

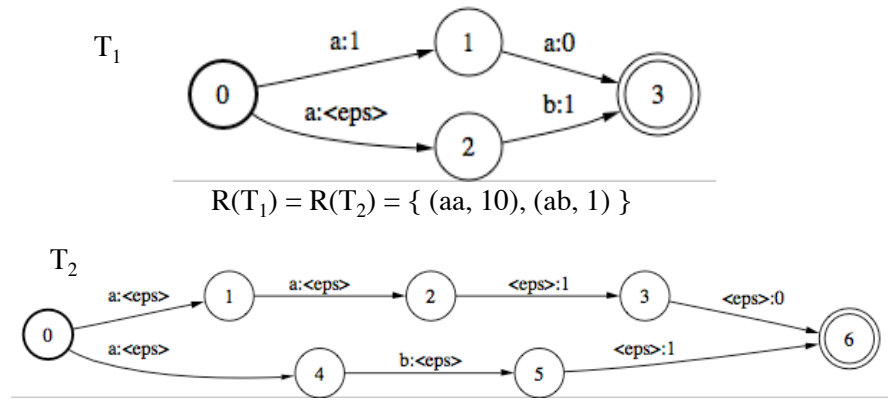
MACM 300

Formal Languages and Automata

Anoop Sarkar

<http://www.cs.sfu.ca/~anoop>

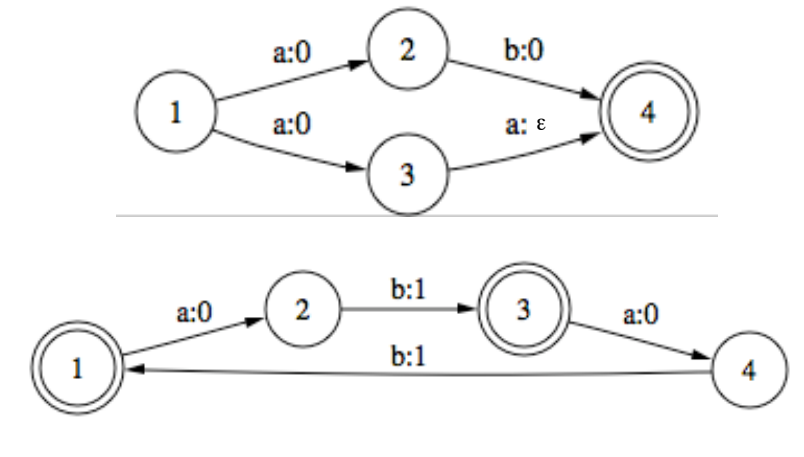
Finite-state transducers



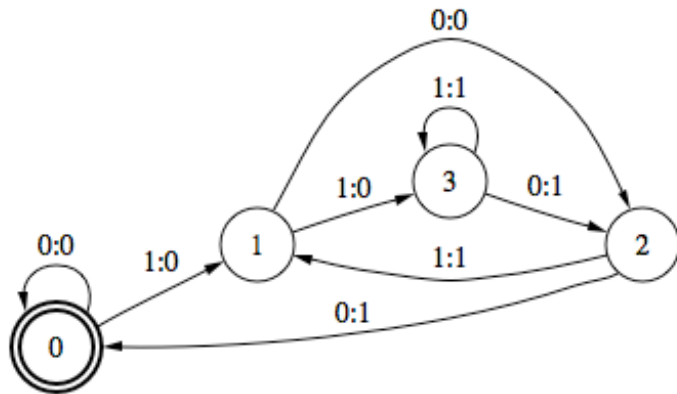
Finite-state transducers

- $a:0$ is a notation for a mapping between two alphabets $a \in \Sigma_1$ and $0 \in \Sigma_2$
- Finite-state transducers (FSTs) accept pairs of strings
- Finite-state automata equate to regular languages and FSTs equate to regular relations
- e.g. $L = \{ (x^n, y^n) \mid n > 0, x \in \Sigma_1 \text{ and } y \in \Sigma_2 \}$ is a regular relation accepted by some FST. It maps a string of x 's into an equal length string of y 's

Finite-state transducers



Finite-state transducers



Finite-state transducers

- Formal definition:
 - Q : finite set of states, q_0, q_1, \dots, q_n
 - Σ : 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

Regular relations

- A generalization of regular languages
- The set of regular relations is:
 - The empty set and (x,y) for all $x, y \in \Sigma_1 \times \Sigma_2$ is a regular relation
 - If R_1, R_2 and R are regular relations then:

$$R_1 \cdot R_2 = \{(x_1x_2, y_1y_2) \mid (x_1, y_1) \in R_1, (x_2, y_2) \in R_2\}$$

$$R_1 \cup R_2$$

$$R^* = \bigcup_{i=0}^{\infty} R_i$$
 - There are no other regular relations

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 *pvcure*
- 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 oofygay*
- *convert numbers into pronunciations, e.g. 230.34 two hundred and thirty point three four

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
 - $(a^n b^*, c^n) \cap (a^* b^n, c^n)$ produces $(a^n b^n, c^n)$
 - However, regular relations with input and output of equal lengths **are** closed under intersection

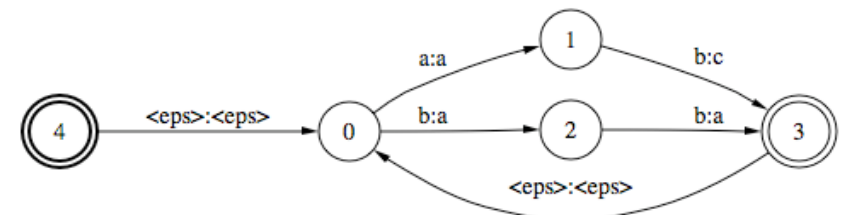
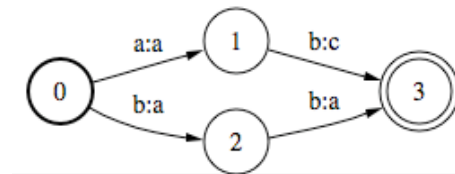
Regular Relations Closure Properties

- New operations for regular relations:
 - *composition*
 - *project* input (or output) language to regular language; for FST t , input language = $\pi_1(t)$, output = $\pi_2(t)$
 - 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 Relations Closure Properties

- 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^+ = one or more repeats of R_1)
 - Kleene closure (R_1^* = 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

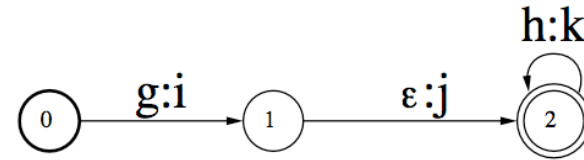
Regular Relation/FST Kleene Closure



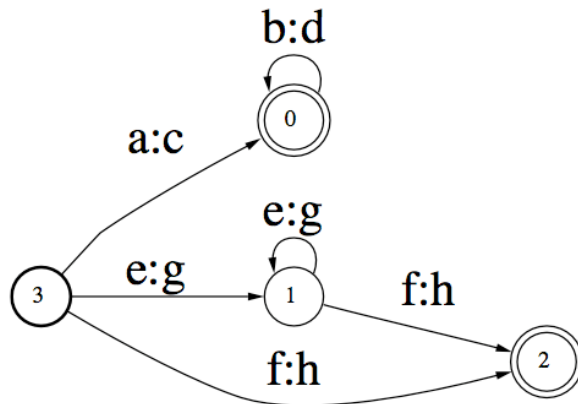
Regular Expressions for FSTs



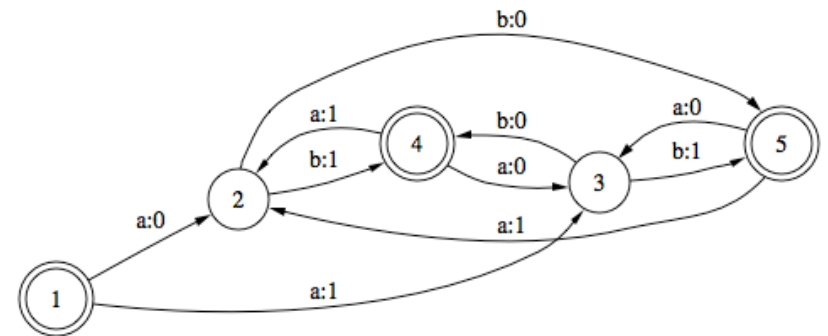
$(a:c) (b:d)^*$



$g:i \epsilon:j (h:k)^*$

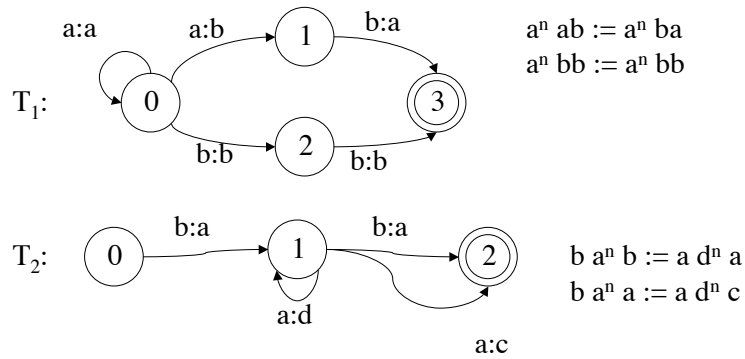


$(a:c (b:d)^*) \cup ((e:g)^* f:h)$



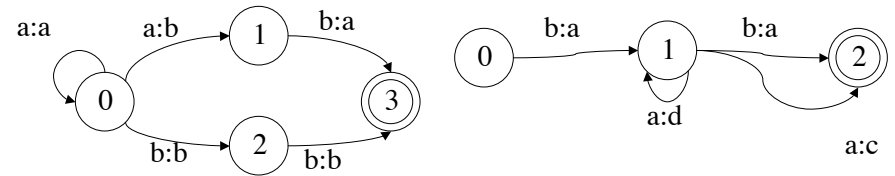
$((a:0 \cup a:1) (b:0 \cup b:1))^*$

FSTs: Closure under Composition



What is T_1 composed with T_2 , written as $T_1 \circ T_2$?

Composing FSTs



$0 \rightarrow 1$ a:b
$0 \rightarrow 2$ b:b
$2 \rightarrow 3$ b:b

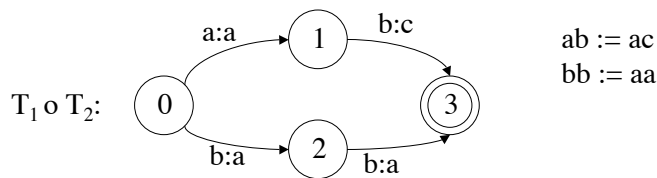
$0 \rightarrow 1$ b:a
$1 \rightarrow 2$ b:a

$0 \rightarrow 0$ a:a
$1 \rightarrow 3$ b:a

$1 \rightarrow 1$ a:d
$1 \rightarrow 2$ a:c

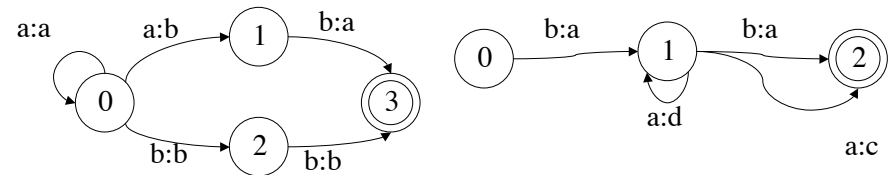
- $(0,0) \rightarrow (1,1)$ a:a
- $(0,0) \rightarrow (2,1)$ b:a
- $(0,1) \rightarrow (0,1)$ a:d
- $(1,1) \rightarrow (3,1)$ b:d
- $(0,1) \rightarrow (1,2)$ a:a
- $(0,1) \rightarrow (2,2)$ b:a
- $(0,1) \rightarrow (0,2)$ a:c
- $(1,1) \rightarrow (3,2)$ b:c
- $(2,0) \rightarrow (3,1)$ b:a
- $(2,1) \rightarrow (3,2)$ b:a

Composing FSTs



- 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



$0 \rightarrow 1$ a:b
$0 \rightarrow 2$ b:b
$2 \rightarrow 3$ b:b

$0 \rightarrow 1$ b:a
$1 \rightarrow 2$ b:a

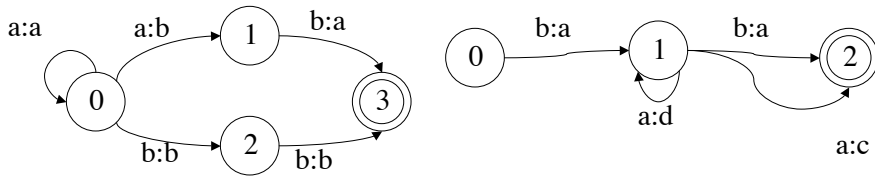
$0 \rightarrow 0$ a:a
$1 \rightarrow 3$ b:a

$1 \rightarrow 1$ a:d
$1 \rightarrow 2$ a:c

- $(0,0) \rightarrow (1,1)$ a:a
- $(0,0) \rightarrow (2,1)$ b:a
- $(0,1) \rightarrow (0,1)$ a:d
- $(1,1) \rightarrow (3,1)$ b:d
- $(0,1) \rightarrow (1,2)$ a:a
- $(0,1) \rightarrow (2,2)$ b:a
- $(0,1) \rightarrow (0,2)$ a:c
- $(1,1) \rightarrow (3,2)$ b:c
- $(2,0) \rightarrow (3,1)$ b:a
- $(2,1) \rightarrow (3,2)$ b:a

start with pair of final states

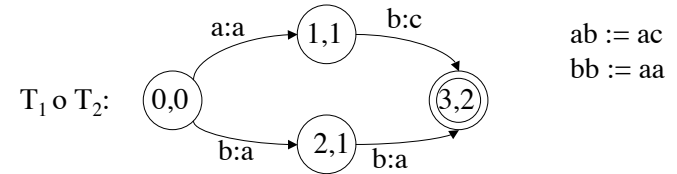
Composing FSTs



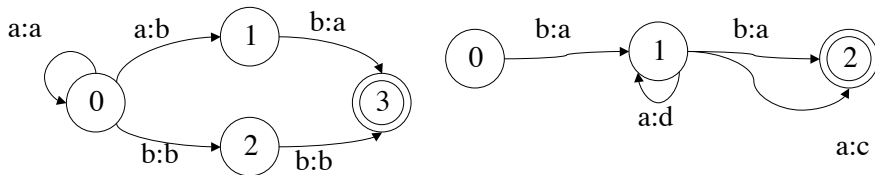
0 → 1 a:b	0 → 1 b:a	0 → 0 a:a	1 → 1 a:d
0 → 2 b:b	1 → 2 b:a	1 → 3 b:a	1 → 2 a:c
2 → 3 b:b			

(0,0) → (1,1) a:a (0,0) → (2,1) b:a (0,1) → (0,1) a:d (1,1) → (3,1) b:d
 (0,1) → (1,2) a:a (0,1) → (2,2) b:a (0,1) → (0,2) a:c (1,1) → (3,2) b:c
 (2,0) → (3,1) b:a (2,1) → (3,2) b:a

Composing FSTs



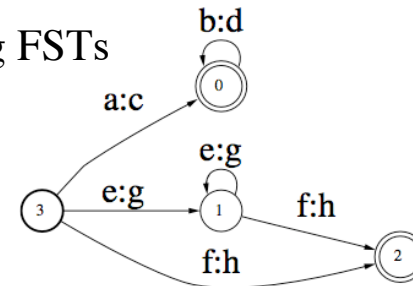
Composing FSTs



0 → 1 a:b	0 → 1 b:a	0 → 0 a:a	1 → 1 a:d
0 → 2 b:b	1 → 2 b:a	1 → 3 b:a	1 → 2 a:c
2 → 3 b:b			

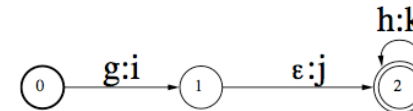
(0,0) → (1,1) a:a (0,0) → (2,1) b:a (0,1) → (0,1) a:d (1,1) → (3,1) b:d
 (0,1) → (1,2) a:a (0,1) → (2,2) b:a (0,1) → (0,2) a:c (1,1) → (3,2) b:c
 (2,0) → (3,1) b:a (2,1) → (3,2) b:a

Composing FSTs

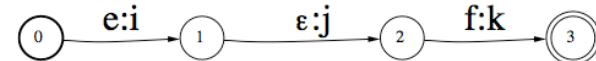


$(a:c (b:d)^*) \cup ((e:g)^* f:h)$

$g:i \epsilon:j (h:k)^*$



$e:i \epsilon:j f:k$



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

FST Composition

- $\text{match}(s,t) =$

$$\{ (s,t) \xrightarrow{x:z} (s',t') \mid s \xrightarrow{x:y} s' \in \text{S.edges and } t \xrightarrow{y:z} t' \in \text{T.edges} \} \cup$$

$$\{ (s,t) \xrightarrow{x:\varepsilon} (s',t) \mid s \xrightarrow{x:\varepsilon} s' \in \text{S.edges} \} \cup$$

$$\{ (s,t) \xrightarrow{\varepsilon:z} (s,t') \mid t \xrightarrow{\varepsilon:z} t' \in \text{T.edges} \}$$
- **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

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:

$$T = (\Sigma_1, \Sigma_2, Q_1 \times Q_2, \langle q_1, q_2 \rangle, F_1 \times F_2, \delta)$$

$$\delta(\langle s_1, s_2 \rangle, a, b) = \delta_1(s_1, a) \times \delta_2(s_2, b)$$

for any $s_1 \in Q_1, s_2 \in Q_2$ and $a, b \in \Sigma \cup \{\varepsilon\}$

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)