Finite-state transducers

- a:0 is a notation for a mapping between two alphabets a ∈ Σ₁ and 0 ∈ Σ₂
- Finite-state transducers (FSTs) accept pairs of strings
- Finite-state automata equate to regular languages and FSTs equate to regular relations
- e.g. L = { (xⁿ, yⁿ) | n > 0, x ∈ Σ₁ and y ∈ Σ₂} is a regular relation accepted by some FST. It maps a string of x’s into an equal length string of y’s
Regular relations

- A generalization of regular languages
- The set of regular relations is:
  - The empty set and \((x, y)\) for all \(x, y\) is a regular relation
  - If \(R_1, R_2\) and \(R\) are regular relations then:
    - \(R_1 \cdot R_2 = \{(x_1, x_2, y_1, y_2) \mid (x_1, y_1) \in R_1, (x_2, y_2) \in R_2\}\)
    - \(R_1 \cup R_2\)
    - \(R^* = \cup_{i=0}^\infty R_i\)
  - There are no other regular relations

Finite-state transducers

- Formal definition:
  - \(Q\): finite set of states, \(q_0, q_1, \ldots, q_n\)
  - \(\Sigma\): alphabet composed of input/output pairs \(i:o\)
    where \(i \in \Sigma_1\) and \(o \in \Sigma_2\) and so \(\Sigma \subseteq \Sigma_1 \times \Sigma_2\)
  - \(q_0\): start state
  - \(F\): set of final states
  - \(\delta(q, i:o)\) is the transition function which returns a set of states

Finite-state transducers: Examples

- \((a^n, b^n)\): map \(n\) \(a\)'s into \(n\) \(b\)'s
- rot13 encryption (the Caesar cipher): assuming 26 letters each letter is mapped to the letter 13 steps ahead (mod 26), e.g. \(cipher \rightarrow pvcur\)
- reversal of a fixed set of words
- reversal of all strings upto fixed length \(k\)
- input: binary number \(n\), and output: binary number \(n+1\)
- upcase or lowercase a string of any length
- *Pig latin: \(pig\) \(latin\) \(is\) \(goofy\) \(\rightarrow igpay\) \(atinlay\) \(is\) \(oofy\)\(gay\)
- *convert numbers into pronunciations,
  e.g. 230.34 two hundred and thirty point three four
Regular Relations Closure Properties

- New operations for regular relations:
  - *composition*
  - *project* input (or output) language to regular language;
    for FST $t$, input language = $\pi_i(t)$, output = $\pi_o(t)$
  - take a regular language and create the *identity* regular relation;
    for FSM $f$, let FST for identity relation be $\text{Id}(f)$
  - take two regular languages and create the *cross product*
    relation; for FSMs $f$ & $g$, FST for cross product is $f \times g$
  - take two regular languages, and *mark each time the first language
    matches any string in the second language*

Regular Relation/FST Kleene Closure

- Regular relations (rr) are *closed* under some operations
- For example, if $R_1$, $R_2$ are regular relns:
  - union ($R_1 \cup R_2$ results in $R_3$ which is a rr)
  - concatenation
  - iteration ($R_1^+ = \text{one or more repeats of } R_1$)
  - Kleene closure ($R_1^* = \text{zero or more repeats of } R_1$)
- However, unlike regular languages, regular relns are not closed under:
  - intersection (possible for equal length regular relns)
  - complement

Finite-state transducers

- Following relations are cannot be expressed as a FST
  - $(a^n b^n, c^n)$: because $a^n b^n$ is not regular
  - reversal of strings of any length
  - $a^i b^j \rightarrow b^j a^i$ for any $i, j$
- Unlike regular languages, regular relations are not closed under intersection
  - $(a^n b^n, c^n) \cap (a^n b^n, c^n)$ produces $(a^n b^n, c^n)$
  - However, regular relations with input and output of equal lengths *are* closed under intersection
Regular Expressions for FSTs

(a:c) (b:d)*

( a:c (b:d)* ) ∪ ( (e:g)* f:h )

( (a:0 ∪ a:1) (b:0 ∪ b:1) )*
The basic idea is similar to the closure of regular languages under union or intersection.

**But**, instead of cross-product of *states*, we consider cross-product of *edges*: compose those edges where output/input matches.
Composing FSTs

\[
\begin{align*}
(0,0) &\rightarrow (1,1) \text{ a:a} \\
(0,1) &\rightarrow (2,1) \text{ b:a} \\
(0,1) &\rightarrow (1,2) \text{ a:a} \\
(1,1) &\rightarrow (3,1) \text{ b:d} \\
(0,1) &\rightarrow (0,2) \text{ a:c} \\
(1,1) &\rightarrow (3,2) \text{ b:c} \\
(2,0) &\rightarrow (3,1) \text{ b:a} \\
(2,1) &\rightarrow (2,2) \text{ b:a} \\
(2,1) &\rightarrow (3,2) \text{ b:a} \\
\end{align*}
\]

Composing FSTs

\[
\begin{align*}
(0,0) &\rightarrow (1,1) \text{ a:a} \\
(0,1) &\rightarrow (2,1) \text{ b:a} \\
(0,1) &\rightarrow (1,2) \text{ a:a} \\
(1,1) &\rightarrow (3,1) \text{ b:d} \\
(0,1) &\rightarrow (0,2) \text{ a:c} \\
(1,1) &\rightarrow (3,2) \text{ b:c} \\
(2,0) &\rightarrow (3,1) \text{ b:a} \\
(2,1) &\rightarrow (2,2) \text{ b:a} \\
(2,1) &\rightarrow (3,2) \text{ b:a} \\
\end{align*}
\]
FST Composition

- Input: transducer S and T
- Transducer composition results in a new transducer with states and transitions defined by matching compatible input-output pairs.
- \( \text{match}(s, t) \) : defines the edges for the new composed FST, \( s \) is a state in S, \( t \) is a state in T

Cross-product FST

- For regular languages \( L_1 \) and \( L_2 \), we have two FSAs, \( M_1 \) and \( M_2 \)
  
  \[
  M_1 = (\Sigma_1, Q_1, q_1, F_1, \delta_1) \\
  M_2 = (\Sigma_2, Q_2, q_2, F_2, \delta_2) 
  \]
- Then a transducer accepting \( L_1 \times L_2 \) is defined as:
  
  \[
  T' = (\Sigma_1, \Sigma_2, Q_1 \times Q_2, \langle q_1, q_2 \rangle, F_1 \times F_2, \delta) \\
  \delta(s_1, s_2, a, b) = \delta_1(s_1, a) \times \delta_2(s_2, b) \\
  \text{for any } s_1 \in Q_1, s_2 \in Q_2 \text{ and } a, b \in \Sigma \cup \{\varepsilon\}
  \]

FST Composition

- \( \text{match}(s, t) = \)
  
  \[
  \begin{align*}
  \{ (s, t) \rightarrow^{x:y} (s', t') \mid s \rightarrow^{x:y} s' \in S.\text{edges} \text{ and} \\
  t \rightarrow^{y:z} t' \in T.\text{edges} \} \cup \\
  \{ (s, t) \rightarrow^{x:e} (s', t) \mid s \rightarrow^{x:e} s' \in S.\text{edges} \} \cup \\
  \{ (s, t) \rightarrow^{e:z} (s, t') \mid t \rightarrow^{e:z} t' \in T.\text{edges} \}
  \end{align*}
  \]
- **Correctness**: any path in composed transducer mapping \( u \) to \( w \) arises from a path mapping \( u \) to \( v \) in S and path mapping \( v \) to \( w \) in T, for some \( v \)

Summary

- Finite state transducers specify regular relations
  - Encoding computation as finite-state transducers
- Extension of regular expressions to the case of regular relations/FSTs
- FST closure properties: union, concatenation, composition
- FST special operations:
  - creating regular relations from regular languages (Id, cross-product);
  - creating regular languages from regular relations (projection)