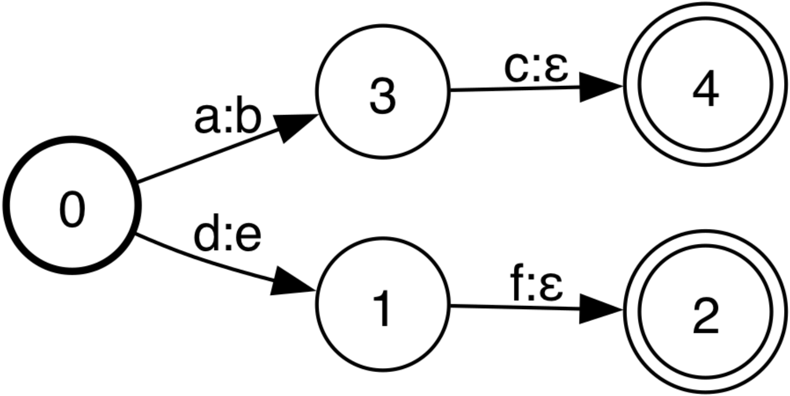
Projection

Projection converts a FST to an FSA that is either equal to its domain (*input-projection*) or range (*output-projection*). By convention, input-projection is indicated by the prefix operator π_i and output-project by π_o. Projection can be computed simply by copying all input (resp. output) labels onto the output (resp. input) labels along each arc.

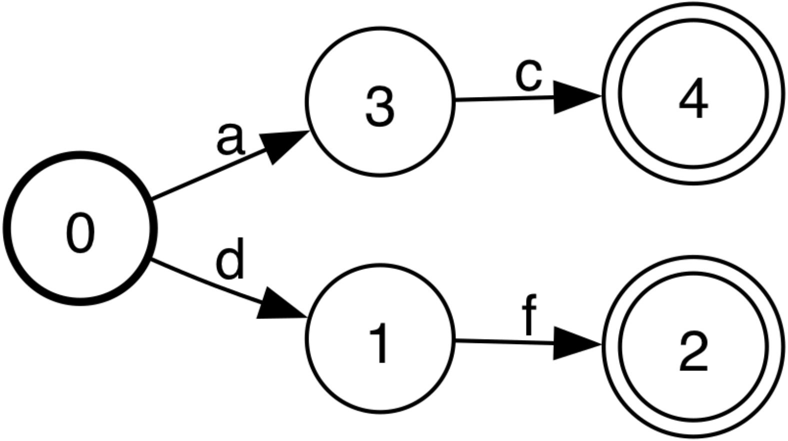
Inversion

Inversion swaps the domain and range of an FST. By convention, it is indicated by a superscripted -1. Inversion can be computed simply by swapping input and output labels along each arc.

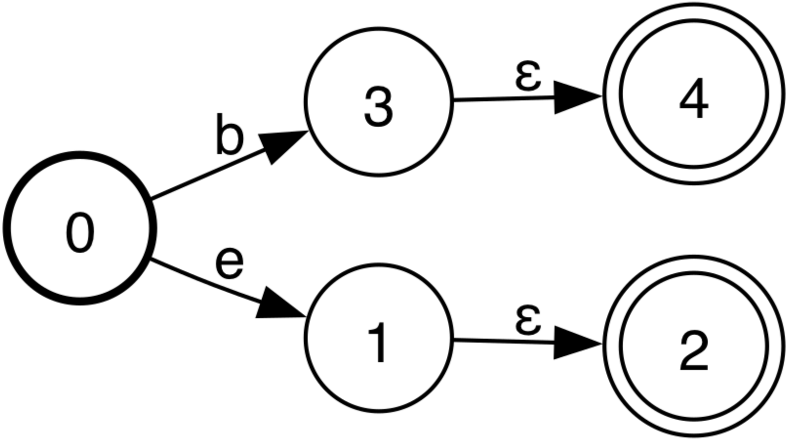
$(\{\mathtt{ac}\} \times \{\mathtt{b}\}) \cup (\{\mathtt{df}\} \times \{\mathtt{e}\})$



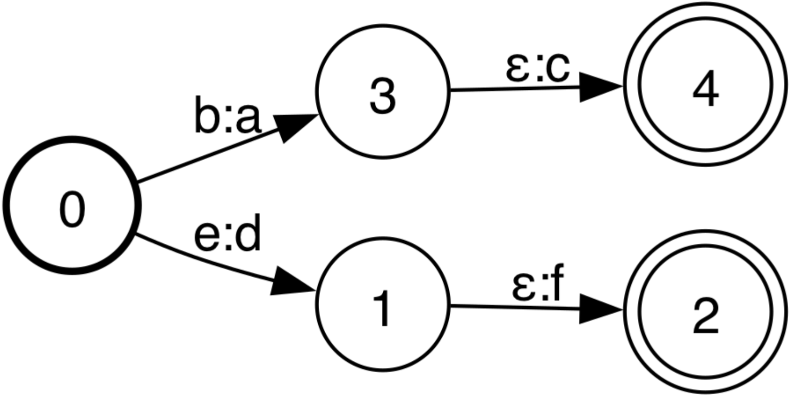
$$\pi_i\left((\{\mathtt{ac}\}\times\{\mathtt{b}\})\cup(\{\mathtt{df}\}\times\{\mathtt{e}\})\right)$$



$\pi_o\left((\{\mathtt{ac}\} \times \{\mathtt{b}\}) \cup (\{\mathtt{df}\} \times \{\mathtt{e}\})\right)$



$$\left((\{\mathtt{ac}\} \times \{\mathtt{b}\}) \cup (\{\mathtt{df}\} \times \{\mathtt{e}\})\right)^{-1}$$



Intersection

Recall that the regular languages—and thus FSAs—are also closed under intersection, implemented with an algorithm called *composition*. However, FSTs are not closed under intersection.

Composition

Composition is a generalization of intersection and relation chaining. Its precise interpretation depends on whether the inputs are languages/FSAs M, N or relations/FSTs μ, ν:

- $M \circ N$ yields their intersection $M \cap N$.
- $M \circ \nu$ yields $\{(a, b) \mid a \in M \wedge b \in \nu[a]\}$; i.e., it restricts the domain of ν by intersecting it with M.
- $\mu \circ N$ yields $\{(a, b) \mid b \in \mu[a] \wedge b \in N\}$; i.e., it restricts the range of μ by intersecting it with N.
- $\mu \circ \nu$ yields $\{(a, c) \mid b \in \mu[a] \wedge c \in \nu[b]\}$; i.e., it chains the output of μ to the input of ν.

Associativity

Composition is associative and *n*-ary composition can be implemented by a sequence of two-way compositions. Note however that for automata, one bracketing into a sequence of two-way compositions—e.g., $A \circ B \circ C$ factored as the *left-associative* $(A \circ B) \circ C$ versus the *right-associative* $A \circ (B \circ C)$—may be far more efficient than other equivalent associativities.

Rewrites

Why rewrites?

- Grammarians, since at least Pāṇini (fl. 4th c. BCE), have conceived of grammars not as sets of permissible strings but rather as a series of rules which "rewrite" abstract inputs to produce surface forms.
- One particularly influential rule notation is the one popularized by Chomsky and Halle (1968), henceforth SPE.
- Johnson (1972) proves this notation, with some sensible restrictions, is equivalent to the *rational relations* and thus to *finite transducers*.

Formalism

Let Σ be the set of symbols over which the rule will operate.

- For phonological rules, Σ might consist of all phonemes and their allophones in a given language.
- For grapheme-to-phoneme rules, it would contain both graphemes and phonemes.

Let $s, t, l, r \in \Sigma^*$. Then, the following is a possible rewrite rule.

$$s \longrightarrow t \ / \ l \_\_ r$$

where $s \longrightarrow t$ is the *structural change* and l and r as the *environment*. By convention, l and/or r can be omitted when they are null (i.e., are ϵ).

Interpretation

The above rule can be read as "*s* goes to *t* between *l* and *r*", and specifies a rational relation with domain and range Σ^* such that all instances of *lsr* are replaced with *ltr*, with all other strings in Σ^* passed through.

Example

Let $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ and consider the following rule.

$$\mathsf{b} \longrightarrow \mathsf{a} \ / \ \mathsf{b} \_\_ \mathsf{b}$$

$$\begin{array}{lcl} \mathsf{bbba} & \longrightarrow & \mathsf{baba} \\ \mathsf{abbbabbbc} & \longrightarrow & \mathsf{abababab c} \end{array}$$

Input: cbbca

Output: `cbbca`

Input: abbbba

Output: ???

Directionality

However, application is ambiguous with respect to certain input strings.

a.	*simultaneous* application	abaaba
b.	*left-to-right* or *right-linear* application	ababba
c.	*right-to-left* or *left-linear* application	abbaba

Directional application

In SPE it is assumed that that all rules apply simultaneously (op. cit., 343f.). However, Johnson (1972) adduces a number of phonological examples where directional application—either left-to-right or right-to-left—is required. However, note that directionality has no discernable effect on many rules and can often be ignored.

Boundary symbols

Let $\hat{}, \$ \notin \Sigma$ be *boundary symbols* disjoint from Σ. Now let $\hat{}$, the beginning-of-string symbol, to optionally appear as the leftmost symbol in *l*, and permit $, the end-of-string-symbol, to optionally appear as the rightmost symbol in *r*. These boundary symbols are not permitted to appear elsewhere in *l* or *r*, or anywhere within the structural description and change.

Example

Let $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ and consider the following rule.

$$\mathsf{b} \longrightarrow \mathsf{a} \;/\; \hat{}\; \mathsf{b} \_\_ \mathsf{b}$$

$$\begin{array}{lcl} \texttt{bbba} & \longrightarrow & \texttt{baba} \\ \texttt{abbbc} & \longrightarrow & \texttt{abbbc} \end{array}$$

Generalization

We can generalize the elements of rules from single strings to languages and relations. Then, a rewrite rule is specified by a five-tuple consisting of

- an *alphabet* Σ,
- a *structural change* $\tau \subseteq \Sigma^* \times \Sigma^*$,
- a *left environment* $L \subseteq \{\hat{}\}^? \Sigma^*$,
- a *right environment* $R \subseteq \Sigma^* \{\$\}^?$, and
- a *directionality* (one of: "simultaneous", "left-to-right", or "right-to-left").

Briefly noted:
Features
Abbreviatory devices
Constraint-based formalisms

Rule compilation

Rules which apply at the end or beginning of a string are generally trivial to express as a finite transducer. For example, the following rules prepend a prefix p or append a suffix s, respectively.

$$\begin{array}{ccl} \varnothing & \longrightarrow & \{p\} \;/\; \hat{} \;\_\_\; \Sigma^* \\ \varnothing & \longrightarrow & \{s\} \;/\; \Sigma^* \;\_\_\; \$ \end{array}$$

Such rules, respectively, correspond to the rational relations:

$$(\{\epsilon\} \times \{p\})\, \Sigma^*$$
$$\Sigma^*\, (\{\epsilon\} \times \{s\})$$

Challenges

Greater difficulties arise from the possibility of

- multiple sites for application and
- multiple overlapping contexts for application.

It thus proved challenging to develop a general-purpose algorithm for compilation, and was not widely-known until the 1990s (e.g., Kaplan and Kay, 1994; Karttunen, 1995). We review a generalization put forth by Mohri and Sproat (1996), which builds a rewrite rule from a cascade of five transducers, each a simple rational relation.

The algorithm I

If X is a language, let $\bar{X}$ denote its *complement*, the language consisting of all strings not in X. Then, let $<_1, <_2, > \notin \Sigma$ be *marker symbols* disjoint from the alphabet Σ. L and R are acceptors defining the left and right contexts, respectively. The constituent transducers are as follows:

- ρ inserts the $>$ marker before all substrings matching R: $\Sigma^* R \rightarrow \Sigma^* > R$.
- ϕ inserts markers $<_1$ and $<_2$ before all substrings matching $\pi_i(\tau) >$: $(\Sigma \cup \{>\})^* \pi_i(\tau) \rightarrow (\Sigma \cup \{>\})^* \{<_1, <_2\} \pi_i(\tau)$. Note that this introduces two paths, one with $<_1$ and one with $<_2$, which will ultimately correspond, respectively, to the cases where L does/does not occur to the left (see steps 4, 5 below).
- γ applies the structural change τ anywhere $\pi_i(\tau)$, the input projection of τ, is preceded by $<_1$ and followed by $>$. It simultaneously deletes the $>$ marker everywhere.

The algorithm II

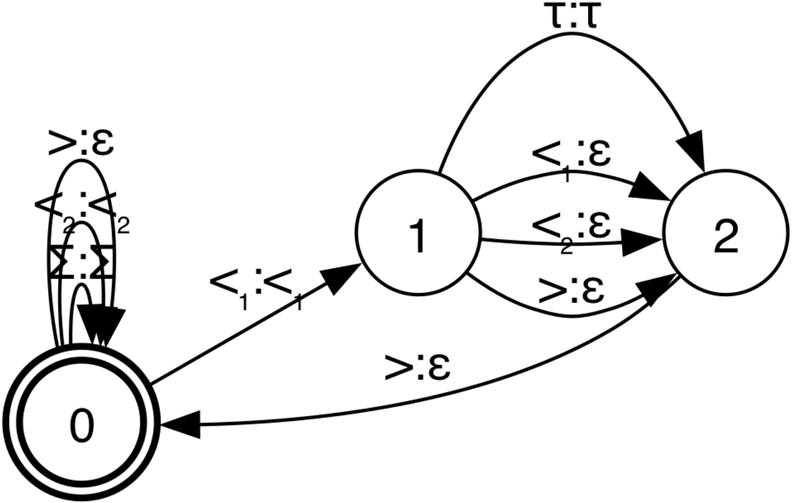
- λ_1 admits only those strings in which L is followed by the $<_1$ marker and deletes all $<_1$ markers satisfying this condition: $\Sigma^* L <_1 \rightarrow \Sigma^* L$.
- λ_2 admits only those strings in which all $<_2$ markers are not preceded by L and deletes all $<_2$ markers satisfying this condition: $\Sigma^* \overline{L} <_2 \rightarrow \Sigma^* \overline{L}$

Then, the final context-dependent rewrite rule transducer is given by

$$T = \rho \circ \phi \circ \gamma \circ \lambda_1 \circ \lambda_2$$

Slight variants are used for right-to-left and simultaneous transduction.

Schematic of γ



Briefly noted:
Efficiency considerations

Rule application

Rule application

To a first approximation, one can apply a rule ρ to a string i by compiling both, composing (*lattice construction*), and then extracting the output from the output projection $\pi_o(i \circ \rho)$ (*string extraction*). In practice, there are a few "gotchas" to watch out for.

Lattice construction

```
lattice = i @ rho
assert lattice.start() != pynini.NO_STATE_ID
lattice.project("output").rmepsilon()
```

Simple string extraction

There are several ways to extract a string from the lattice:

- If one expects only a single string:

  ```
  o = lattice.string()
  ```

- If one only wants the highest-weighted string:

  ```
  o = pynini.shortestpath(lattice).string()
  ```

Advanced string extraction

- If one wants the *n*-highest-weighted strings:

```
lattice = pynini.shortestpath(lattice,
                              nshortest=n,
                              unique=True)
for o in lattice.paths().ostrings():
    ...
```

- If one wants all the top-weighted strings:

```
lattice = pynini.determinize(lattice,
                             weight=0)
for o in lattice.paths().ostrings():
    ...
```

OpenFst and friends

OpenFst (Allauzen et al., 2007)

OpenFst is a open-source C++17 library for weighted finite state transducers developed at Google. Among other things, it is used in:

- automatic speech(-to-text) recognizers (e.g., Kaldi and many commercial products).
- text-to-speech synthesizers (as part of the “front-end”).
- input method engines (e.g., mobile text entry systems).
- many other kinds of text hacking.

Features

- One serialization format (`.fst`) is shared across all OpenFst and OpenGrm libraries.
- FSTs can be compacted; e.g., unweighted string acceptors can be stored as integer arrays.
- Collections of FSTs can be stored in FST archives (`.far`), a shardable key-value store.

OpenFst design

There are (at least) four layers to OpenFst:

- a C++ template/header library in `<fst/*.h>`
- a C++ “scripting” library in `<fst/script/*.{h,cc}>`
- CLI programs in `/usr/local/bin/*`
- a Python extension module `pywrapfst`

OpenGrm

- Baum-Welch (Gorman and Allauzen, 2024): CLI tools and libraries for performing expectation maximization on WFSTs
- NGram (Roark et al., 2012): CLI tools and libraries for building conventional n-gram language models
- Pynini (Gorman, 2016; Gorman and Sproat, 2021): Python extension module for WFST grammar development
- SFst (Allauzen and Riley, 2018): CLI tools and libraries for building *stochastic FSTs*
- Thrax (Roark et al., 2012): DSL-based compiler for WFST grammar development

WFSTs: the later years

We are now a decade into what has been called "deep learning tsunami" (Manning, 2015). Yet weighted finite-state transducers continue to play a crucial role in industrial speech and language technologies.

A battle between two great powers?

- knowledge-based vs. data-driven
- rationalism vs. empiricism
- neats vs. scruffies
- cowboys vs. aliens

Text normalization

Many speech and language technologies map between “written” and “spoken” representations of language. *Text normalization* (Sproat et al., 2001) refers to mappings between pseudo-ideographic representations like **$4.20** to more pronounceable representations like *four dollars and twenty cents*.

Semiotic categories (Ebden and Sproat, 2014)

- Cardinal: **69** $\longrightarrow$ *sixty nine*
- Date: **11/2/1985** $\longrightarrow$ *November second nineteen eighty five*
- Decimal: **23.3** $\longrightarrow$ *twenty three point three*
- Electronic: **kgorman@gc.cuny.edu** $\longrightarrow$ *k gorman at gc dot cuny dot edu*
- Fraction: **2/5** $\longrightarrow$ *two fifths*
- Measure: **12kg** $\longrightarrow$ *twelve kilograms*
- Money: **$5.96** $\longrightarrow$ *five dollars and ninety six cents*
- Ordinal: **69th** $\longrightarrow$ *sixty ninth*
- Roman numeral: **LIV** $\longrightarrow$ *fifty four*
- Telephone: **566-6123** $\longrightarrow$ *five six six, six one two three*
- Time: **11:58** $\longrightarrow$ *eleven fifty eight*

Wikipedia (“written” domain)

The giraffe has an extremely elongated neck, which can be up to **2 m** (**6 ft 7 in**) in length, accounting for much of the animal’s vertical height. Each cervical vertebra is over **28 cm** (**11 in**) long. They comprise **52-54 percent** of the length of the giraffe’s vertebral column, compared with the **27–33 percent** typical of similar large ungulates, including the giraffe’s closest living relative, the okapi.

Wikipedia ("spoken" domain)

The giraffe has an extremely elongated neck, which can be up to *two meters* (*six feet seven inches*) in length, accounting for much of the animal's vertical height. Each cervical vertebra is over *twenty eight centimeters* (*eleven inches*) long. They comprise *fifty two to fifty four percent* of the length of the giraffe's vertebral column, compared with the *twenty seven to thirty three percent* typical of similar large ungulates, including the giraffe's closest living relative, the okapi.

Applications

- In text-to-speech synthesis, the front-end is responsible for providing pronunciations for semiotic classes.
- In automatic speech recognition:
    - the written text used to train language models are converted to spoken form.
    - spoken form transcriptions from the recognizer are converted back to written form (e.g., Shugrina, 2010; Pusateri et al., 2017).
- In information extraction, verbalizations can be used as a canonical form for spoken and the various written forms of dates, times, etc.

Machine learning for text normalization at Google

- Sentence boundary detection (Sproat and Hall, 2014)
- English abbreviation expansion (Roark and Sproat, 2014; Gorman et al., 2021)
- Grapheme-to-phoneme prediction (Jansche, 2014; Rao et al., 2015)
- Russian word stress prediction (Hall and Sproat, 2013)
- Number name generation (Gorman and Sproat, 2016; Ritchie et al., 2019)
- Letter sequence prediction (Sproat and Hall, 2014)
- Homograph disambiguation (Gorman et al., 2018)
- End-to-end research (Ng et al., 2017; Sproat and Jaitly, 2017; Zhang et al., 2019)

But...

- Yet nearly all text normalization is still done with hand-written language-specific grammars, just like 25 years ago (e.g., Sproat, 1996), not with sequence-to-sequence neural networks.
- The required native speaker-cum-computational-linguistic sophistication needed to develop and maintain these grammars is thin on the ground and this is **the** major barrier to internationalization in speech technology.

Some new(ish) WFST algorithms

- general-purpose WFST optimization
- A* shortest string decoding over non-idempotent semirings

General-purpose WFST optimization

Optimal for what?

There are many ways a WFST might be said to be optimal. For instance, a WFST could be optimal for:

- composition efficiency (i.e., by eliminating internal ϵ-labels or moving them later along paths).
- footprint in memory (i.e., by reducing the number of states and arcs).
- cache utilization or other application-specific use patterns.

Minimality

- An automaton is *minimal* if it expresses its (weighted) language or relation using the minimal number of states.
- Efficient algorithms exist for minimizing deterministic automata (e.g., Mohri, 2000).
- However, finding an equivalent deterministic automaton for an arbitrary WFST can be computationally expensive if not impossible.

Implementation

- Pynini `Fst` objects have a destructive instance method `optimize`.
- Thrax has a function `Optimize`.

Both share the same C++ template implementation in `optimize.h`.

Preprocessing

We first apply ϵ-removal (Mohri, 2002a) if the input WFST is not known to be ϵ-free.

Optimizing acceptors

Not all acceptors are determinizable.

- Acceptors which do not have weights other than $\bar{0}$ and/or $\bar{1}$ along their cycles—as well as acylic and unweighted acceptors—are determinizable over a wide variety of semirings (Mohri, 2009). We then apply determinization and minimization if the acceptor is not known to be deterministic.
- However, it is difficult to determine whether determinization will even terminate for cyclic weighted non-deterministic acceptors (Allauzen and Mohri, 2003). Therefore, we heuristically apply determinization and minimization to such acceptors *viewed as unweighted*. This is guaranteed to halt.

Optimizing transducers

Similarly, not all transducers are determinizable.

- Even transducers without weighted cycles may be non-functional. Therefore, we heuristically apply determinization and minimization to such transducers *viewed as acceptors*. This is guaranteed to halt.
- For cyclic weighted transducers, we heuristically apply determinization and minimization to such transducers *viewed as unweighted acceptors*. This is also guaranteed to halt.

Postprocessing

When an weighted cyclic automaton is heuristically optimized as if it was unweighted, we also apply arc-sum mapping as a post-processing step. This eliminates trivial (i.e., same-state) cases of non-determinism due to identically labeled arcs with different weights leaving the same state, which may be introduced during heuristic determinization-minimization.

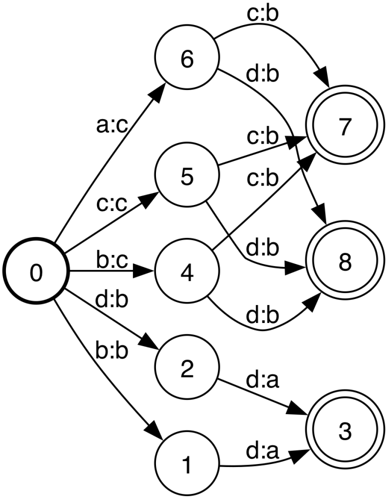


Figure: Finite transducer before optimization.

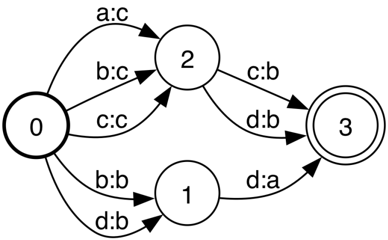


Figure: Equivalent finite transducer after optimization.

Evaluation

- We apply the above algorithm to a sample of 700 speech recognition word lattices derived from Google Voice Search traffic, lattices previously used Mohri and Riley (2015) to evaluate related algorithms.
- Each lattice path represents a single hypothesis transcription from a production-grade automatic speech recognizer.
- These lattices are acyclic and ϵ-free, non-deterministic, and weighted, and thus the algorithm above is guaranteed to produce a deterministic, minimal, ϵ-free acceptor.

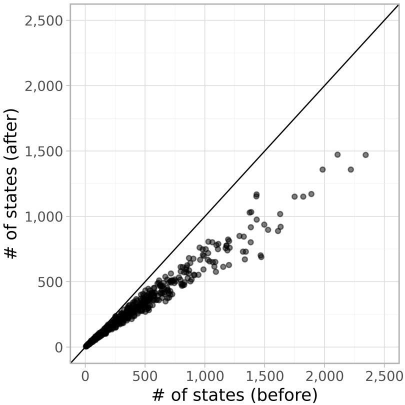


Figure: Word lattice optimization with the proposed algorithm. The *x*-axis shows the number of states before optimization; the *y*-axis shows the number of states after optimization.

Results

- Optimization substantially reduces the number of states, particularly for the larger lattices.
- The post-optimization “after” automaton is never larger than the “before” automaton.

Related work

An earlier version of this algorithm was proposed by Allauzen et al. (2004). The above evaluation is reported by Gorman and Sproat (2021, §4.5).

A* shortest string decoding for non-idempotent semirings

Motivations

Various circumstances force us to build WFST models we cannot decode efficiently or exactly due to restrictions on shortest-path algorithms. We attempt to remedy these restrictions.

Three types of expectation maximization

- In *vanilla EM* (Dempster et al., 1977), we collect counts in semirings isomorphic to Plus-Times.
- In *Viterbi EM* (Brown et al., 1993, 293), we collect counts in semirings isomorphic to Max-Times.
- In *lateen EM* (Spitkovsky et al., 2011), we alternate between vanilla and Viterbi EM according to some training schedule.

Yet there is no way to compute the shortest path in semirings isomorphic to Plus-Times.

Preliminaries

Without loss of generality, we consider single-source ϵ-free acyclic acceptors, using $z[p] = x[p] = y[p]$ to denote the string of a path p.

Shortest distance

Let $P_{q \to r}$ be the set of all paths from q to r where $q, r \in Q$. Then:

- the *forward shortest distance* $\alpha \subseteq Q \times \mathbb{K}$ maps from a state $q \in Q$ to the $\oplus$-sum of the $\otimes$-product of the weights of all paths from s to q:

$$\alpha(q) = \bigoplus_{p \in P_{s \to q}} \bigotimes_{k_i \in k[p]} k_i.$$

- the *backwards shortest distance* $\beta \subseteq Q \times \mathbb{K}$ maps from a state $q \in Q$ to the $\oplus$-sum of $\otimes$-product of the weights of all paths from q to any final state:

$$\beta(q) = \bigoplus_{f \in F} \left(\bigoplus_{p \in P_{q \to f}} \bigotimes_{k_i \in k[p]} k_i \otimes \omega(f) \right).$$

Shortest path

- The *total shortest distance* through an automaton is given by $\beta(s)$.
- The *shortest path* through an automaton is a complete path whose weight is equal to $\beta(s)$.

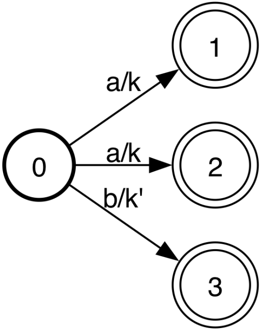


Figure: Automata over non-idempotent semirings need not have a shortest path. Consider the figure above. If $k \oplus k \preceq k < k'$, then the total shortest distance is $k \oplus k$, which need not correspond to any one path.

Shortest string

Let P_z be a set of paths with string $z \in \Sigma^*$, and let the weight of P_z be

$$\sigma(z) = \bigoplus_{p \in P_z} \bar{k}[p].$$

Then a *shortest string z* is one such that $\forall z' \in \Sigma^*, \sigma(z) \preceq \sigma(z')$.

Lemma I

Lemma

In an idempotent semiring, a shortest path's string is also a shortest string.

Proof

Let p be a shortest path. By definition, $\bar{k}[p] \preceq \bar{k}[p']$ for all complete paths p'. It follows that

$$\forall z' \in \Sigma^* : \sigma(z[p]) = \bigoplus_{p \in P_z} \bar{k}[p] \preceq \sigma(z'[p']) = \bigoplus_{p' \in P_z} \bar{k}[p']$$

so $z[p]$ is the shortest string.

Companion semirings

The *companion semiring* of a monotonic negative semiring $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ with a total order $\preceq$ is the semiring $(\mathbb{K}, \widehat{\oplus}, \otimes, \bar{0}, \bar{1})$ where $\widehat{\oplus}$ is the minimum binary operator for $\preceq$:

$$a \mathbin{\widehat{\oplus}} b = \begin{cases} a & \text{if } a \preceq b \\ b & \text{otherwise} \end{cases}$$

For example, the tropical semiring

$$(\mathbb{R} \cup \{-\infty, +\infty\}, \min, +, +\infty, 0)$$

is the companion semiring for the log semiring

$$(\mathbb{R} \cup \{-\infty, +\infty\}, \oplus_{\log}, +, +\infty, 0).$$

Lemma II

Lemma

In a DFA over a monotonic semiring, a shortest string is the string of a shortest path in that DFA viewed over the corresponding companion semiring.

Proof

Determinism implies that for all complete path p', $\bar{k}[p'] = \sigma(z[p'])$. Let z be the shortest string in the DFA and p the unique path admitting the string z. Then

$$\bar{k}[p] = \sigma(z) \preceq \sigma(z[p']) = \bar{k}[p']$$

for any complete path p'. Hence

$$\bar{k}[p] = \widehat{\bigoplus_{p' \in P_{s \to F}}} \bar{k}[p'].$$

Thus p is a shortest path in the DFA viewed over the companion semiring.

Shortest-first search

Dijkstra's (1959) algorithm is an example of a *shortest-first* search strategy appropriate for idempotent semirings. At every iteration, the algorithm explores the state q which minimizes $\alpha(q)$, the shortest distance from the initial state s to q, until all states have been visited.

A* search

In the variant known as A* search (Hart et al., 1968), search priority is instead a function of $F \subseteq Q \times \mathbb{K}$, known as the *heuristic*, which gives an estimate of the weight of paths from some state to a final state. At every iteration, A* instead explores the state q which minimizes $\alpha(q) \otimes F(q)$.

Dijkstra again

Then, Dijkstra's algorithm is just a special case of A* search using $F = \bar{1}$.

Heuristics

A heuristic is:

- *admissible* if it never overestimates the shortest distance to a state. That is, it is admissible if $\forall q \in Q : F(q) \preceq \beta(q)$.
- *consistent* if it never overestimates the cost of reaching a successor state. That is, it is consistent if $\forall q, r \in Q$ such that $F(q) \preceq k \otimes F(r)$ if $(q, z, k, r) \in \delta$, i.e., if there is a transition from q to r with some label z and weight k.

If F is *admissible* and *consistent*, A* search is guaranteed to find a shortest path (if one exists) after visiting all states such that $F(q) \preceq \beta(s)$ (Hart et al., 1968, 104f.).

Preliminaries

Consider an acyclic, ϵ-free WFSA over a monotonic negative semiring $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ with total order $\preceq$ for which we wish to find the shortest string. The same WFSA can also be viewed as a WFSA over the corresponding companion semiring $(\mathbb{K}, \widehat{\oplus}, \otimes, \bar{0}, \bar{1})$, and we denote by $\widehat{\beta}$ the backward shortest-distance over this companion semiring.

Proof I

Theorem
The backwards shortest distance of an WFSA over a monotonic negative semiring is an admissible heuristic for the A* search over its companion semiring.

Proof
In a monotonic negative semiring, the $\oplus$-sum of any n terms is upper-bounded by each of the n terms and hence by the $\widehat{\oplus}$-sum of these n terms. It follows that

$$\beta(q) = \bigoplus_{p \in P_{q \to F}} \bar{k}[p] \preceq \widehat{\bigoplus_{p \in P_{q \to F}}} \bar{k}[p] = \widehat{\beta}(q)$$

showing that $F = \beta$ is an admissible heuristic for $\widehat{\beta}$.

Proof II

Theorem

The backwards shortest distance of an WFSA over a monotonic negative semiring is a consistent heuristic for the A* search over its companion semiring.

Proof

We again use the property that an $\oplus$-sum of any n terms is upper-bounded by any of these terms. If (q, z, k, r) be a transition in δ

$$\beta(q) = \bigoplus_{p \in P_{q \to F}} \bar{k}[p] = \bigoplus_{(q,z',k',r') \in \delta} k' \otimes \beta(r') \preceq k \otimes \beta(r)$$

showing that $F = \beta$ is a consistent heuristic.

Naïve algorithm

A naïve algorithm suggests itself. Given a non-deterministic WFSA over the monotonic negative semiring $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$:

- apply determinization to obtain an equivalent DFA.
- compute β_d, the DFA's backwards shortest distance.
- perform A* search using β_d as the heuristic.

Exponential blowup

Determinization has an exponential worse-case complexity in time and space and is often prohibitive in practice. Yet determinization—and the computation of elements of β_d—only need to be performed for states **actually visited** during search.

Our algorithm

Let β_n denote backwards shortest distance over a non-deterministic WFSA over the monotonic negative semiring $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$. Then:

- compute β_n over $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$.
- lazily determinize the WFSA (Mohri, 1997), lazily computing β_d from β_n over $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$
- perform A* search using β_d as the heuristic over the companion semiring $(\mathbb{K}, \widehat{\oplus}, \otimes, \bar{0}, \bar{1})$.

Evaluation

We search for the shortest string over a sample of 700, acyclic, ϵ-free non-deterministic WFSA word lattices derived from Google Voice Search traffic. For this, we use the OpenGrm-BaumWelch command-line tool `baumwelchdecode` to implement the above algorithm over the log semiring, with the tropical semiring as the companion semiring.

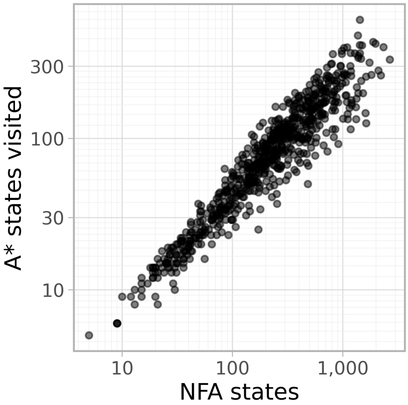


Figure: Word lattice decoding with the proposed algorithm. The *x*-axis shows the number of states in each word lattice NFA; the *y*-axis shows the number of states visited by A* decoding. Both axes are log scale.

Results

- The relationship between the size of the NFA and the number of DFA states visited by the proposed decoding method appears roughly monomial (i.e., log-log linear).
- The size of the full DFA was measured by applying the OpenFst command-line tool `fstdeterminize` to the lattices, which produces an approximately 7x increase in the size of the lattices.
- From this we infer that the proposed heuristic substantially reduces the number of DFA states that are visited.

Applications

Single shortest string over non-idempotent semirings can be used for exact decoding of:

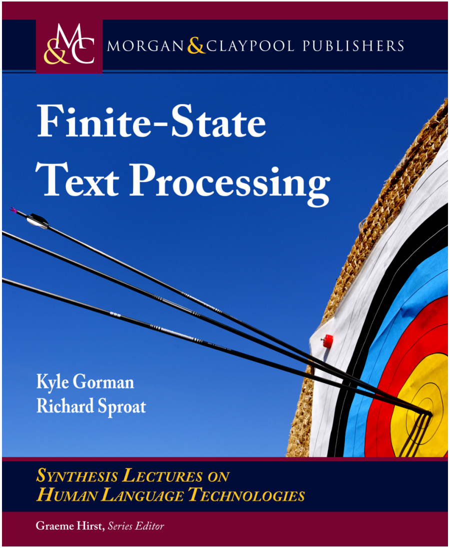
- interpolated (e.g., Jelinek et al., 1983) language models of the form

$$\hat{P}(w \mid h) = \lambda_h \tilde{P}(w \mid h) + (1 - \lambda_h)\hat{P}(w \mid h').$$

- "decipherment" models (e.g., Knight et al., 2006) of the form

$$\hat{P}(p \mid c) \propto P(p)P(c \mid p)$$

trained with classic expectation maximization.



http://wellformedness.com/courses/fstp/

More information

openfst.org

opengrm.org

baumwelch.opengrm.org

ngram.opengrm.org

pynini.opengrm.org

thrax.opengrm.org

sfst.opengrm.org

Further reading

- Gorman and Sproat, 2021: introduces WFST text processing in Python
- Mohri, 2009: reviews major WFST algorithms
- Mohri, 2002b: discusses shortest-distance and shortest-path algorithms

References I

C. Allauzen and M. Mohri. Efficient algorithms for testing the twins property. *Journal of Automata, Languages and Combinatorics*, 8(2): 117–144, 2003.

C. Allauzen and M. Riley. Algorithms for weighted finite automata with failure transitions. In *Proceedings of the 23rd International Conference on Implementation and Application of Automata*, pages 46–58, 2018.

C. Allauzen, M. Mohri, M. Riley, and B. Roark. A generalized construction of integrated speech recognition transducers. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 761–764, 2004.

C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri. OpenFst: a general and efficient weighted finite-state transducer library. In *Proceedings of the 12th International Conference on Implementation and Application of Automata*, pages 11–23, 2007.

References II

P. Brown, S. A. Della Pietra, V. J. Della Pietra, and R. L. Mercer. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993.

N. Chomsky and M. Halle. *Sound Pattern of English*. Harper & Row, 1968.

N. Chomsky and G. A. Miller. Introduction to the formal analysis of natural languages. In R. D. Luce, R. R. Bush, and E. Galanter, editors, *Handbook of Mathematical Psychology*, pages 269–321. Wiley, 1963.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

References III

P. Ebden and R. Sproat. The Kestrel TTS text normalization system. *Natural Language Engineering*, 21:1–21, 2014.

E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

K. Gorman. Pynini: a Python library for weighted finite-state grammar compilation. In *ACL Workshop on Statistical NLP and Weighted Automata*, pages 75–80, 2016.

K. Gorman and C. Allauzen. A* shortest string decoding for non-idempotent semirings. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics*, 2024.

K. Gorman and R. Sproat. Minimally supervised models for number normalization. *Transactions of the Association for Computational Linguistics*, 4:507–519, 2016.

References IV

K. Gorman and R. Sproat. *Finite-State Text Processing*. Morgan & Claypool, 2021.

K. Gorman, G. Mazovetskiy, and V. Nikolaev. Improving homograph disambiguation with supervised machine learning. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation*, pages 1349–1352, 2018.

K. Gorman, C. Kirov, B. Roark, and R. Sproat. Structured abbreviation expansion in context. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 995–1005, 2021.

K. Hall and R. Sproat. Russian stress prediction using maximum entropy ranking. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 879–883, 2013.

References V

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimal cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

M. Jansche. Computer-aided quality assurance of an Icelandic pronunciation dictionary. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation*, pages 2111–2114, 2014.

F. Jelinek, L. R. Bahl, and R. L. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 5:179–190, 1983.

C. D. Johnson. *Formal aspects of phonological description*. Mouton, 1972.

R. Kaplan and M. Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.

References VI

L. Karttunen. The replace operator. In *33rd Annual Meeting of the Association for Computational Linguistics*, pages 16–23, 1995.

S. C. Kleene. Representations of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–42. Princeton University Press, 1956.

K. Knight, A. Nair, N. Rashod, and K. Yamada. Unsupervised analysis for decipherment problems. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 499–506, 2006.

C. D. Manning. Last words: computational linguistics and deep learning. *Computational Linguistics*, 41(4):701–707, 2015.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

References VII

M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

M. Mohri. Minimization algorithms for sequential transducers. *Journal of Automata, Languages and Combinatorics*, 234(1–2):177–201, 2000.

M. Mohri. Generic epsilon-removal and input epsilon-normalization algorithms for weighted transducers. *International Journal of Computer Science*, 13(1):129–143, 2002a.

M. Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3): 321–350, 2002b.

M. Mohri. Weighted automata algorithms. In M. Droste, W. Kuich, and H. Vogler, editors, *Handbook of weighted automata*, pages 213–254. Springer, 2009.

References VIII

M. Mohri and M. D. Riley. On the disambiguation of weighted automata. In *Proceedings of the 20th International Conference on Implementation and Application of Automata*, pages 263–278, 2015.

M. Mohri and R. Sproat. An efficient compiler for weighted rewrite rules. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 231–238, 1996.

A. H. Ng, K. Gorman, and R. Sproat. Minimally supervised written-to-spoken text normalization. In *ASRU*, pages 665–670, 2017.

E. Pusateri, B. R. Ambati, E. Brooks, O. Platek, D. McAllaster, and V. Nagesha. A mostly data-driven approach to inverse text normalization. In *Proceedings of INTERSPEECH*, pages 2784–2788, 2017.

K. Rao, F. Peng, H. Sak, and F. Beaufays. Grapheme-to-phoneme conversion using long short-term memory recurrent neural networks. In *ICASSP*, pages 4225–4229, 2015.

References IX

S. Ritchie, R. Sproat, K. Gorman, D. van Esch, C. Schallhart, N. Bampounis, B. Brard, J. F. Mortensen, M. Holt, and E. Mahon. Unified verbalization for speech recognition & synthesis across languages. In *Proceedings of INTERSPEECH*, pages 3530–3534, 2019.

B. Roark and R. Sproat. Hippocratic abbreviation expansion. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 364–369, 2014.

B. Roark, R. Sproat, C. Allauzen, M. Riley, J. Sorensen, and T. Tai. The OpenGrm open-source finite-state grammar software libraries. In *Proceedings of the ACL 2012 System Demonstrations*, pages 61–66, 2012.

References X

M. Shugrina. Formatting time-aligned ASR transcriptions for readability. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 198–206, 2010.

V. I. Spitkovsky, H. Alshawi, and D. Jurafsky. Lateen EM: unsupervised training with multiple objectives, applied to dependency grammar induction. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1269–1280, 2011.

R. Sproat. Multilingual text analysis for text-to-speech synthesis. *Natural Language Engineering*, 2(4):369–380, 1996.

R. Sproat and K. Hall. Applications of maximum entropy rankers to problems in spoken language processing. In *INTERSPEECH*, pages 761–764, 2014.

References XI

R. Sproat and N. Jaitly. An RNN model of text normalization. In *INTERSPEECH*, pages 754–758, 2017.

R. Sproat, A. W. Black, S. Chen, S. Kumar, M. Ostendorf, and C. Richards. Normalization of non-standard words. *Computer Speech & Language*, 15:287–333, 2001.

K. Thompson. Programming techniques: regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

K. Vijay-Shanker, D. J. Weir, and A. K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In *25th Annual Meeting of the Association for Computational Linguistics*, pages 104–111, 1987.

H. Zhang, R. Sproat, A. H. Ng, F. Stahlberg, X. Peng, K. Gorman, and B. Roark. Neural models of text normalization for speech applications. *Computational Linguistics*, 45(2):293–337, 2019.

Backup slides