Building Finite-State Machines

Xerox Finite-State Tool

- You'll use it for homework …
- Commercial product (but we have academic license here)
  - One of several finite-state toolkits available
  - This one is easiest to use but doesn't have probabilities
- Usage:
  - Enter a regular expression; it builds FSA or FST
  - Now type in input string
    - FSA: It tells you whether it's accepted
    - FST: It tells you all the output strings (if any)
    - Can also invert FST to let you map outputs to inputs
  - Could hook it up to other NLP tools that need finite-state processing of their input or output

Common Regular Expression Operators

	concatenation	EF	
`* +`	iteration	E*, E+	
`|`	union	E | F	
`&`	intersection	E & F	
`~ \ -`	complementation, minus	~E, \x, E-F	
`.x.`	crossproduct	E .x. F	
`.o.`	composition	E .o. F	
`.u`	upper (input) language	E.u	"domain"
`.l.`	lower (output) language	E.l	"range"

What the Operators Mean

- **[blackboard discussion]**
- **[Composition is the most interesting case: see following slides.]**

How to define transducers?

- state set Q
- initial state i
- set of final states F
- input alphabet Σ (also define Σ*, Σ+, Σ?)
- output alphabet Δ
- transition function d: Q x Σ? --> 2^Q
- output function s: Q x Σ? x Q --> Δ*

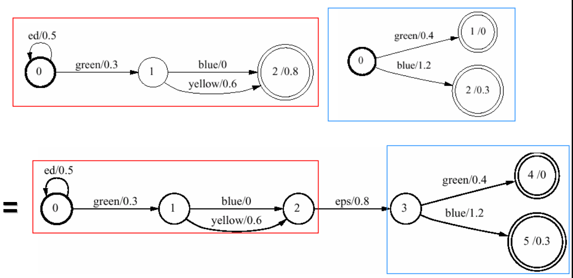
slide courtesy of L. Karttunen (modified)

How to implement?

	concatenation	EF	
`* +`	iteration	E*, E+	
`|`	union	E | F	
`~ \ -`	complementation, minus	~E, \x, E-F	
`&`	intersection	E & F	
`.x.`	crossproduct	E .x. F	
`.o.`	composition	E .o. F	
`.u`	upper (input) language	E.u	"domain"
`.o.`	lower (output) language	E.l	"range"

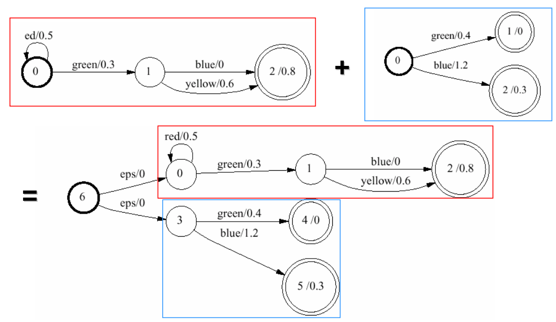
example courtesy of M. Mohri

Concatenation



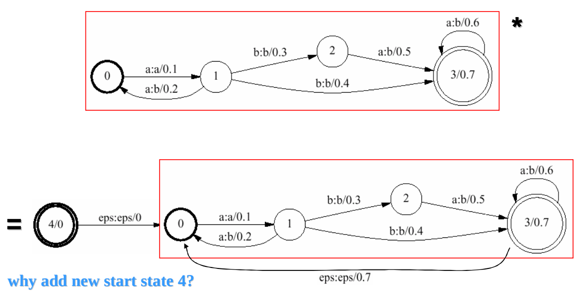
example courtesy of M. Mohri

Union



example courtesy of M. Mohri

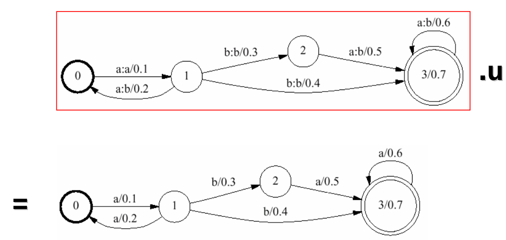
Closure (this example has outputs too)



why add new start state 4?
why not just make state 0 final?

example courtesy of M. Mohri

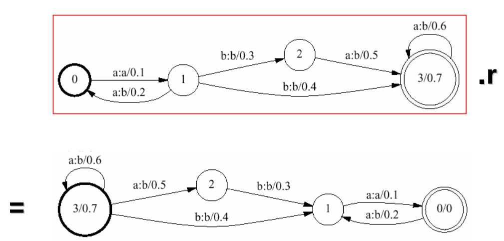
Upper language (domain)



similarly construct lower language .l
also called input & output languages

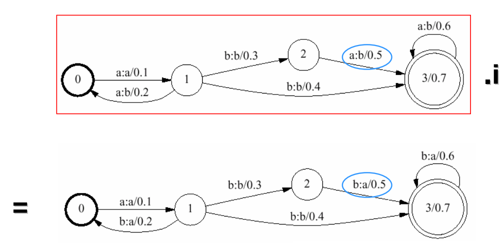
example courtesy of M. Mohri

Reversal



example courtesy of M. Mohri

Inversion

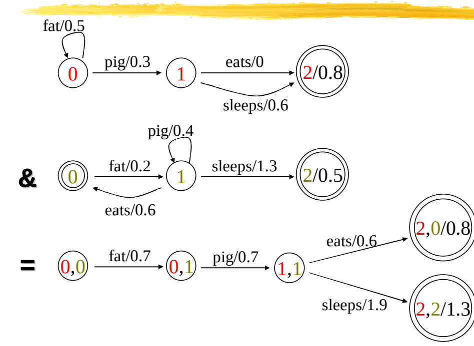


Complementation

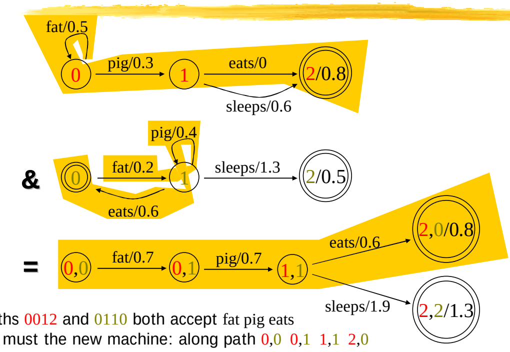
- Given a machine M, represent all strings *not* accepted by M
- Just change final states to non-final and vice-versa
- Works only if machine has been determinized and completed first (why?)

example adapted from M. Mohri

Intersection

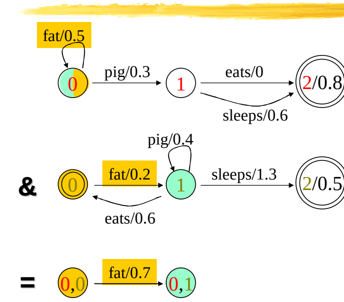


Intersection



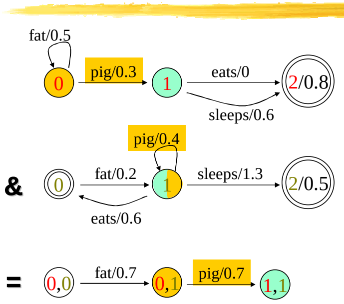
Paths 0012 and 0110 both accept fat pig eats
So must the new machine: along path 0,0 0,1 1,1 2,0

Intersection



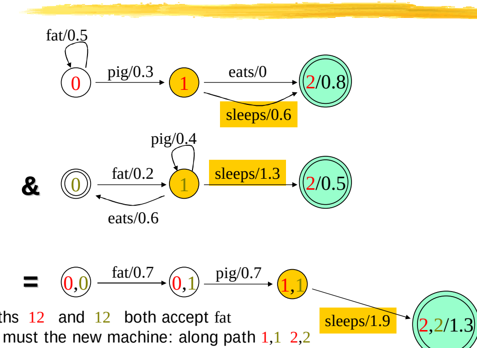
Paths 00 and 01 both accept fat
So must the new machine: along path 0,0 0,1

Intersection



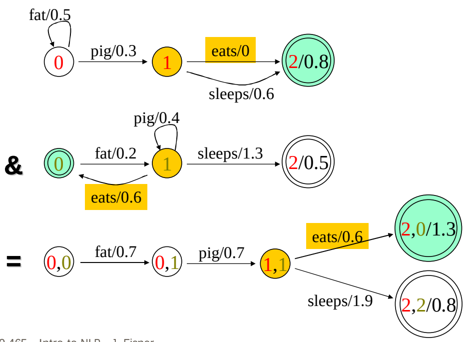
Paths 00 and 11 both accept pig
So must the new machine: along path 0,1 1,1

Intersection

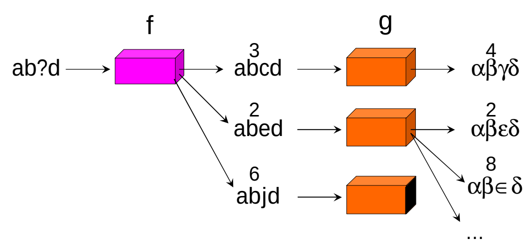


Paths 12 and 12 both accept fat
So must the new machine: along path 1,1 2,2

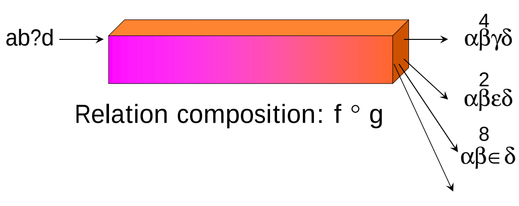
Intersection



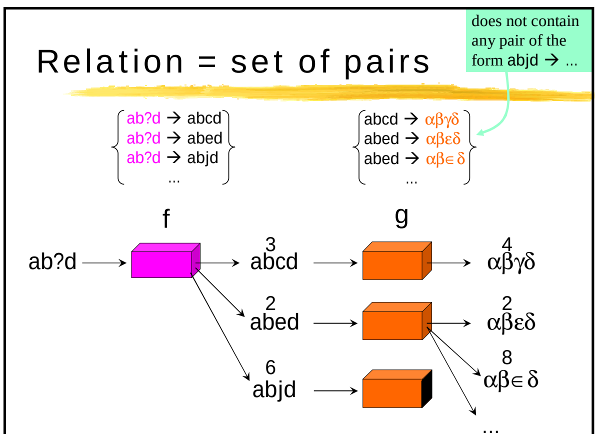
What Composition Means



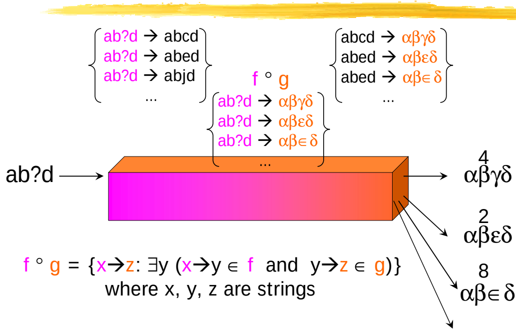
What Composition Means



Relation = set of pairs

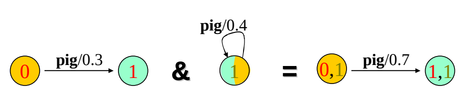


Relation = set of pairs



Intersection vs. Composition

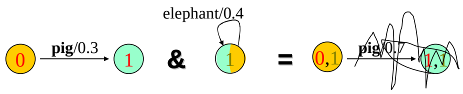
Intersection



Composition

Intersection vs. Composition

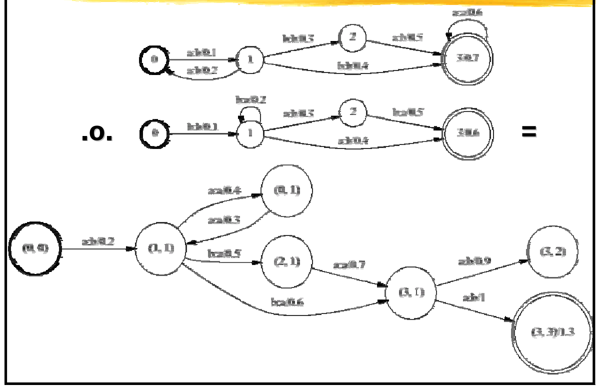
Intersection mismatch



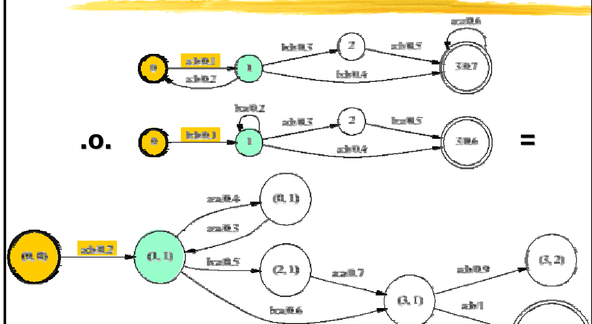
Composition mismatch

Composition

example courtesy of M. Mohri

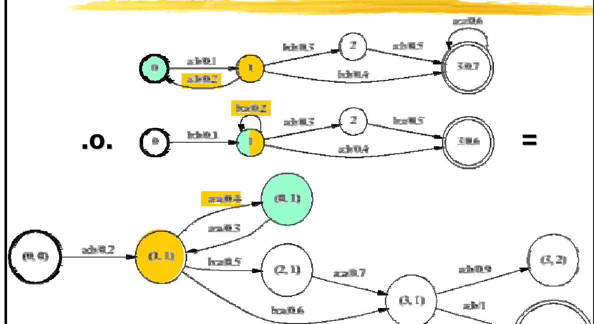


Composition



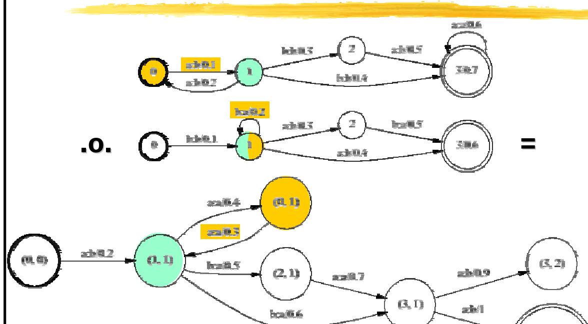
a:b .o. b:b = a:b

Composition



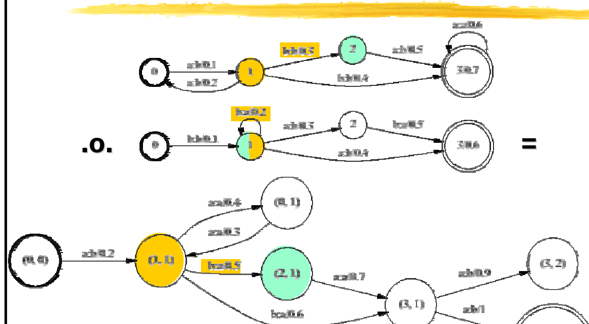
a:b .o. b:a = a:a

Composition



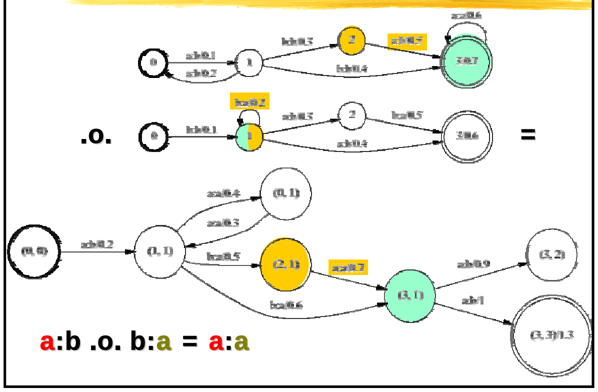
a:b .o. b:a = a:a

Composition



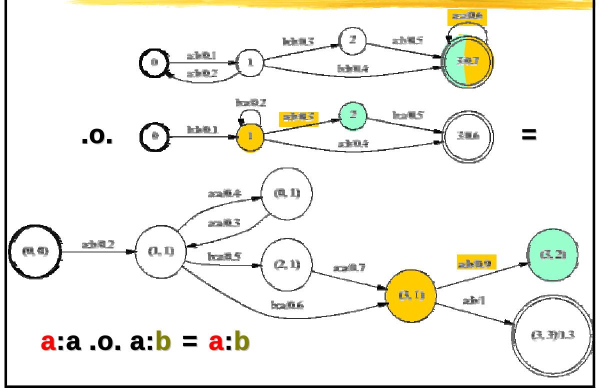
b:b .o. b:a = b:a

Composition



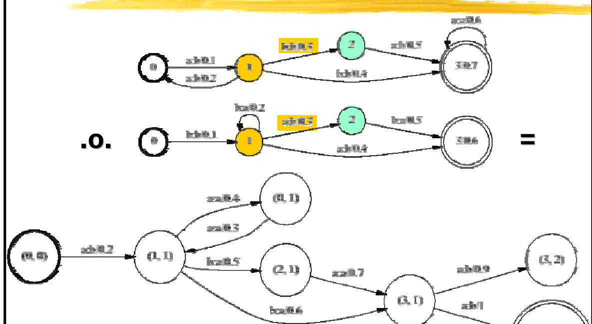
a:b .o. b:a = a:a

Composition



a:a .o. a:b = a:b

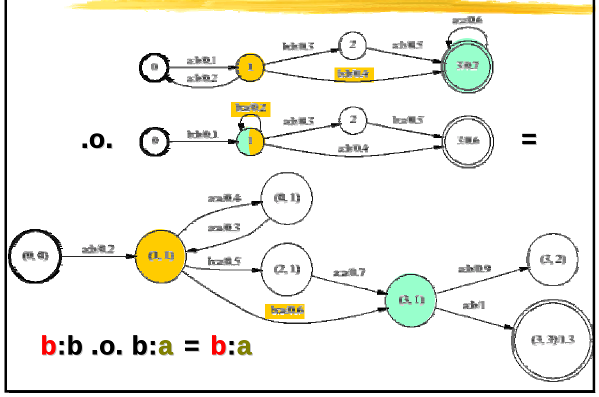
Composition



b:b .o. a:b = nothing

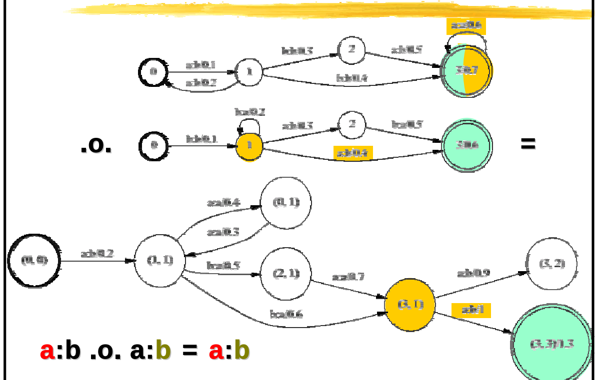
(since intermediate symbol doesn't match)

Composition



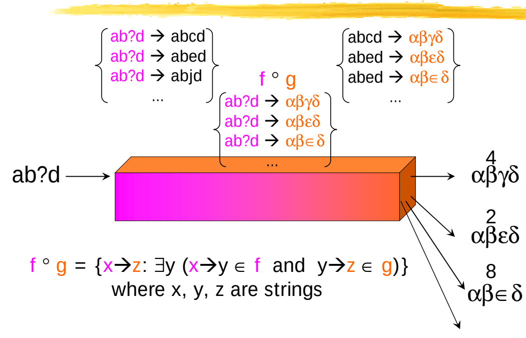
b:b .o. b:a = b:a

Composition



a:b .o. a:b = a:b

Relation = set of pairs



$f \circ g = \{x \rightarrow z: \exists y\ (x \rightarrow y \in f \text{ and } y \rightarrow z \in g)\}$

where x, y, z are strings

Composition with Sets

- We've defined A .o. B where both are FSTs
- Now extend definition to allow one to be a FSA
- Two relations (FSTs):
  $A \circ B = \{x \rightarrow z: \exists y\ (x \rightarrow y \in A \text{ and } y \rightarrow z \in B)\}$
- Set and relation:
  $A \circ B = \{x \rightarrow z: \quad x \in A \text{ and } x \rightarrow z \in B\}$
- Relation and set:
  $A \circ B = \{x \rightarrow z: \quad x \rightarrow z \in A \text{ and } \quad z \in B\}$
- Two sets (acceptors) – same as intersection:
  $A \circ B = \{x: \quad x \in A \text{ and } \quad x \in B\}$

Composition and Coercion

- Really just treats a set as identity relation on set
  {abc, pqr, …} = {abc→abc, pqr→pqr, …}
- Two relations (FSTs):
  $A \circ B = \{x \rightarrow z: \exists y\ (x \rightarrow y \in A \text{ and } y \rightarrow z \in B)\}$
- Set and relation is now special case (if $\exists y$ then y=x):
  $A \circ B = \{x \rightarrow z: \quad x \rightarrow x \in A \text{ and } x \rightarrow z \in B\}$
- Relation and set is now special case (if $\exists y$ then y=z):
- $A \circ B = \{x \rightarrow z: \quad x \rightarrow z \in A \text{ and } z \rightarrow z \in B\}$
- Two sets (acceptors) is now special case:
  $A \circ B = \{x \rightarrow z: \quad x \rightarrow x \in A \text{ and } x \rightarrow x \in B\}$

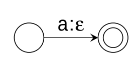
3 Uses of Set Composition:

- **Feed string into Greek transducer:**
  - {abed→abed} .o. Greek = {abed→αβεδ, abed→αβ∊δ}
  - {abed} .o. Greek = {abed→αβεδ, abed→αβ∊δ}
  - [{abed} .o. Greek].l = {αβεδ, αβ∊δ}
- **Feed several strings in parallel:**
  - {abcd, abed} .o. Greek
    = {abcd→αβγδ, abed→αβεδ, abed→αβ∊δ}
  - [{abcd,abed} .o. Greek].l = {αβγδ, αβεδ, αβ∊δ}
- **Filter result via Noε** = {αβγδ, αβ∊δ, …}
  - {abcd,abed} .o. Greek .o. Noε
    = {abcd→αβγδ, abed→αβ∊δ}

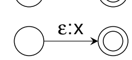
What are the "basic" transducers?

- The operations on the previous slides combine transducers into bigger ones
- But where do we start?

- a:ε for $a \in \Sigma$



- ε:x for $x \in \Delta$



- Q: Do we also need a:x? How about ε:ε ?