Finite-state transducers

LING83800

Outline

- Rational relations
- Finite-state transducers
- Composition
- Rewrites
- Demo

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.

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

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

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

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.

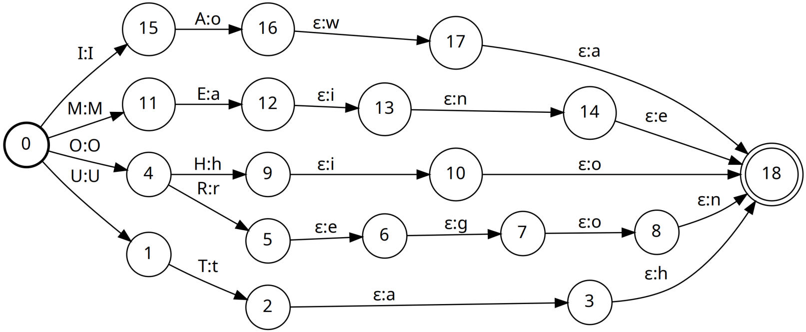
FSAs as FSTs

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.

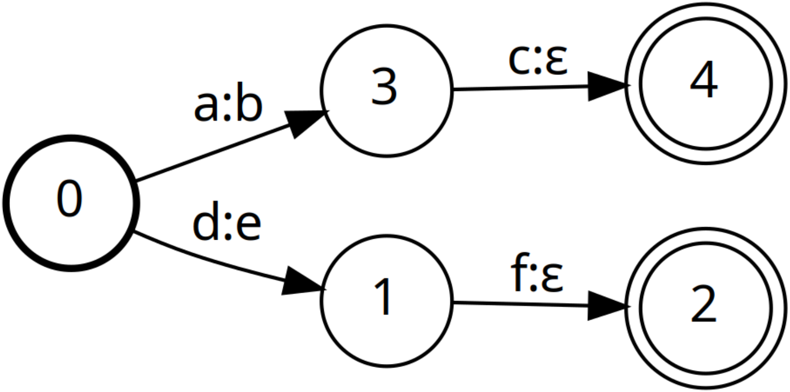
Projection

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.

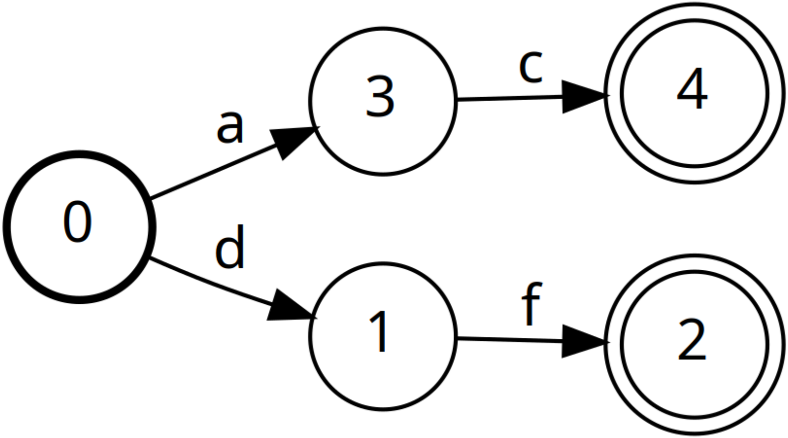
Inversion

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.

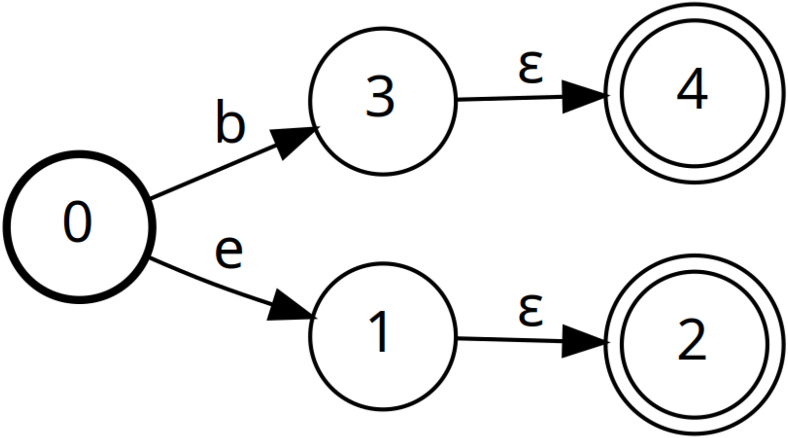
$(\{ac\} \times \{b\}) \cup (\{df\} \times \{e\})$



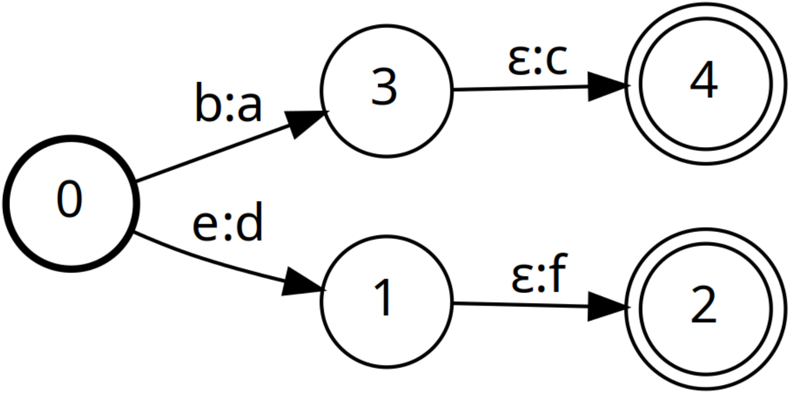
$\pi_i\left((\{\mathtt{ac}\} \times \{\mathtt{b}\}) \cup (\{\mathtt{df}\} \times \{\mathtt{e}\})\right)$



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



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



Intersection

Recall from last week's lecture 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.

Briefly noted:
Associativity
Implementational details

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

$$\text{b} \rightarrow \text{a} \ / \ \text{b} \_\_ \text{b}$$

$$\begin{array}{lcl} \text{bbba} & \rightarrow & \text{baba} \\ \text{abbbabbbc} & \rightarrow & \text{ababababc} \end{array}$$

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

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

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

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.

$$\begin{aligned} \varnothing &\rightarrow \{p\} \;/\; \hat{}\;\_\_\;\Sigma^* \\ \varnothing &\rightarrow \{s\} \;/\; \Sigma^*\;\_\_\;\$ \end{aligned}$$

Such rules, respectively, correspond to the rational relations:

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

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

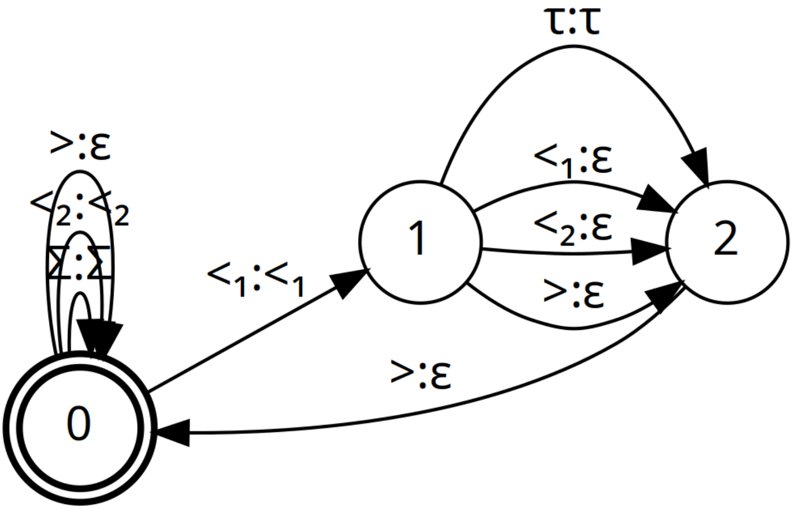
- λ_1 admits only those strings in which L is followed by the $<_1$ marker and deletes all $<_1$ markers satisfying this condition: $\Sigma^* L <_1 \rightarrow \Sigma^* L$.
- λ_2 admits only those strings in which all $<_2$ markers are not preceded by L and deletes all $<_2$ markers satisfying this condition: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

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

Demo

References I

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

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.

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

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.

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