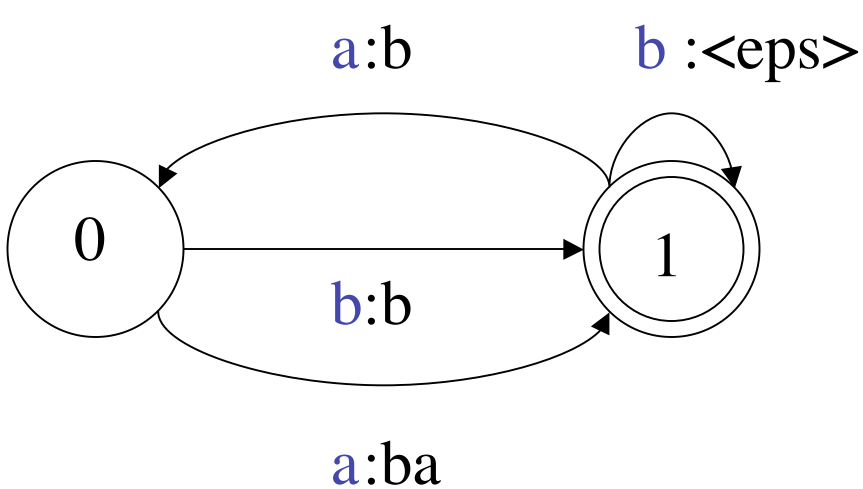
CMPT 825
Natural Language Processing

Anoop Sarkar

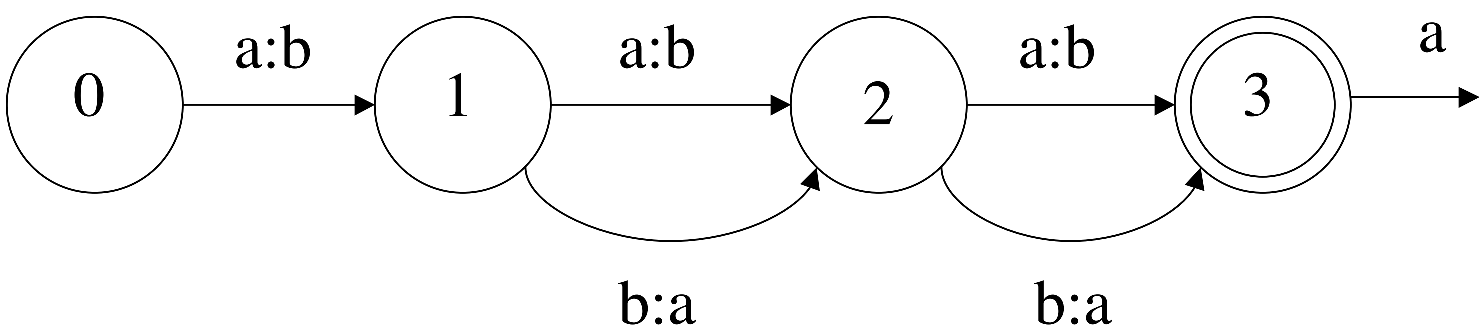
http://www.cs.sfu.ca/~anoop

Sequential transducers

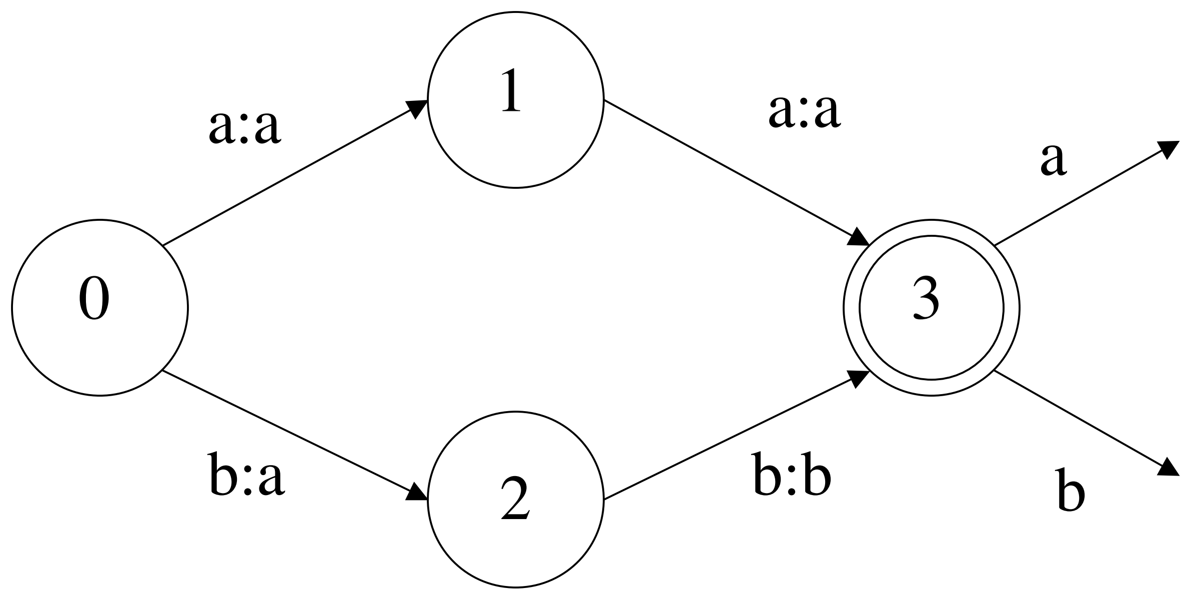


- determinization
- minimization

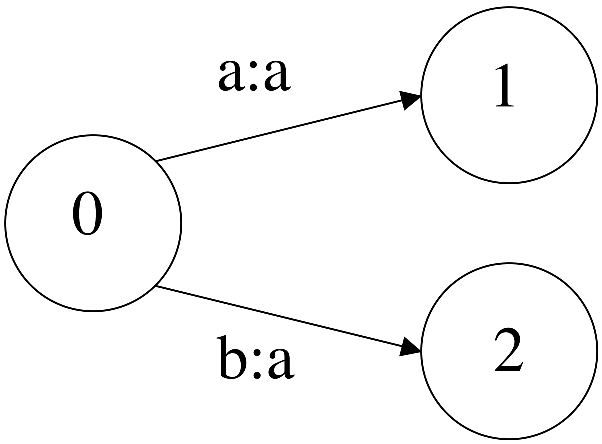
Subsequential transducers

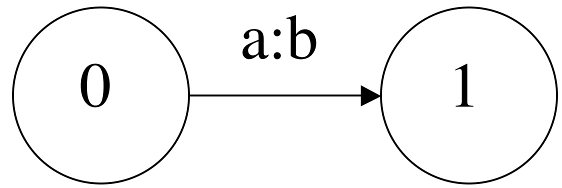


p-subsequential Transducers



Composition: T_1 o T_2



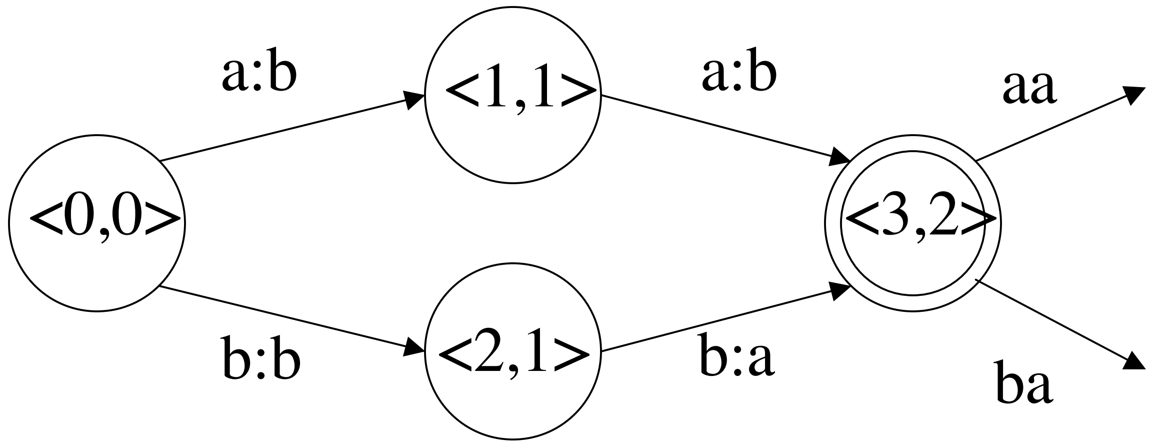


$\langle 0,0 \rangle$
$\langle 0,1 \rangle$
$\langle 1,0 \rangle$
$\langle 1,1 \rangle$
$\langle 2,0 \rangle$
$\langle 2,1 \rangle$

$\langle 0,0 \rangle \rightarrow \langle\, \mathbf{tf}_1(0,a)\ ,\ \mathbf{tf}_2(0,\ \mathbf{out}_1(0,a))\rangle \rightarrow \langle 1,1 \rangle$

$\langle 0,0 \rangle \rightarrow \langle\, \mathbf{tf}_1(0,b)\ ,\ \mathbf{tf}_2(0,\ \mathbf{out}_1(0,b))\rangle \rightarrow \langle 2,1 \rangle$

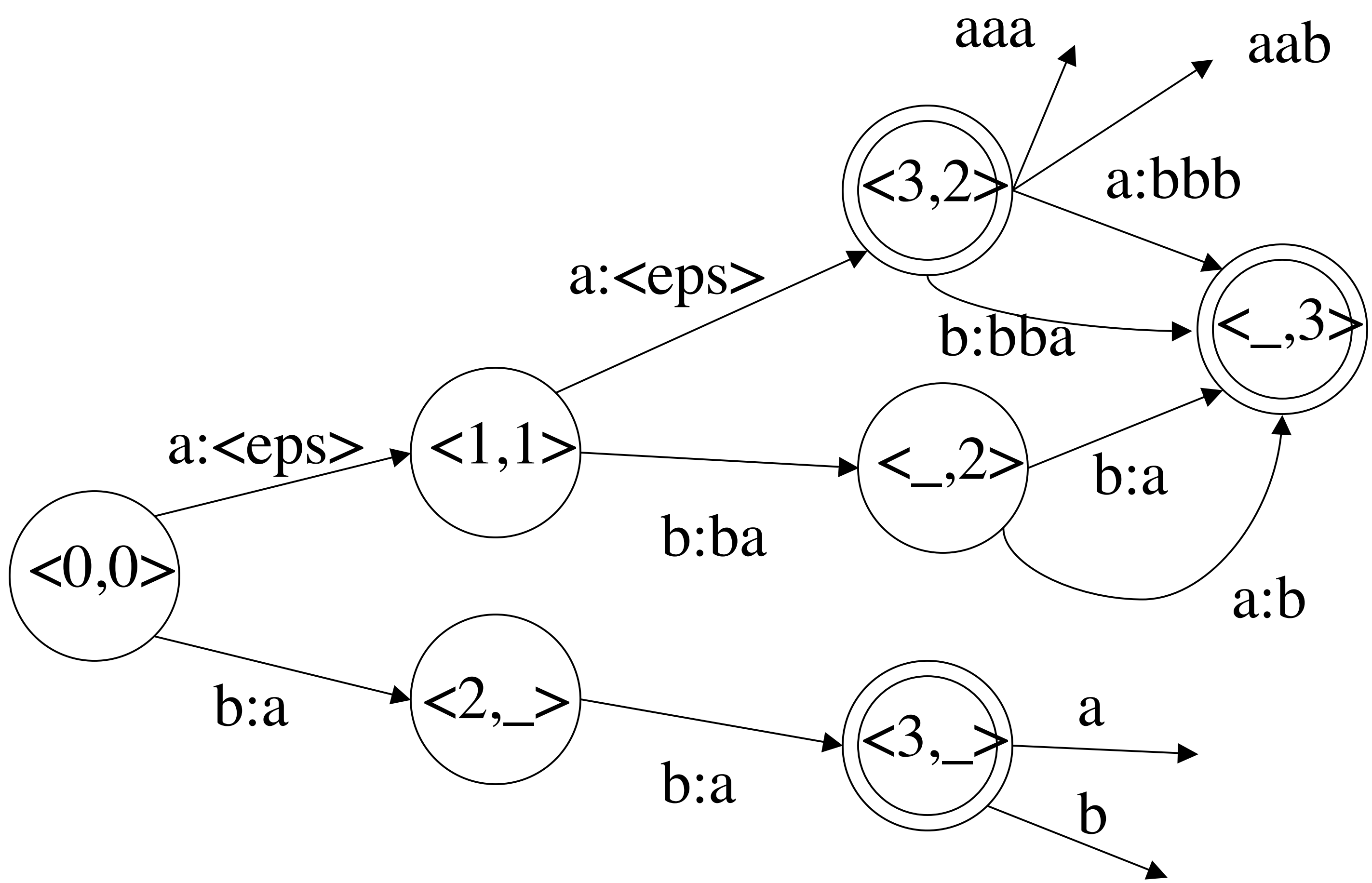
Composition: T_1 o T_2



$$\langle 0,0\rangle \to \langle\ \mathbf{tf}_1(0,a)\ ,\ \mathbf{tf}_2(0,\ \mathbf{out}_1(0,a))\rangle \to \langle 1,1\rangle$$

$$\langle 0,0\rangle \to \langle\ \mathbf{tf}_1(0,b)\ ,\ \mathbf{tf}_2(0,\ \mathbf{out}_1(0,b))\rangle \to \langle 2,1\rangle$$

Union: $T_1 + T_2$



Arbitrary Transducers

- We've seen algorithms over subsequential transducers: but what about arbitrary transducers?
- There exist transducers with no sequential equivalent. e.g. $a^{|w|}$ if $|w|$ is even; else $b^{|w|}$
- If f is a transducer, then there is a left sequential l and a right sequential r such that $f = l \bullet r$
- It is decidable if a transducer is sequential
- See details in Mohri 1997.

Slides on Minimization taken from the presentation made by Jason Eisner at the following conference talk

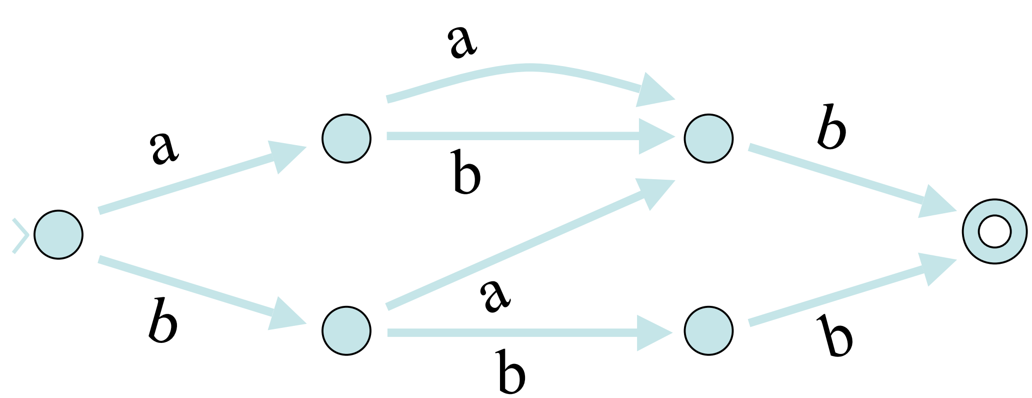
Simpler & More General Minimization for Weighted Finite-State Automata

Jason Eisner
Johns Hopkins University

May 28, 2003 — HLT-NAACL

The Minimization Problem

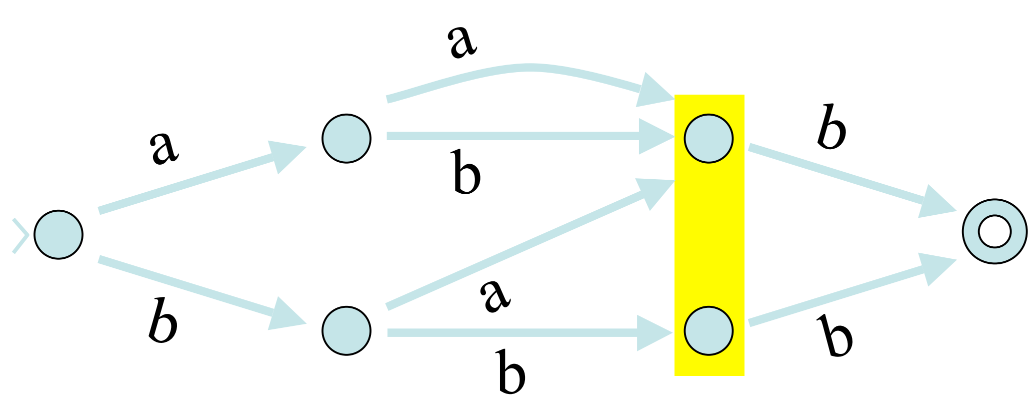
Input:	A DFA	(deterministic finite-state automaton)
Output:	An equiv. DFA with as few states as possible	
Complexity:	O(|arcs| log |states|)	(Hopcroft 1971)



Represents the language {aab, abb, bab, bbb}

The Minimization Problem

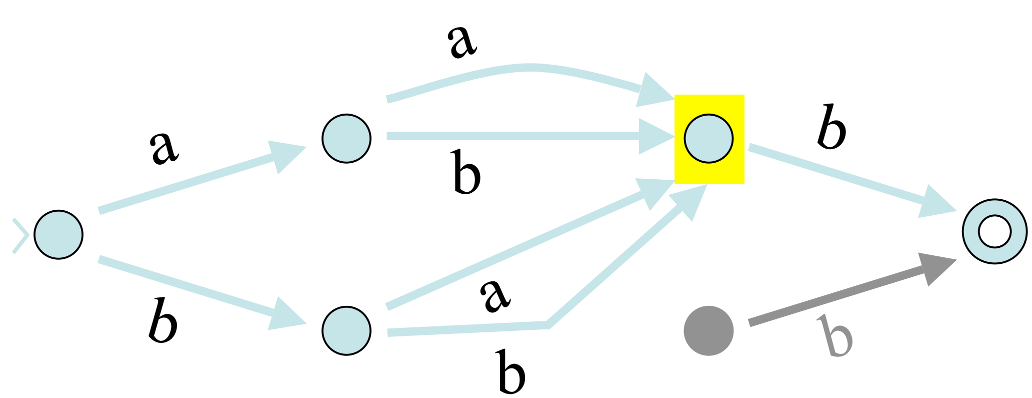
Input:	A DFA	(deterministic finite-state automaton)
Output:	An equiv. DFA with as few states as possible	
Complexity:	O(|arcs| log |states|)	(Hopcroft 1971)



Represents the language {aab, abb, bab, bbb}

The Minimization Problem

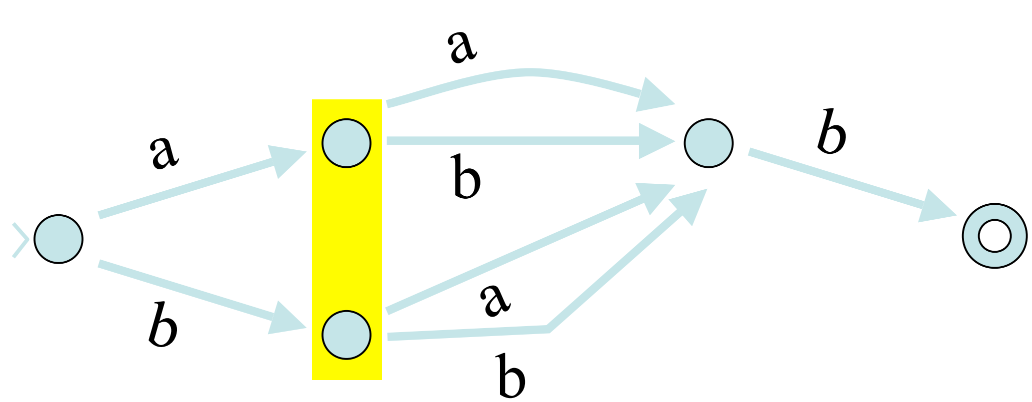
Input:	A DFA	(deterministic finite-state automaton)
Output:	An equiv. DFA with as few states as possible	
Complexity:	O(|arcs| log |states|)	(Hopcroft 1971)



Represents the language {aab, abb, bab, bbb}

The Minimization Problem

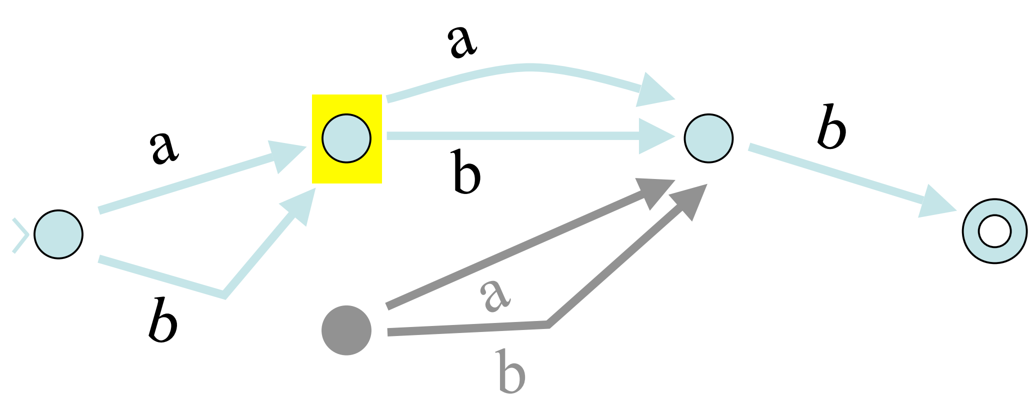
Input:	A DFA	(deterministic finite-state automaton)
Output:	An equiv. DFA with as few states as possible	
Complexity:	O(|arcs| log |states|)	(Hopcroft 1971)



Represents the language {aab, abb, bab, bbb}

The Minimization Problem

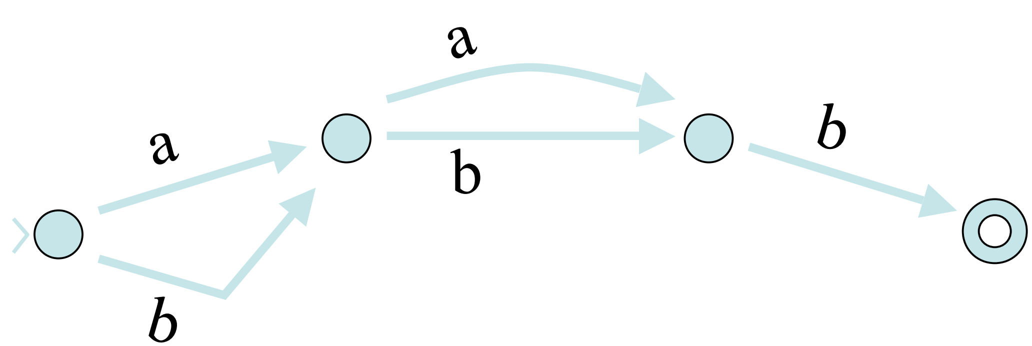
Input:	A DFA	(deterministic finite-state automaton)
Output:	An equiv. DFA with as few states as possible	
Complexity:	O(|arcs| log |states|)	(Hopcroft 1971)



Represents the language {aab, abb, bab, bbb}

The Minimization Problem

Input:	A DFA	(deterministic finite-state automaton)
Output:	An equiv. DFA with as few states as possible	
Complexity:	O(|arcs| log |states|)	(Hopcroft 1971)



Can't always work backward from final state like this.
A bit more complicated because of cycles.

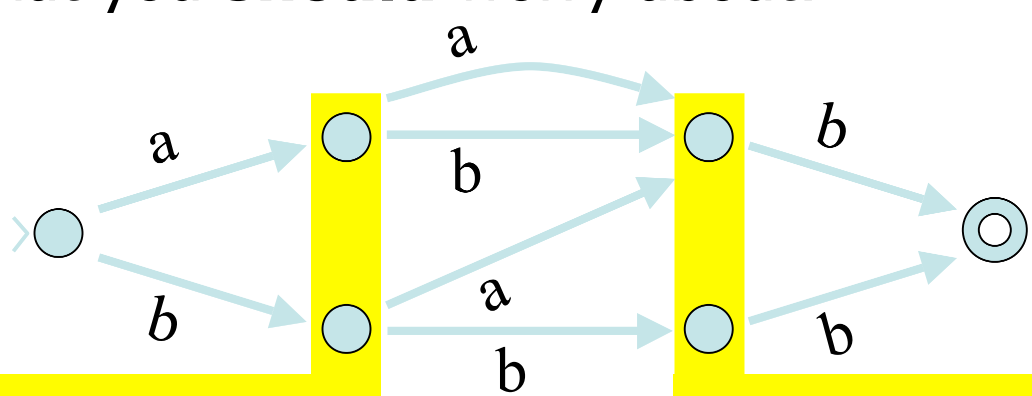
The Minimization Problem

Input: A DFA (deterministic finite-state automaton)

Output: An equiv. DFA with as few states as possible

Complexity: O(|arcs| log |states|) (Hopcroft 1971)

Here's what you **should** worry about:



Mergeable because they have the same **suffix language**: {ab,bb}

Mergeable because they have the same **suffix language**: {b}

An equivalence relation on states … merge the equivalence classes

The Minimization Problem

Input:	A DFA	(deterministic finite-state automaton)
Output:	An equiv. DFA with as few states as possible	
Complexity:	O(|arcs| log |states|)	(Hopcroft 1971)

Q: Why minimize # states, rather than # arcs?

A: Minimizing # states also minimizes # arcs!

Q: What if the input is an NDFA (nondeterministic)?

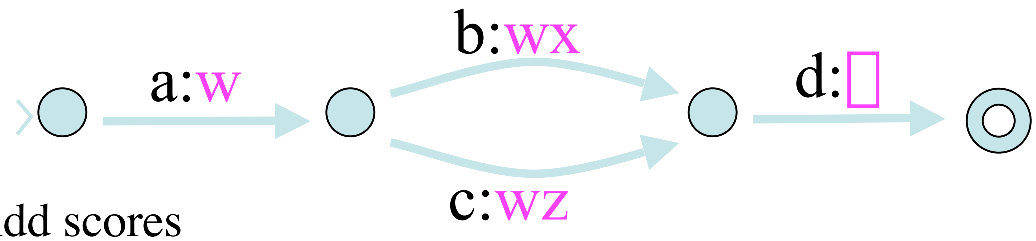
A: Determinize it first. (could yield exponential blowup ☹)

Q: How about minimizing an NDFA to an NDFA?

A: Yes, could be exponentially smaller ☺,
but problem is PSPACE-complete so we don't try. ☹

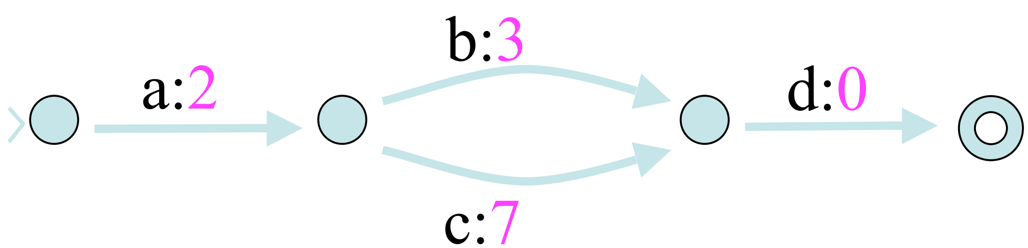
Real-World NLP:
Automata With Weights or Outputs

- Finite-state computation of functions
  - Concatenate strings



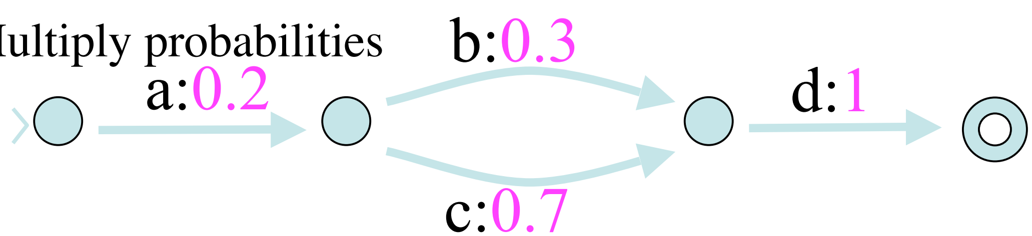
abd □ wwx
acd □ wwz

  - Add scores



abd □ 5
acd □ 9

  - Multiply probabilities

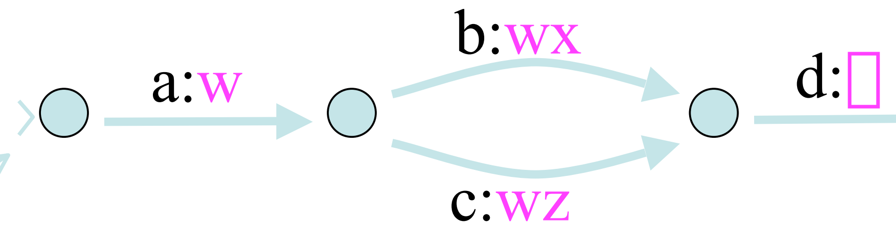


abd □ 0.06
acd □ 0.14

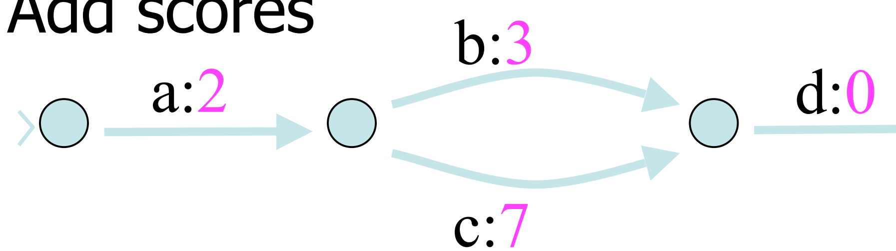
Weight Algebras

- Specify a weight algebra (K,   )
- Define DFAs over (K,   )
- Arcs have weights in set K
- A path's weight is also in K: multiply its arc weights with
- Examples:
  - (strings, concatenation)
  - (scores, addition)
  - (probabilities, multiplication)
  - (score vectors, addition)
  - (real weights, multiplication)
  - (objective func & gradient, product-rule multiplication)
  - (bit vectors, conjunction)

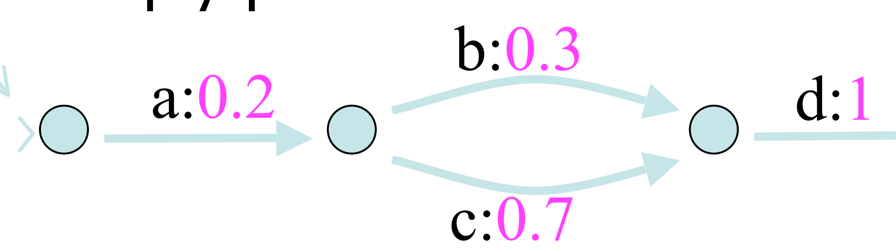
- Finite-state computation of fu
  - Concatenate strings



  - Add scores



  - Multiply probabilities

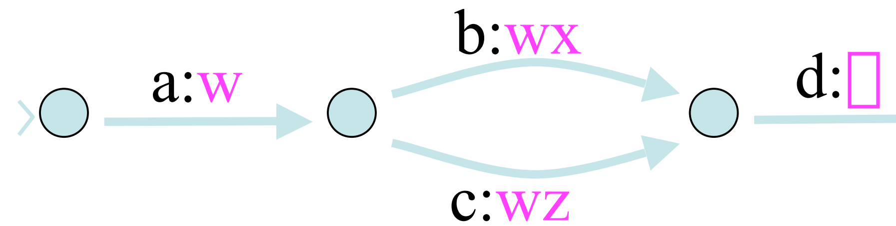


Weight Algebras

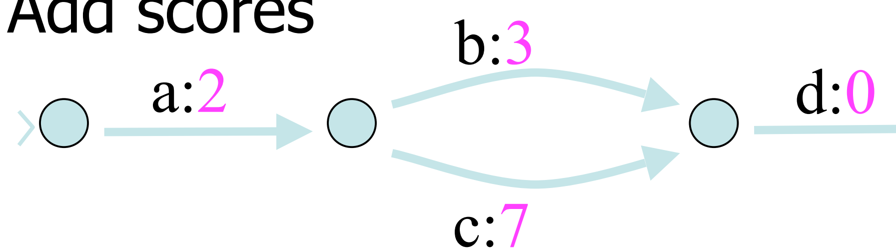
- Specify a weight algebra (K,)
- Define DFAs over (K,)
- Arcs have weights in set K
- A path's weight is also in K: multiply its arc weights with

- Q: Semiring is (K,⊕,). Why not talk about ⊕ too?
- A: Minimization is about DFAs.
- At most one path per input.
- So no need to ⊕ the weights of multiple accepting paths.

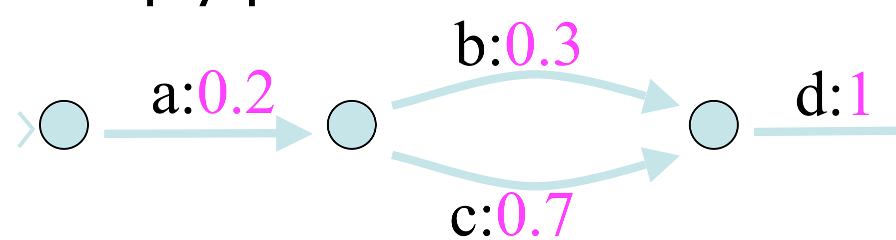
- Finite-state computation of fu
  - Concatenate strings



  - Add scores

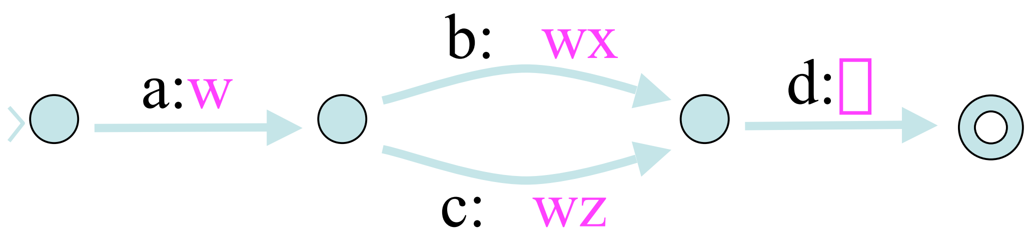


  - Multiply probabilities



Shifting Outputs Along Paths

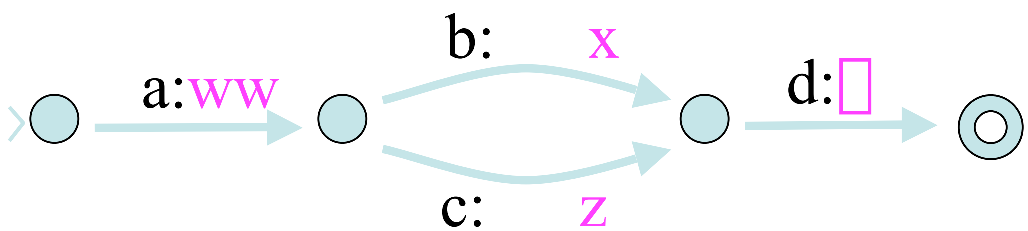
- Doesn't change the function computed:



abd □ wwx
acd □ wwz

Shifting Outputs Along Paths

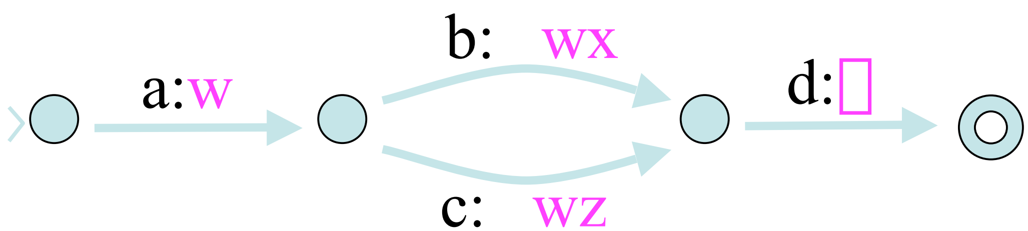
- Doesn't change the function computed:



abd □ wwx
acd □ wwz

Shifting Outputs Along Paths

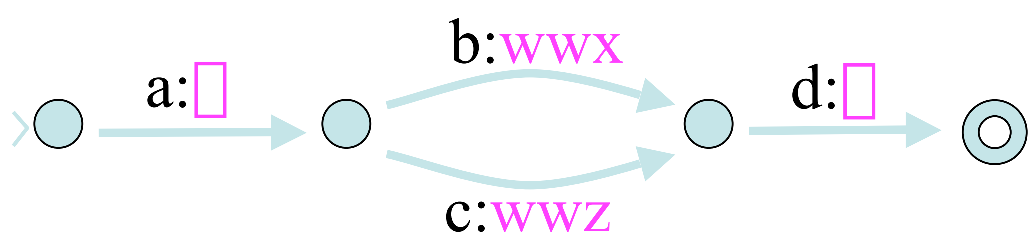
- Doesn't change the function computed:



abd □ wwx
acd □ wwz

Shifting Outputs Along Paths

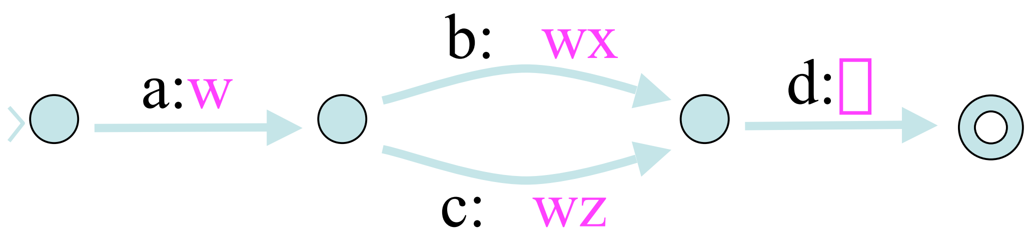
- Doesn't change the function computed:



abd [illegible] wwx
acd [illegible] wwz

Shifting Outputs Along Paths

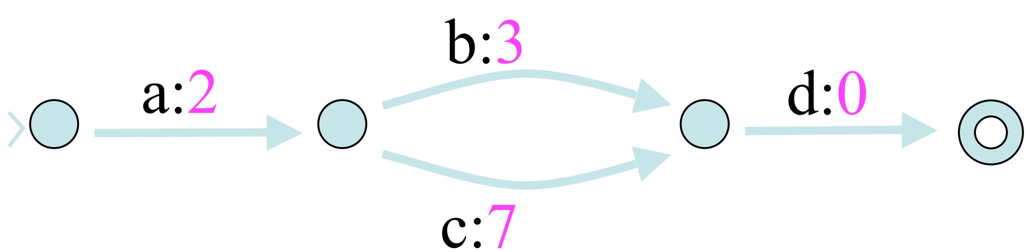
- Doesn’t change the function computed:



abd ▯ wwx
acd ▯ wwz

Shifting Outputs Along Paths

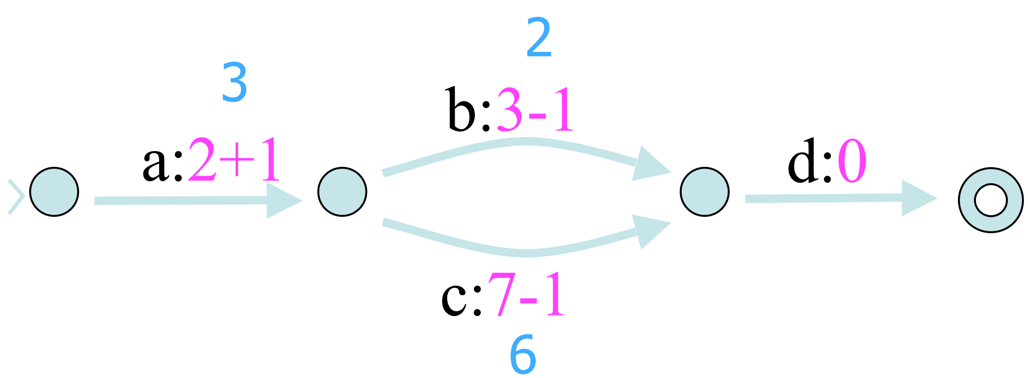
- Doesn't change the function computed:



abd □ 5
acd □ 9

Shifting Outputs Along Paths

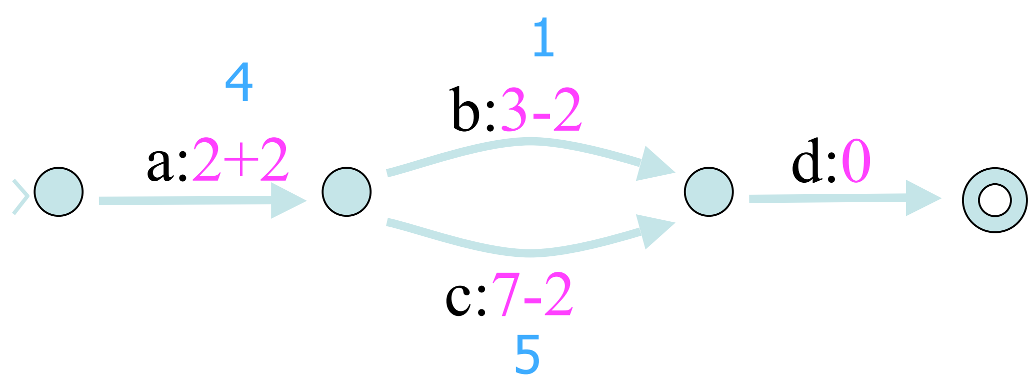
- Doesn't change the function computed:



abd □ 5
acd □ 9

Shifting Outputs Along Paths

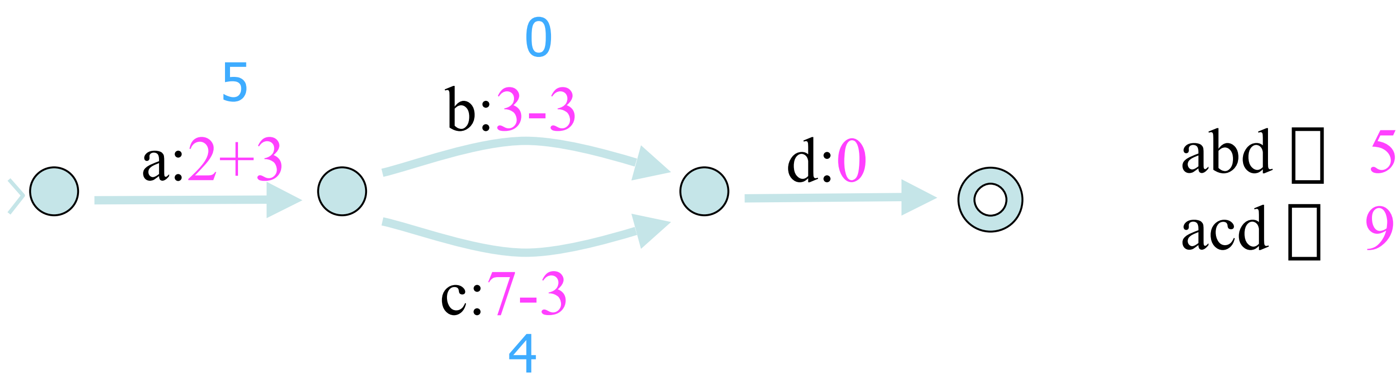
- Doesn't change the function computed:



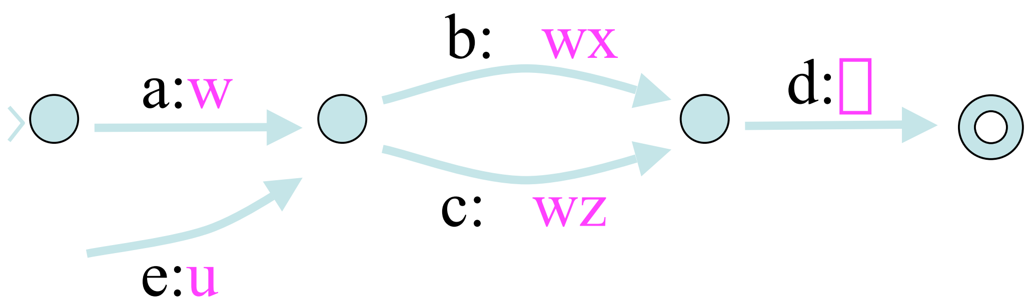
abd □ 5
acd □ 9

Shifting Outputs Along Paths

- Doesn't change the function computed:



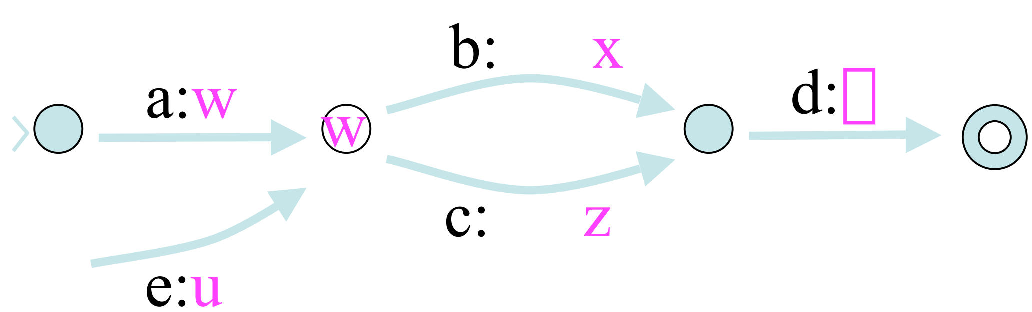
Shifting Outputs Along Paths



abd □ wwx
acd □ wwz
…ebd □ uwx
…ecd □ uwz

Shifting Outputs Along Paths

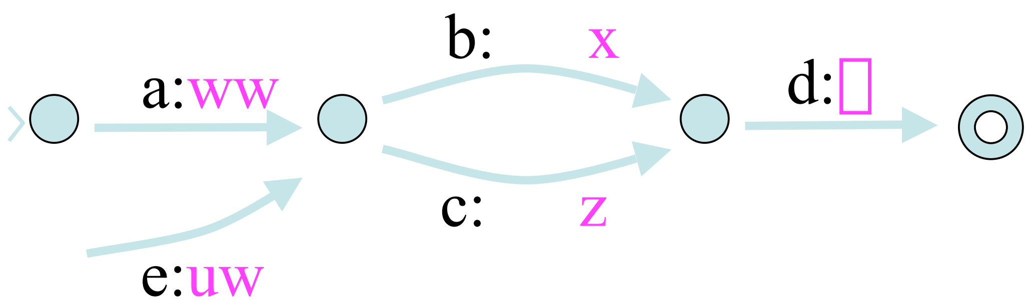
- State sucks back a prefix from its out-arcs



abd wwx
acd wwz
…ebd uwx
…ecd uwz

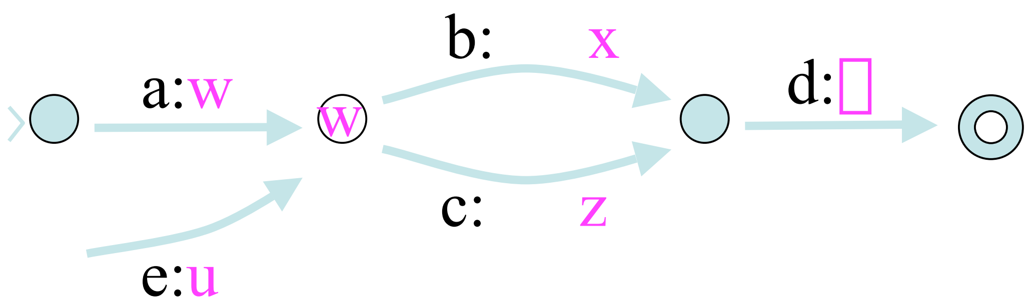
Shifting Outputs Along Paths

- State sucks back a prefix from its out-arcs and deposits it at end of its in-arcs.



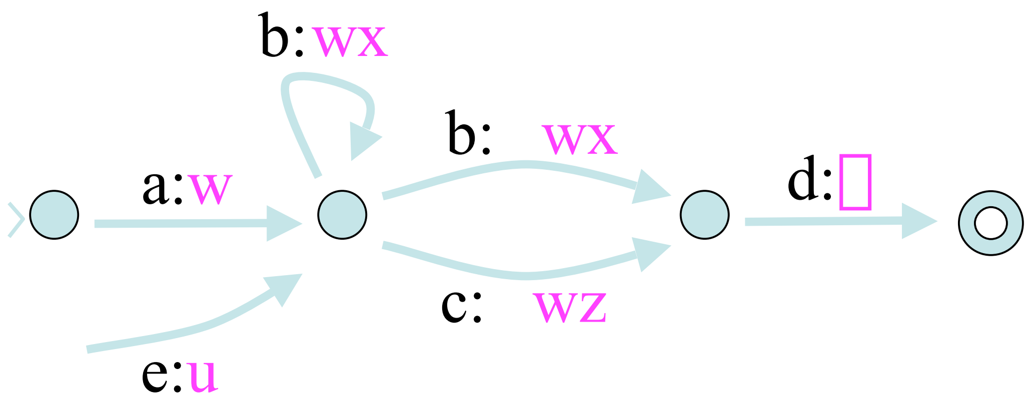
abd □ wwx
acd □ wwz
…ebd □ uwx
…ecd □ uwz

Shifting Outputs Along Paths



abd □ wwx
acd □ wwz
…ebd □ uwx
…ecd □ uwz

Shifting Outputs Along Paths



abd □ wwx

acd □ wwz

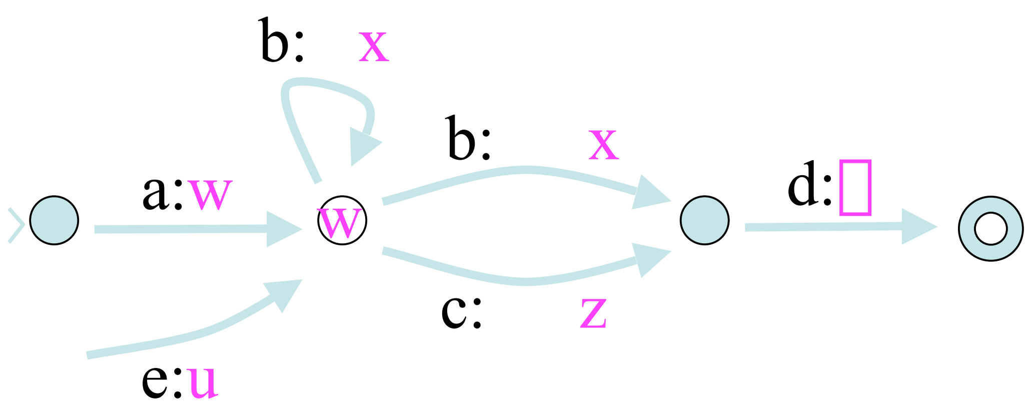
…ebd □ uwx

…ecd □ uwz

…ab^{n}bd □ u(wx)^{n}wx

…ab^{n}cd □ u(wx)^{n}wz

Shifting Outputs Along Paths



abd □ wwx

acd □ wwz

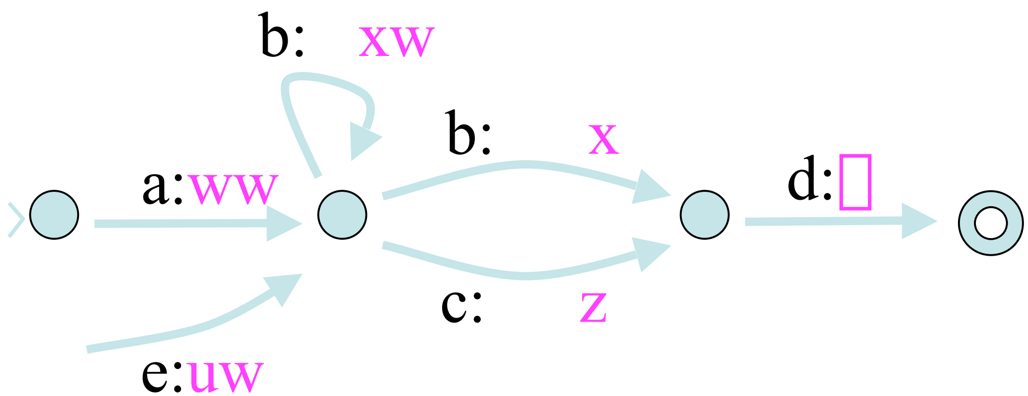
…ebd □ uwx

…ecd □ uwz

…$ab^n bd$ □ $u(wx)^n wx$

…$ab^n cd$ □ $u(wx)^n wz$

Shifting Outputs Along Paths

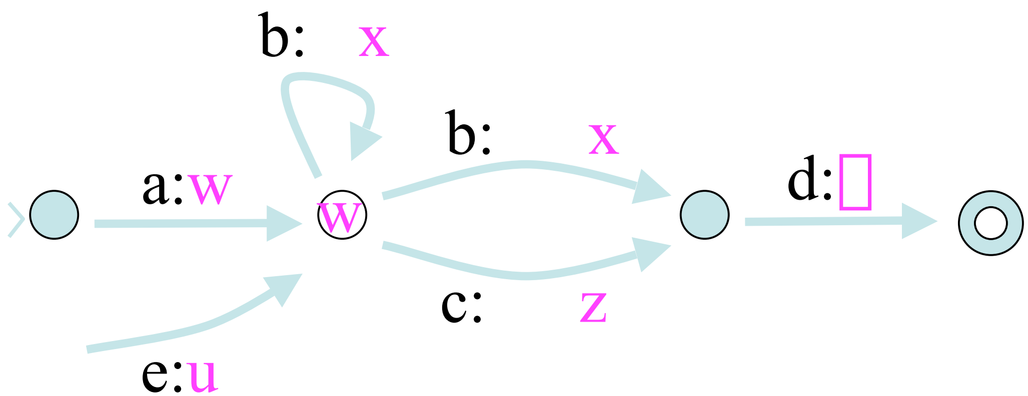


abd □ wwx
acd □ wwz
…ebd □ uwx
…ecd □ uwz

…ab^{n}bd □ u(wx)^{n}wx
…ab^{n}cd □ u(wx)^{n}wz

…ab^{n}bd □ uw(xw)^{n}x
…ab^{n}cd □ uw(xw)^{n}z

Shifting Outputs Along Paths



abd → wwx

acd → wwz

…ebd → uwx

…ecd → uwz

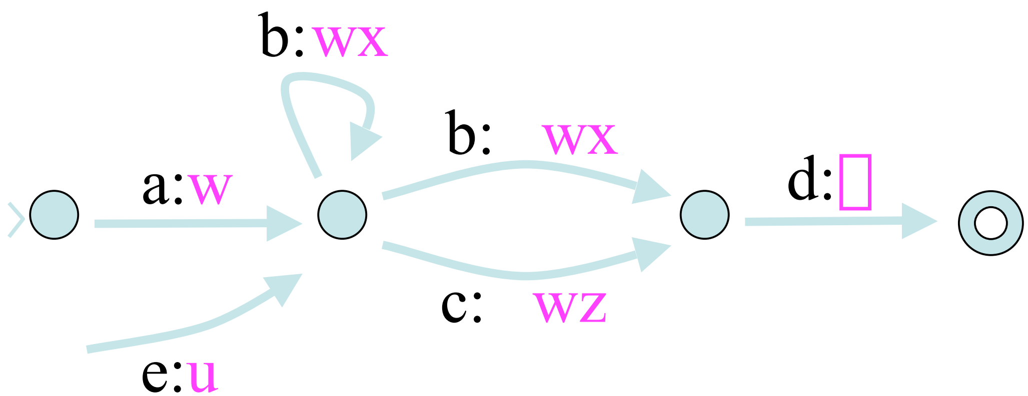
…ab^{n}bd → u(wx)^{n}wx

…ab^{n}cd → u(wx)^{n}wz

…ab^{n}bd → uw(xw)^{n}x

…ab^{n}cd → uw(xw)^{n}z

Shifting Outputs Along Paths



abd □ wwx

acd □ wwz

…ebd □ uwx

…ecd □ uwz

…ab^nbd □ $u(wx)^nwx$

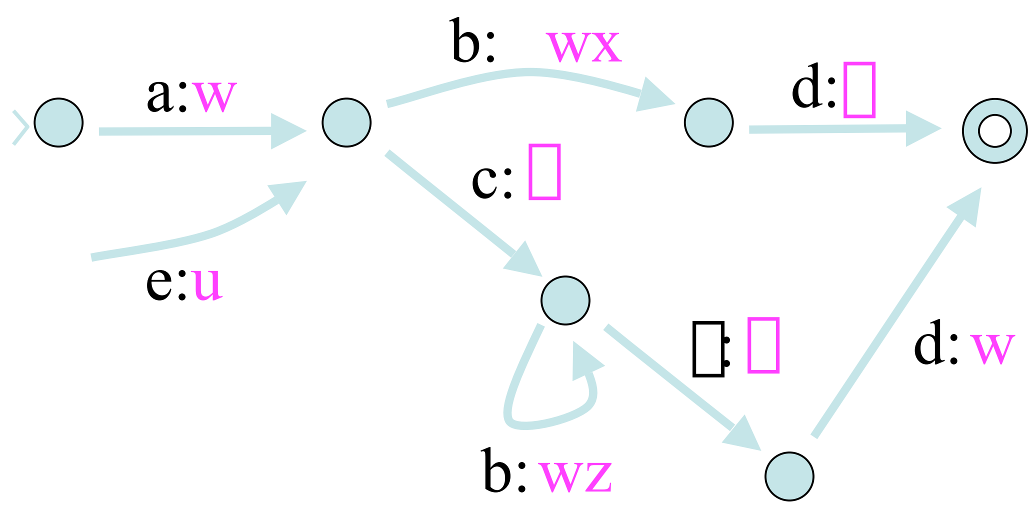
…ab^ncd □ $u(wx)^nwz$

…ab^nbd □ $uw(xw)^nx$

…ab^ncd □ $uw(xw)^nz$

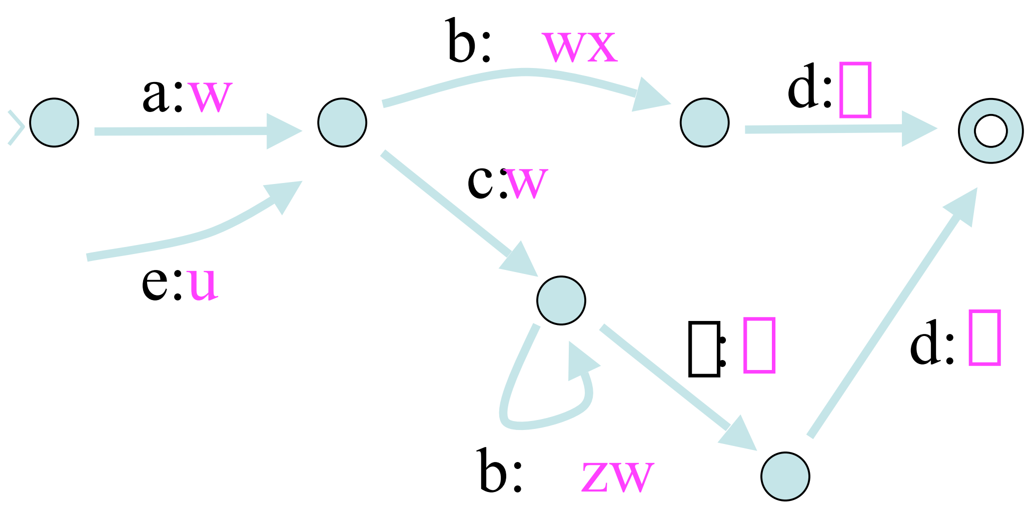
Shifting Outputs Along Paths (Mohri)

- Here, not all the out-arcs start with w
- But all the out-**paths** start with w
- Do pushback at later states first:



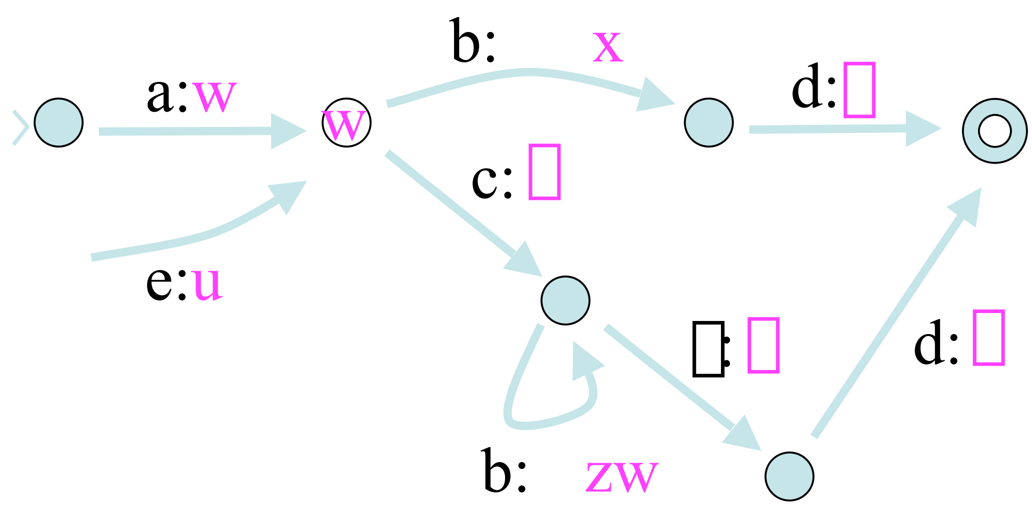
Shifting Outputs Along Paths (Mohri)

- Here, not all the out-arcs start with w
- But all the out-**paths** start with w
- Do pushback at later states first: now we're ok!



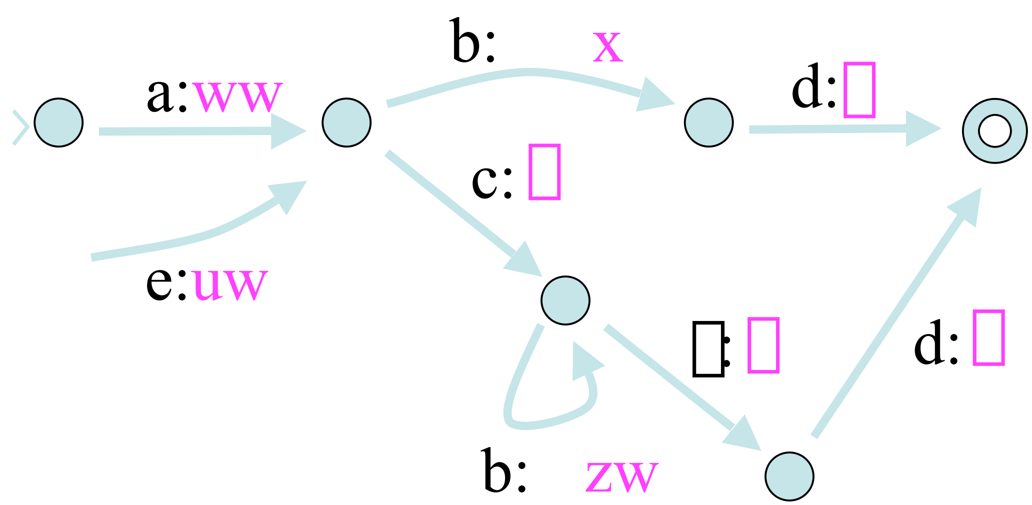
Shifting Outputs Along Paths (Mohri)

- Here, not all the out-arcs start with w
- But all the out-**paths** start with w
- Do pushback at later states first: now we're ok!



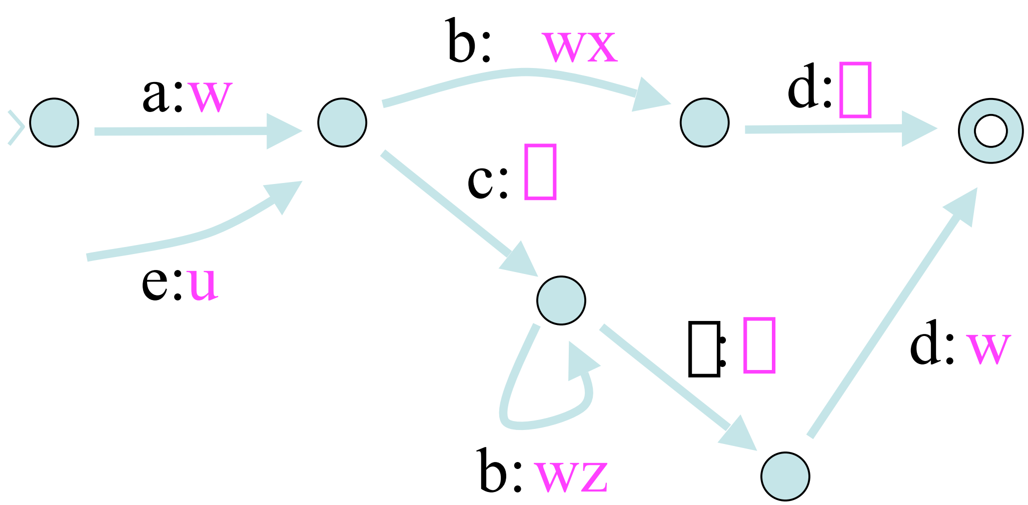
Shifting Outputs Along Paths (Mohri)

- Here, not all the out-arcs start with w
- But all the out-**paths** start with w
- Do pushback at later states first: now we're ok!



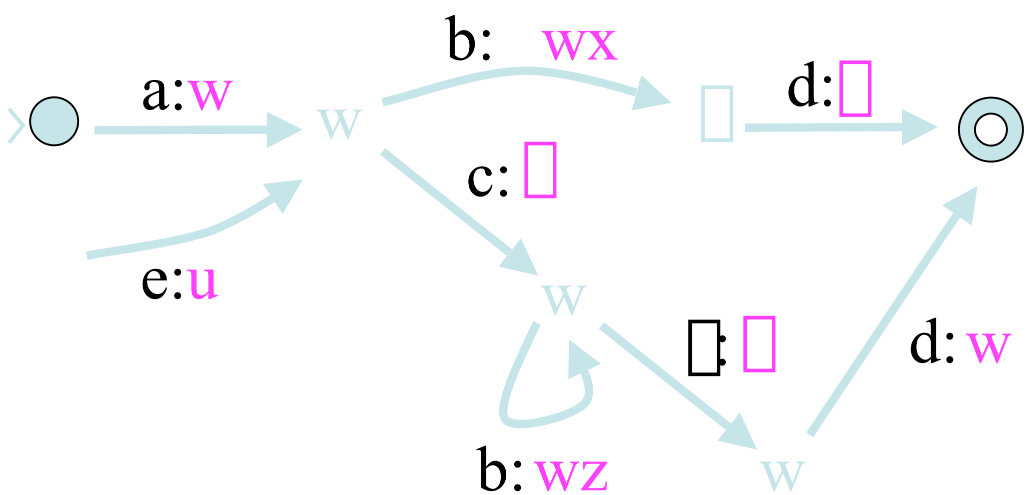
Shifting Outputs Along Paths (Mohri)

- Actually, push back at all states at once



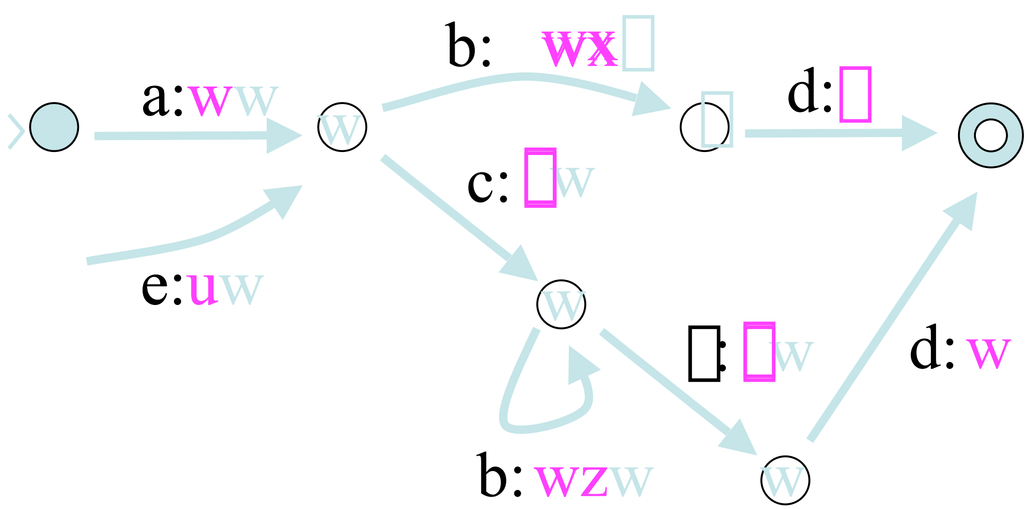
Shifting Outputs Along Paths (Mohri)

- Actually, push back at all states at once
- At every state q, compute some □(q)



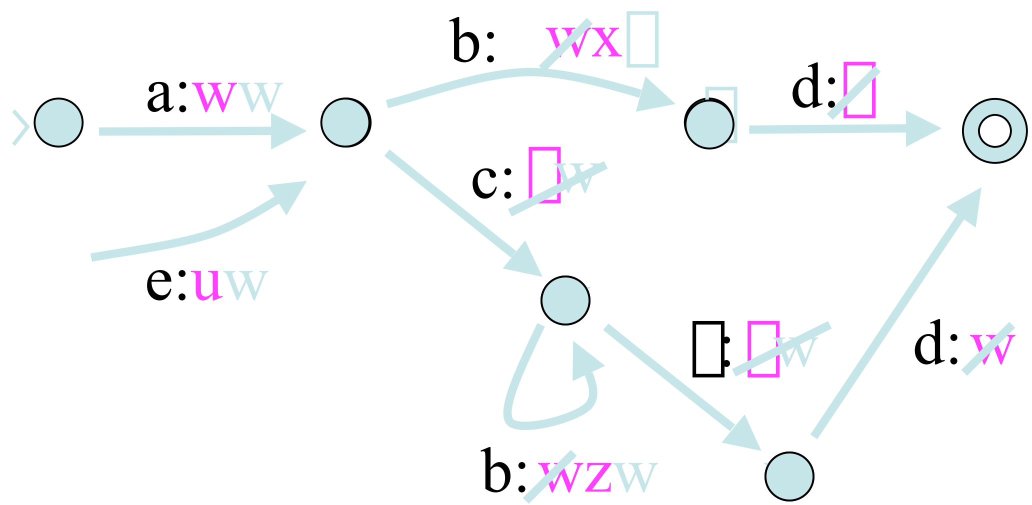
Shifting Outputs Along Paths (Mohri)

- Actually, push back at all states at once
- Add □(q) to end of q's in-arcs



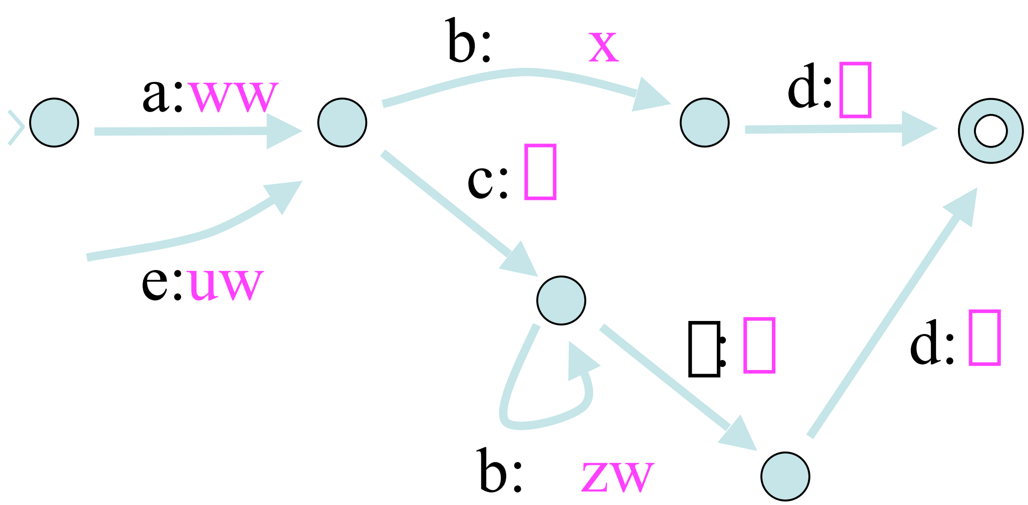
Shifting Outputs Along Paths (Mohri)

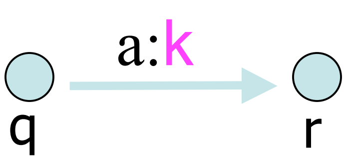
- Actually, push back at all states at once
- Add □(q) to end of q's in-arcs
- Remove □(q) from start of q's out-arcs



Shifting Outputs Along Paths (Mohri)

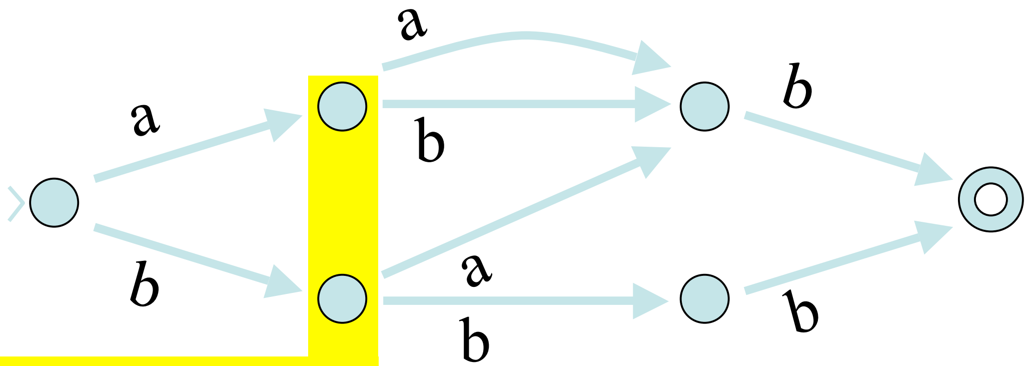
- Actually, push back at all states at once
- Add □(q) to end of q's in-arcs
- Remove □(q) from start of q's out-arcs





becomes

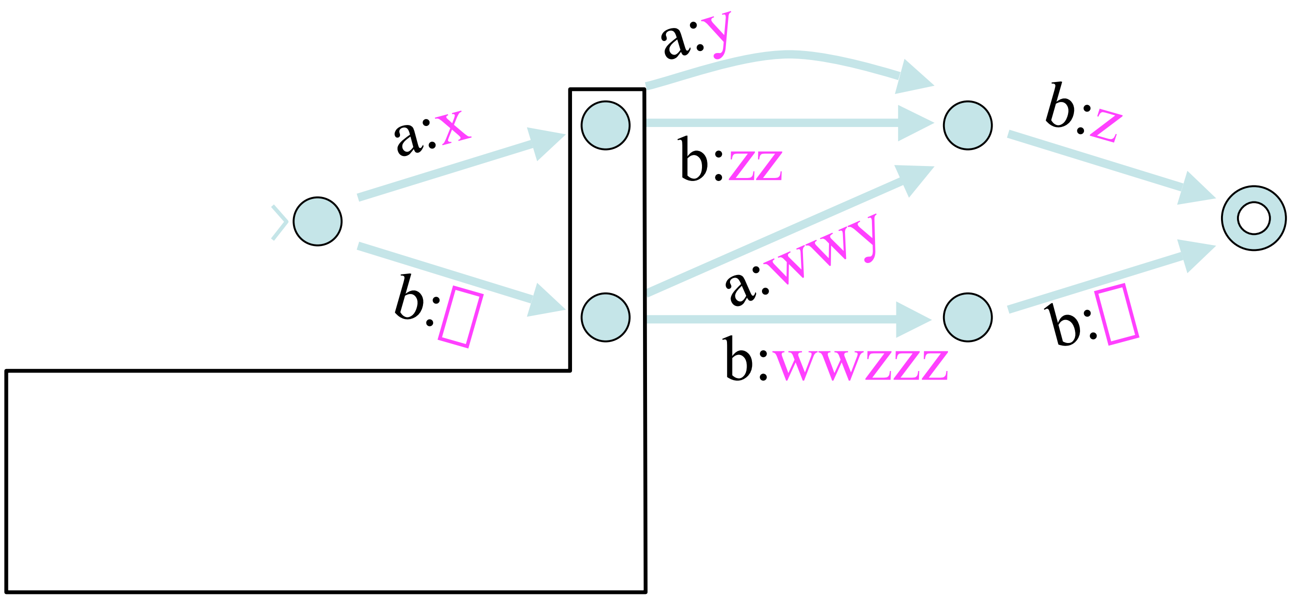
Minimizing Weighted DFAs (Mohri)



Mergeable because they accept the same **suffix language**: {ab,bb}

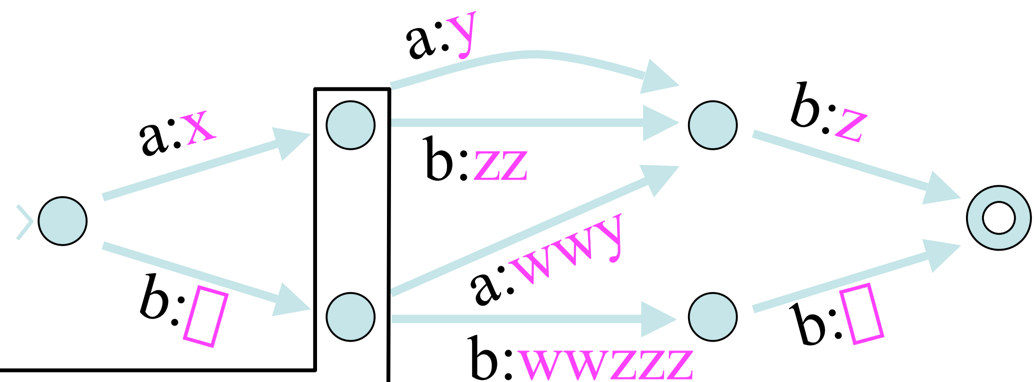
Minimizing Weighted DFAs (Mohri)

Still accept same suffix language,
  but produce different outputs on it



Minimizing Weighted DFAs (Mohri)

Still accept same suffix language,
 but produce different outputs on it



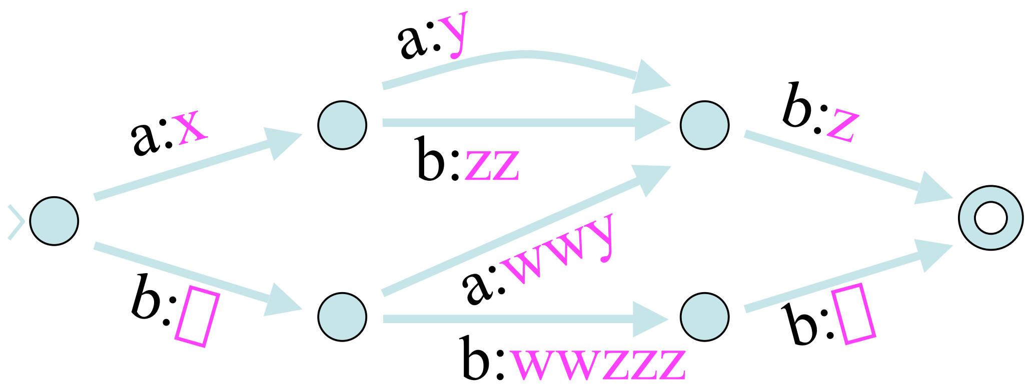
Not mergeable - compute different **suffix functions**:

ab □ yz or wwy

acd □ zzz or wwzzz

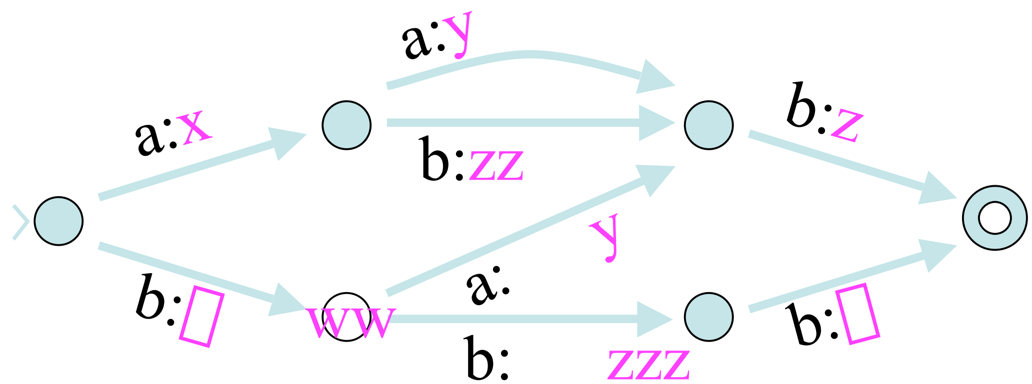
Minimizing Weighted DFAs (Mohri)

Fix by shifting outputs leftward …



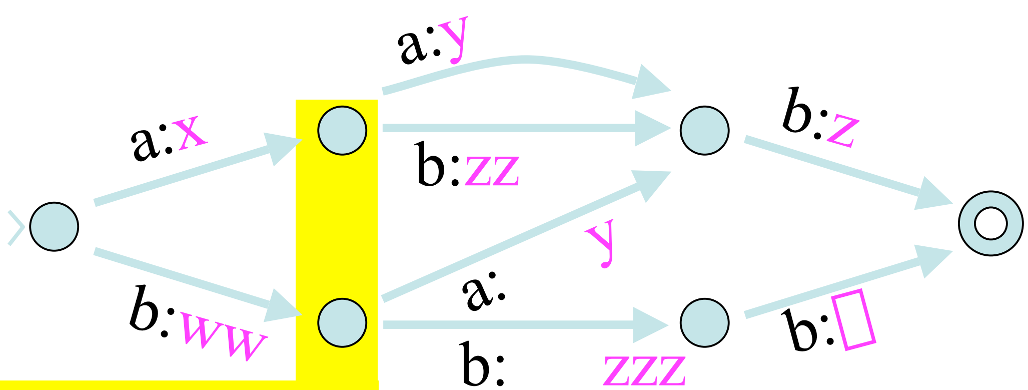
Minimizing Weighted DFAs (Mohri)

Fix by shifting outputs leftward …



Minimizing Weighted DFAs (Mohri)

Fix by shifting outputs leftward …



Now mergeable - they have the same **suffix function**:

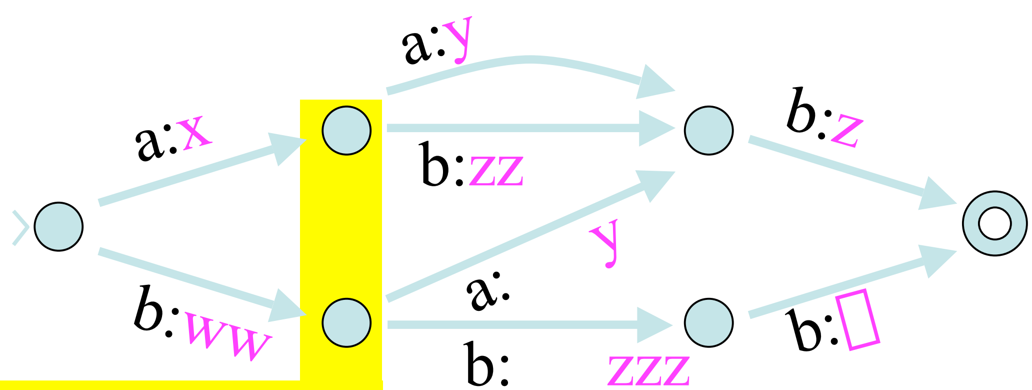
ab □ yz

acd □ zzz

But still no easy way to detect mergeability.

Minimizing Weighted DFAs (Mohri)

If we do this at all states as before …



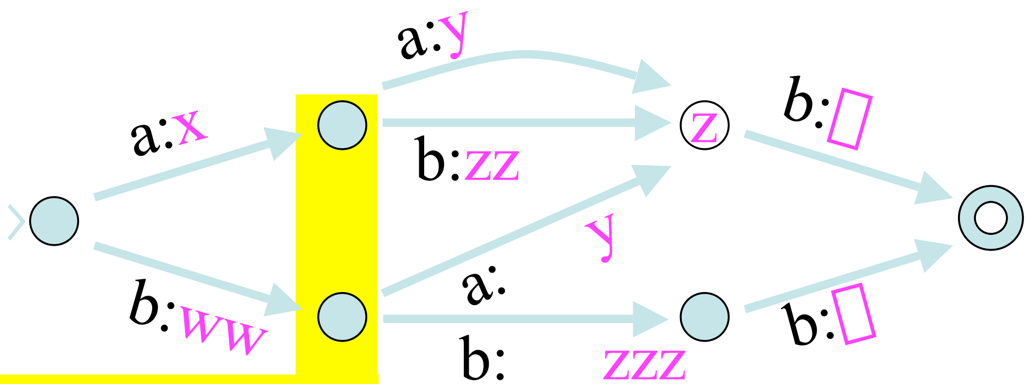
Now mergeable - they have
the same **suffix function**:

ab □ yz

acd □ zzz

Minimizing Weighted DFAs (Mohri)

If we do this at all states as before …



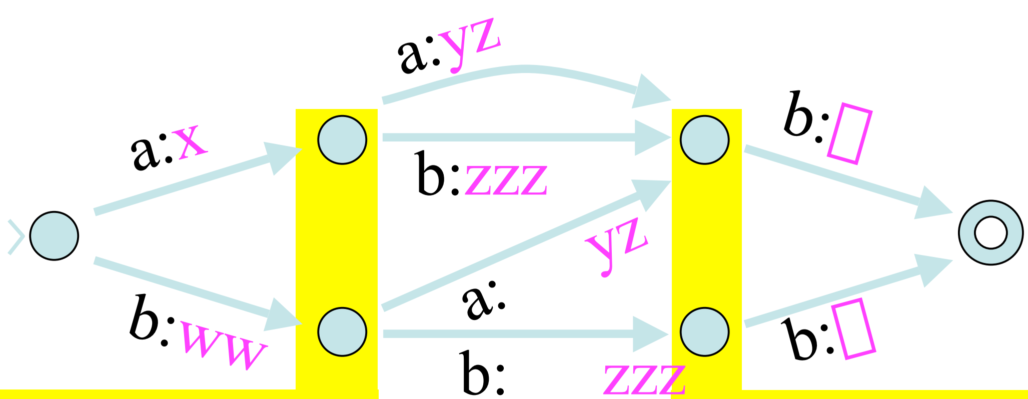
Now mergeable - they have the same **suffix function**:

ab → yz

acd → zzz

Minimizing Weighted DFAs (Mohri)

If we do this at all states as before …

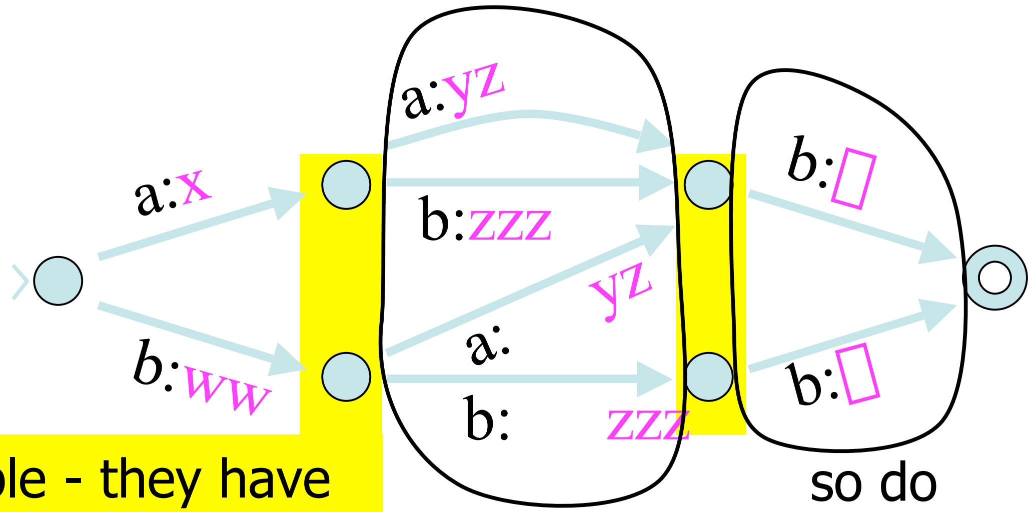


Now mergeable - they have the same **suffix function**:
ab □ yz
acd □ zzz

Now these have the same sufffix function too:
b □ □

Minimizing Weighted DFAs (Mohri)

Now we can discover & perform the merges:



Now mergeable - they have
the same **suffix function**:
ab □ yz
acd □ zzz

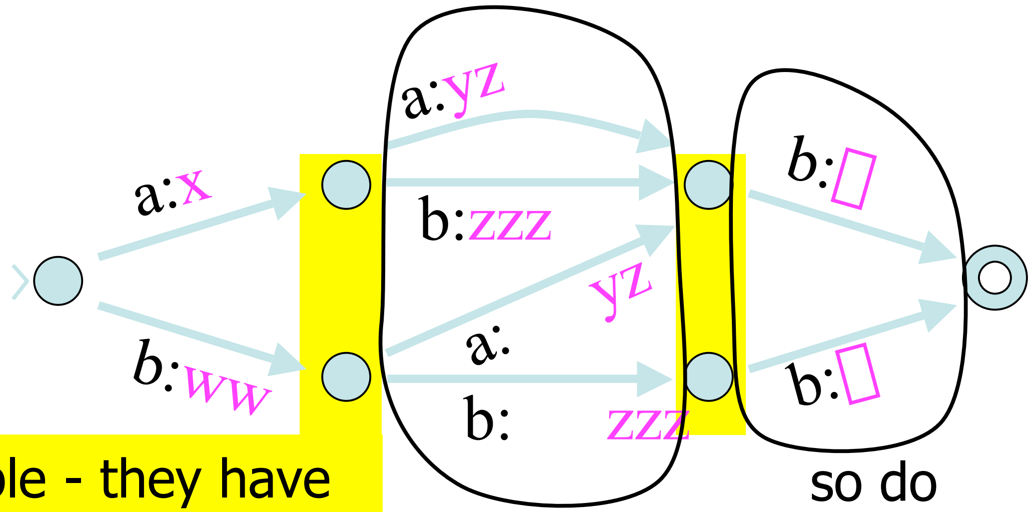
now these
have same
arc labels

so do
these

because we arranged for
a canonical placement of
outputs along paths

Minimizing Weighted DFAs (Mohri)

Treat each label "a:yz" as a single atomic symbol



Now mergeable - they have the same **suffix function**:
ab □ yz
acd □ zzz

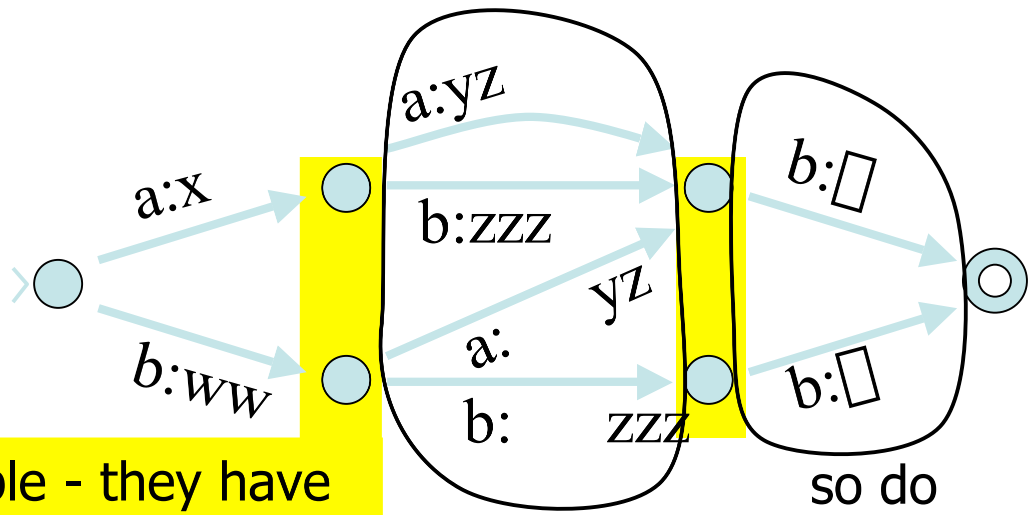
now these have same arc labels

so do these

because we arranged for a canonical placement of outputs along paths

Minimizing Weighted DFAs (Mohri)

Treat each label "a:yz" as a single atomic symbol



Now mergeable - they have the same **suffix function**:

ab □ yz

acd □ zzz

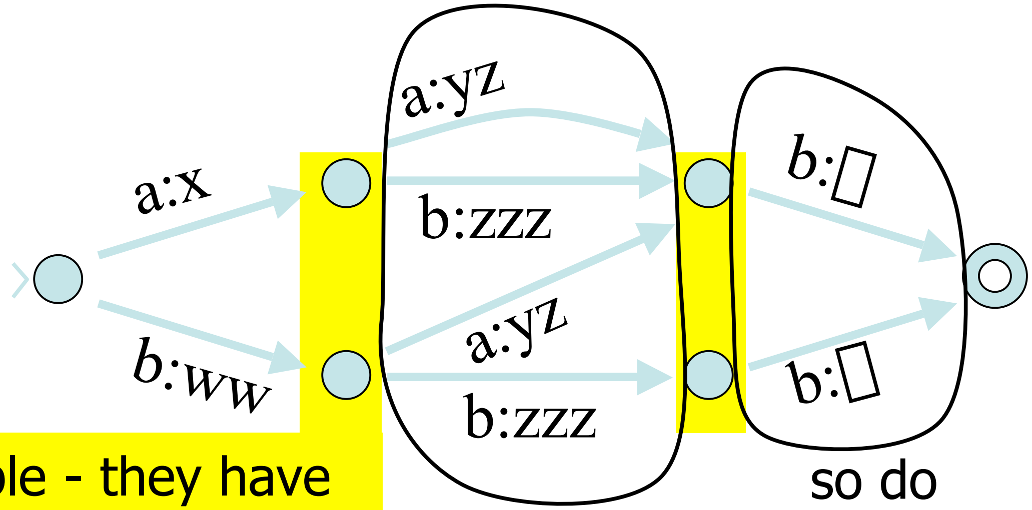
now these have same arc labels

so do these

because we arranged for a canonical placement of outputs along paths

Minimizing Weighted DFAs (Mohri)

Treat each label "a:yz" as a single atomic symbol



Now mergeable - they have the same **suffix function**:

ab □ yz

acd □ zzz

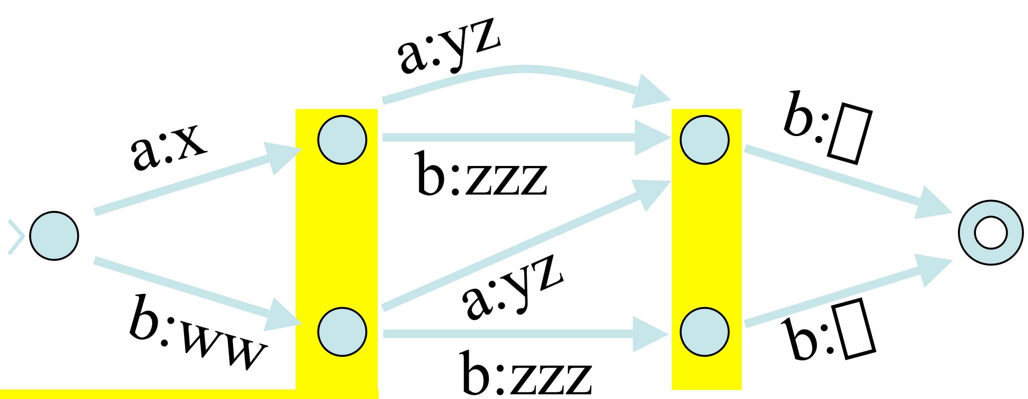
now these have same arc labels

so do these

because we arranged for a canonical placement of outputs along paths

Minimizing Weighted DFAs (Mohri)

Treat each label "a:yz" as a single atomic symbol
Use *unweighted* minimization algorithm!



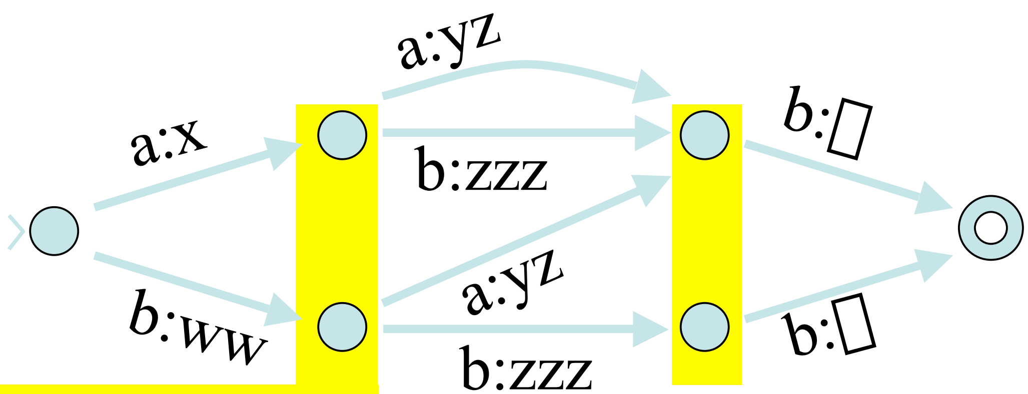
Now mergeable - they have the same **suffix function**:

ab → yz

acd → zzz

Minimizing Weighted DFAs (Mohri)

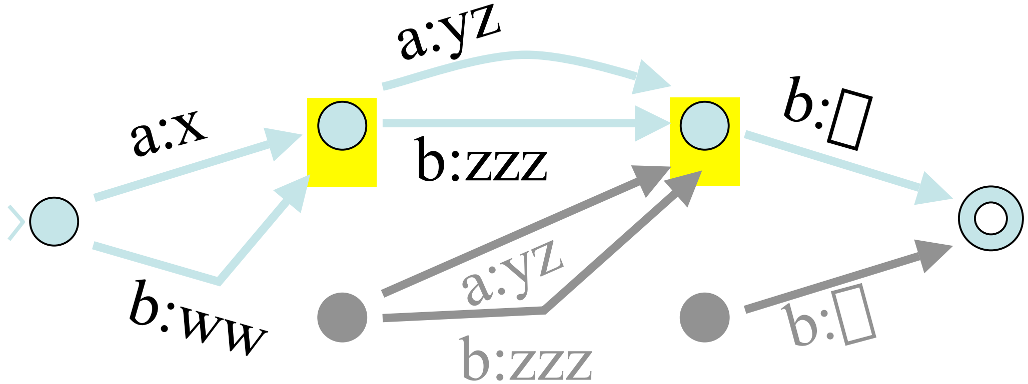
Treat each label "a:yz" as a single atomic symbol
Use *unweighted* minimization algorithm!



Now mergeable - they have the same **suffix language**:
{a:yz b:□,
b:zzz b:□}

Minimizing Weighted DFAs (Mohri)

Treat each label "a:yz" as a single atomic symbol
Use *unweighted* minimization algorithm!



Minimizing Weighted DFAs (Mohri)

Summary of weighted minimization algorithm:

1. **Compute** □(q) at each state q
2. **Push** each □(q) back through state q;
   this changes arc weights
3. **Merge** states via unweighted minimization

Step 3 merges states

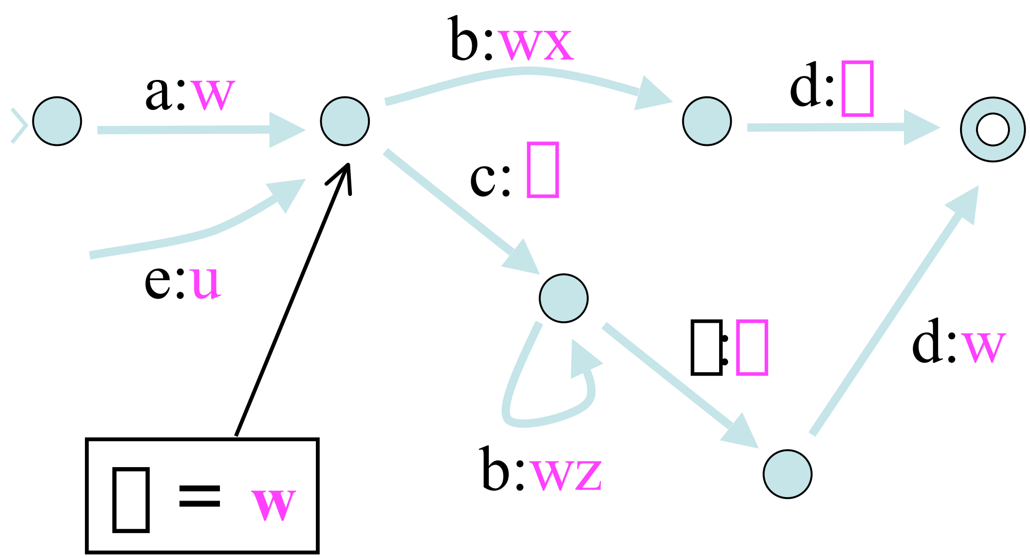
Step 2 allows more states to merge at step 3

Step 1 controls what step 2 does – preferably, to give states the same suffix function whenever possible

So define □(q) carefully at step 1!

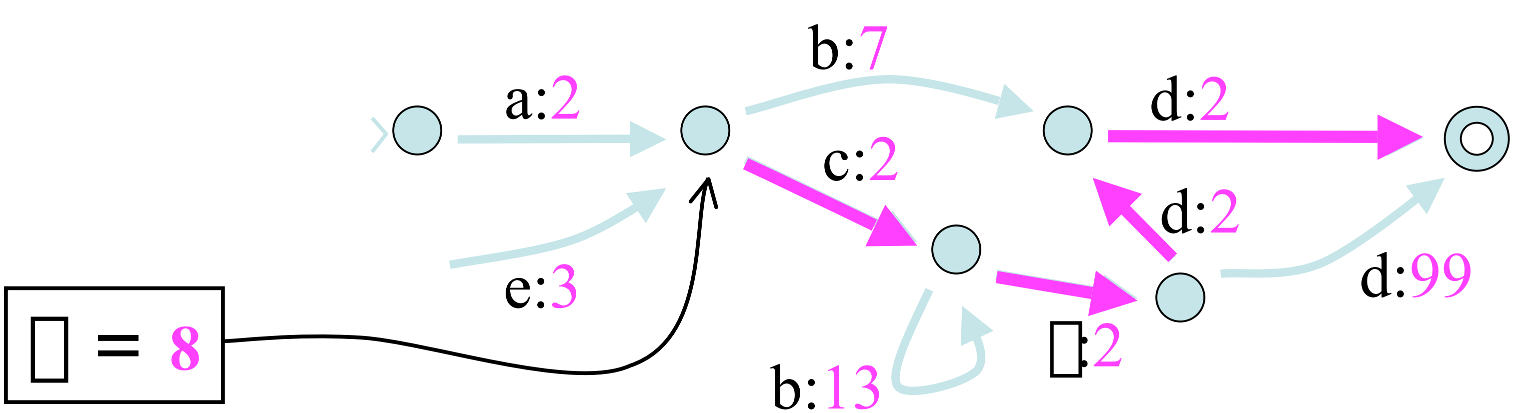
Mohri's Algorithms (1997, 2000)

- Mohri treated two versions of (K,)
- (K,) = (strings, concatenation)
  - □(q) = longest common prefix of all paths from q
  - Rather tricky to find



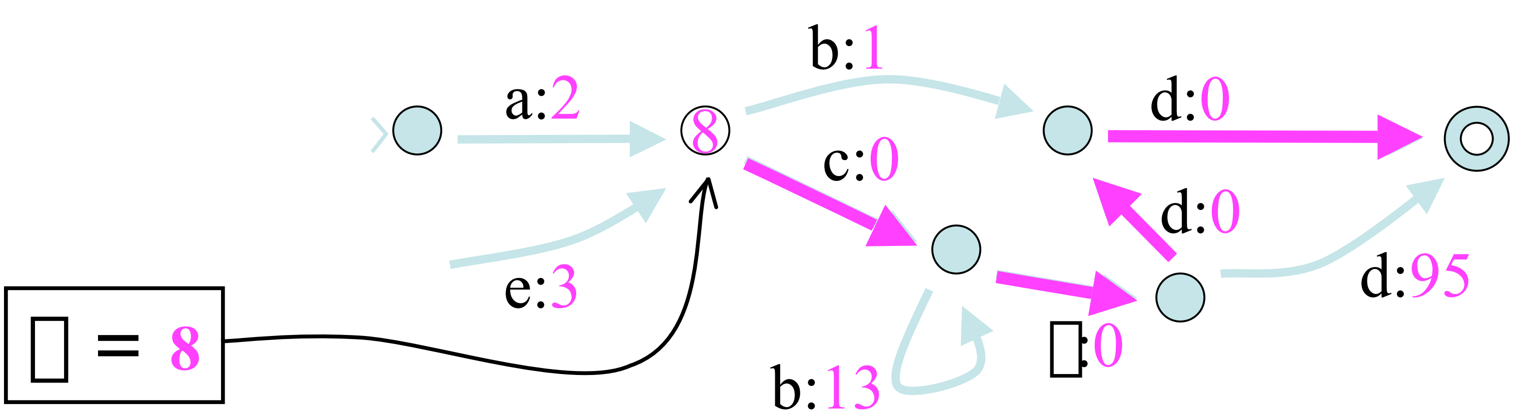
Mohri's Algorithms (1997, 2000)

- Mohri treated two versions of (K,)
- (K,) = (strings, concatenation)
  - □(q) = longest common prefix of all paths from q
  - Rather tricky to find
- (K,) = (nonnegative reals, addition)
  - □(q) = minimum weight of any path from q
  - Find it by Dijkstra's shortest-path algorithm



Mohri's Algorithms (1997, 2000)

- Mohri treated two versions of (K,)
- (K,) = (strings, concatenation)
  - □(q) = longest common prefix of all paths from q
  - Rather tricky to find
- (K,) = (nonnegative reals, addition)
  - □(q) = minimum weight of any path from q
  - Find it by Dijkstra's shortest-path algorithm



Mohri's Algorithms (1997, 2000)

- Mohri treated two versions of (K,)
- (K,) = (strings, concatenation)
  - □(q) = longest common prefix of all paths from q
  - Rather tricky to find
- (K,) = (nonnegative reals, addition)
  - □(q) = minimum weight of any path from q
  - Find it by Dijkstra's shortest-path algorithm