Simpler & More General Minimization for Weighted Finite-State Automata

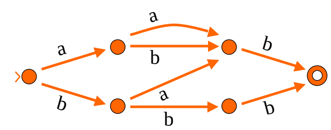
Jason Eisner
Johns Hopkins University

May 28, 2003 — HLT-NAACL

First half of talk is setup - reviews past work.
Second half gives outline of the new results.

The Minimization Problem

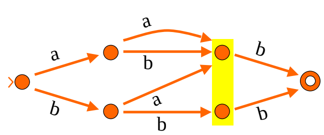
Input:	A DFA	(deterministic finite-state automaton)
Output:	An equiv. DFA with as few states as possible	
Complexity:	O(|arcs| log |states|)	(Hopcroft 1971)



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

The Minimization Problem

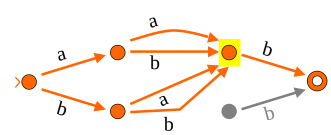
Input:	A DFA	(deterministic finite-state automaton)
Output:	An equiv. DFA with as few states as possible	
Complexity:	O(|arcs| log |states|)	(Hopcroft 1971)



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

The Minimization Problem

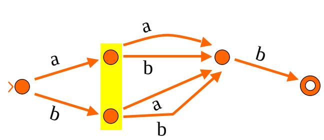
Input:	A DFA	(deterministic finite-state automaton)
Output:	An equiv. DFA with as few states as possible	
Complexity:	O(|arcs| log |states|)	(Hopcroft 1971)



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

The Minimization Problem

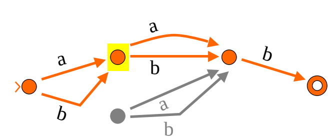
Input:	A DFA	(deterministic finite-state automaton)
Output:	An equiv. DFA with as few states as possible	
Complexity:	O(|arcs| log |states|)	(Hopcroft 1971)



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

The Minimization Problem

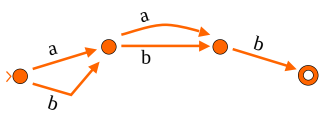
Input:	A DFA	(deterministic finite-state automaton)
Output:	An equiv. DFA with as few states as possible	
Complexity:	O(|arcs| log |states|)	(Hopcroft 1971)



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)

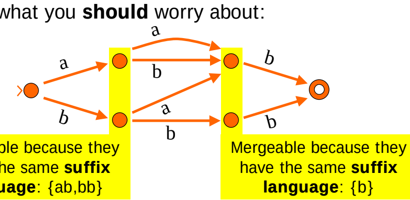


Can't always work backward from final state like this.
A bit more complicated because of cycles.
Don't worry about it for this talk.

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:



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)

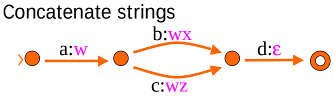
Q: Why minimize # states, rather than # arcs?
A: Minimizing # states also minimizes # arcs!

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. ☹

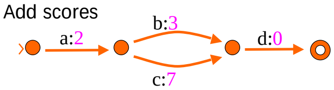
Real-World NLP: Automata With Weights or Outputs

- Finite-state computation of functions
  - Concatenate strings



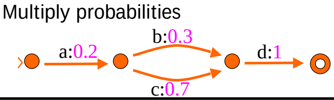
abd → wwx
acd → wwz

  - Add scores



abd → 5
acd → 9

  - Multiply probabilities



abd → 0.06
acd → 0.14

Real-World NLP: Automata With Weights or Outputs

- Want to compute functions on strings: $\Sigma^* \rightarrow K$
  - After all, we're doing language and speech!
- Finite-state machines can often do the job
- Easy to build, easy to combine, run fast
- Build them with weighted regular expressions
  - To clean up the resulting DFA, minimize it to merge redundant portions
  - This smaller machine is faster to intersect/compose
  - More likely to fit on a hand-held device
  - More likely to fit into cache memory

Real-World NLP: Automata With Weights or Outputs

- Want to compute functions on strings: $\Sigma^* \rightarrow K$
  - After all, we're doing language and speech!
- Finite-state machines can often do the job

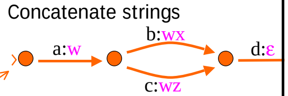
How do we minimize such DFAs?

- Didn't Mohri already answer this question?
- Only for special cases of the output set K!
- **Is there a general recipe?**
- **What new algorithms can we cook with it?**

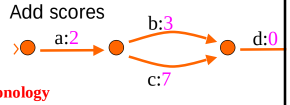
Weight Algebras

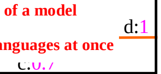
- Specify a weight algebra (K,⊗)
- Define DFAs over (K,⊗)
- Arcs have weights in set K
- A path's weight is also in K: multiply its arc weights with ⊗
- Examples:
  - (strings, concatenation)
  - (scores, addition)
  - (probabilities, multiplication)
  - (score vectors, addition) **OT phonology**
  - (real weights, multiplication) **conditional random fields, rational kernels**
  - (objective func & gradient, product-rule multiplication) **training the parameters of a model**
  - (bit vectors, conjunction) **membership in multiple languages at once**

- Finite-state computation of fu
  - Concatenate strings



  - Add scores



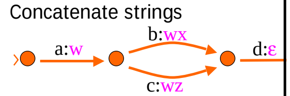


Weight Algebras

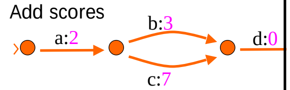
- Specify a weight algebra (K,⊗)
- Define DFAs over (K,⊗)
- Arcs have weights in set K
- A path's weight is also in K: multiply its arc weights with ⊗

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

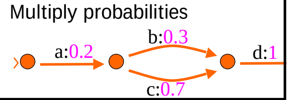
- Finite-state computation of fu
  - Concatenate strings



  - Add scores

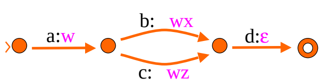


  - Multiply probabilities



Shifting Outputs Along Paths

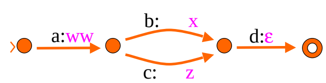
- Doesn't change the function computed:



abd → wwx
acd → wwz

Shifting Outputs Along Paths

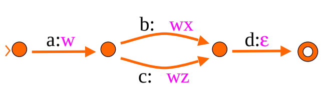
- Doesn't change the function computed:



abd → wwx
acd → wwz

Shifting Outputs Along Paths

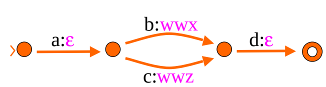
- Doesn't change the function computed:



abd → wwx
acd → wwz

Shifting Outputs Along Paths

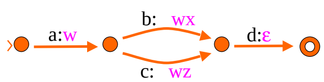
- Doesn't change the function computed:



abd → wwx
acd → wwz

Shifting Outputs Along Paths

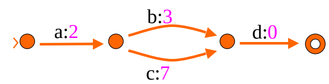
- Doesn't change the function computed:



abd → wwx
acd → wwz

Shifting Outputs Along Paths

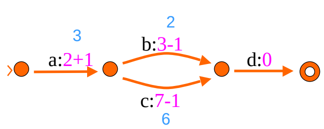
- Doesn't change the function computed:



abd → 5
acd → 9

Shifting Outputs Along Paths

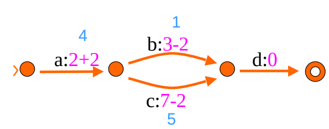
- Doesn't change the function computed:



abd → 5
acd → 9

Shifting Outputs Along Paths

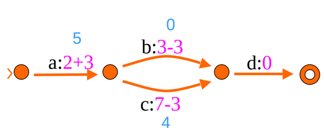
- Doesn't change the function computed:



abd → 5
acd → 9

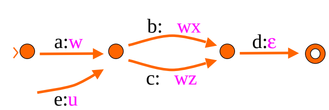
Shifting Outputs Along Paths

- Doesn't change the function computed:



abd → 5
acd → 9

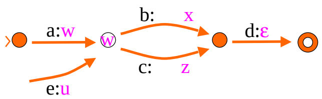
Shifting Outputs Along Paths



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

Shifting Outputs Along Paths

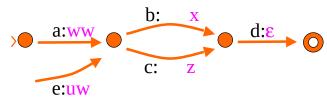
- State sucks back a prefix from its out-arcs



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

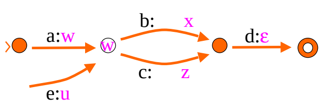
Shifting Outputs Along Paths

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



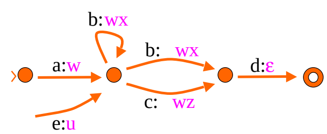
abd → wwx
acd → wwz
…ebd → uwx
…ecd → uwz

Shifting Outputs Along Paths



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

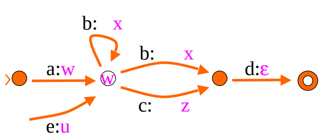
Shifting Outputs Along Paths



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

…$ab^nbd \to u(wx)^nwx$
…$ab^ncd \to u(wx)^nwz$

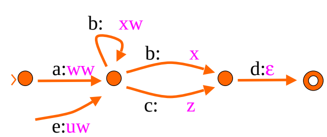
Shifting Outputs Along Paths



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

…$ab^nbd \to u(wx)^nwx$
…$ab^ncd \to u(wx)^nwz$

Shifting Outputs Along Paths

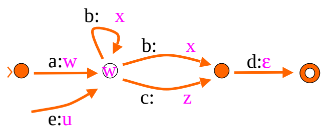


abd → wwx
acd → wwz
…ebd → uwx
…ecd → uwz

…$ab^nbd \to u(wx)^nwx$
…$ab^ncd \to u(wx)^nwz$

…$ab^nbd \to uw(xw)^nx$
…$ab^ncd \to uw(xw)^nz$

Shifting Outputs Along Paths

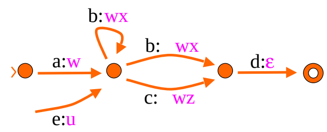


abd → wwx
acd → wwz
…ebd → uwx
…ecd → uwz

…ab^nbd → $u(wx)^nwx$
…ab^ncd → $u(wx)^nwz$

…ab^nbd → $uw(xw)^nx$
…ab^ncd → $uw(xw)^nz$

Shifting Outputs Along Paths



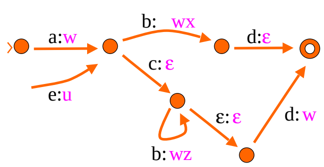
abd → wwx
acd → wwz
…ebd → uwx
…ecd → uwz

…ab^nbd → $u(wx)^nwx$
…ab^ncd → $u(wx)^nwz$

…ab^nbd → $uw(xw)^nx$
…ab^ncd → $uw(xw)^nz$

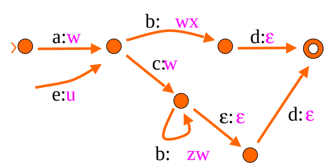
Shifting Outputs Along Paths (Mohri)

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



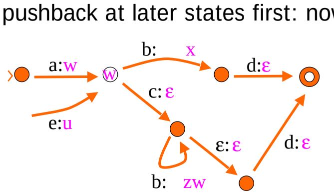
Shifting Outputs Along Paths (Mohri)

- 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!



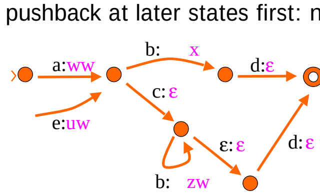
Shifting Outputs Along Paths (Mohri)

- 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!



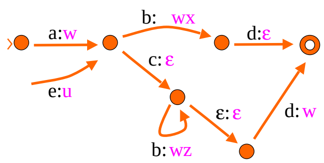
Shifting Outputs Along Paths (Mohri)

- 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!



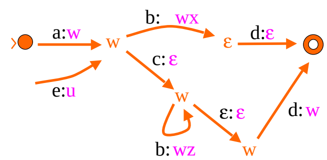
Shifting Outputs Along Paths (Mohri)

- Actually, push back at all states at once



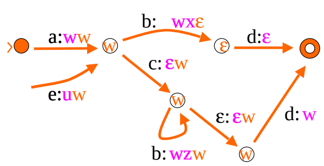
Shifting Outputs Along Paths (Mohri)

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



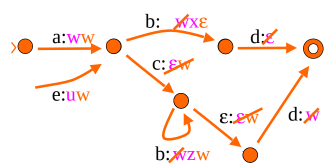
Shifting Outputs Along Paths (Mohri)

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



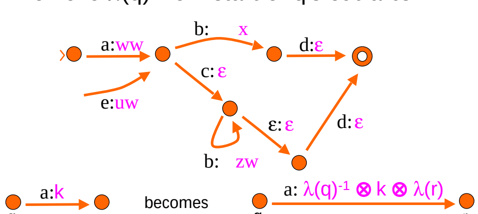
Shifting Outputs Along Paths (Mohri)

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

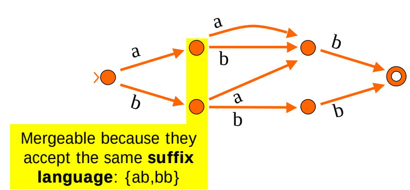


Shifting Outputs Along Paths (Mohri)

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

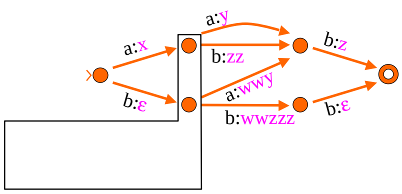


Minimizing Weighted DFAs (Mohri)



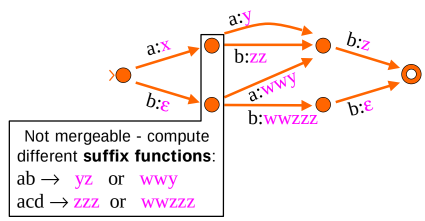
Minimizing Weighted DFAs (Mohri)

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



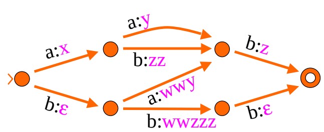
Minimizing Weighted DFAs (Mohri)

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



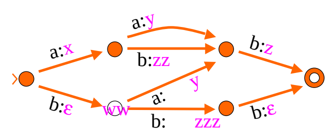
Minimizing Weighted DFAs (Mohri)

Fix by shifting outputs leftward ...



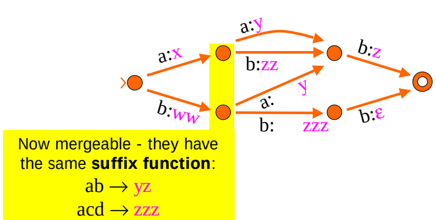
Minimizing Weighted DFAs (Mohri)

Fix by shifting outputs leftward ...



Minimizing Weighted DFAs (Mohri)

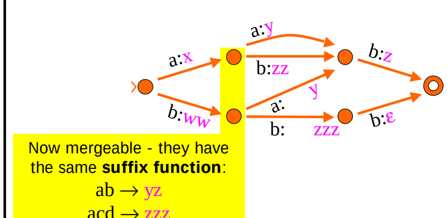
Fix by shifting outputs leftward ...



But still no easy way to detect mergeability.

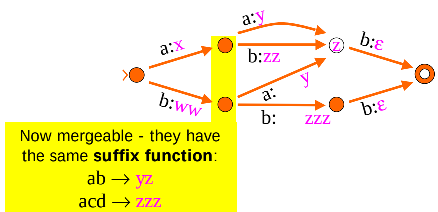
Minimizing Weighted DFAs (Mohri)

If we do this at all states as before ...



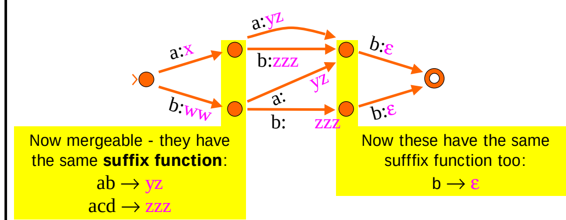
Minimizing Weighted DFAs (Mohri)

If we do this at all states as before ...



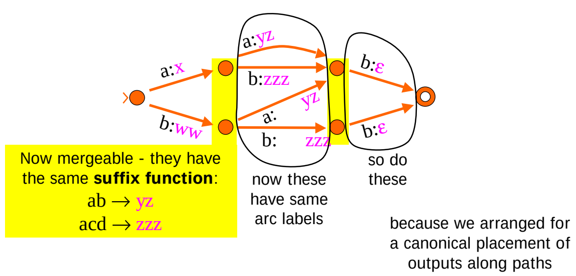
Minimizing Weighted DFAs (Mohri)

If we do this at all states as before ...



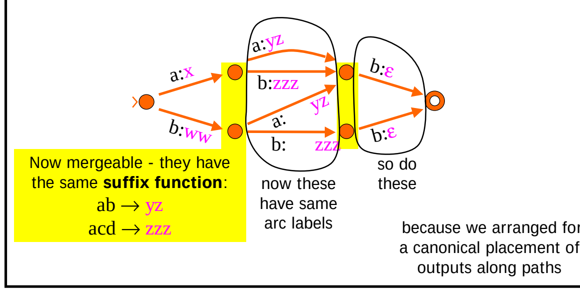
Minimizing Weighted DFAs (Mohri)

Now we can discover & perform the merges:



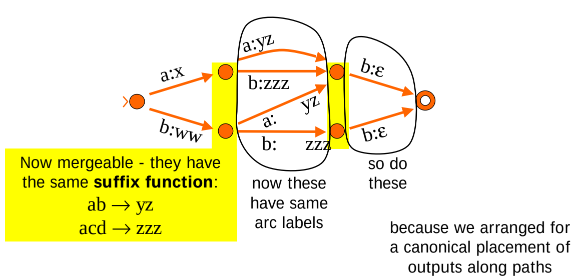
Minimizing Weighted DFAs (Mohri)

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



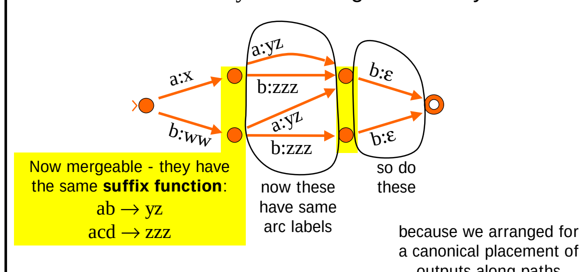
Minimizing Weighted DFAs (Mohri)

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



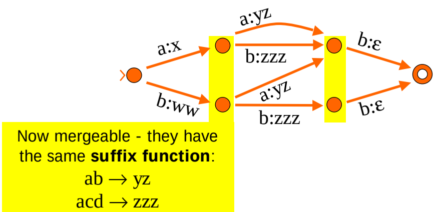
Minimizing Weighted DFAs (Mohri)

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



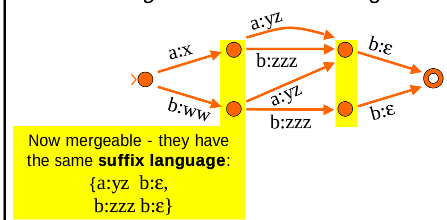
Minimizing Weighted DFAs (Mohri)

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



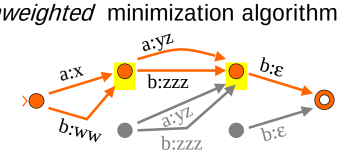
Minimizing Weighted DFAs (Mohri)

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



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

Step 3 merges states

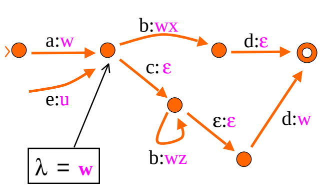
Step 2 allows more states to merge at step 3

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

So define λ(q) carefully at step 1!

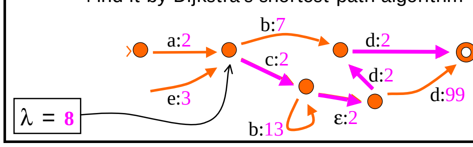
Mohri's Algorithms (1997, 2000)

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



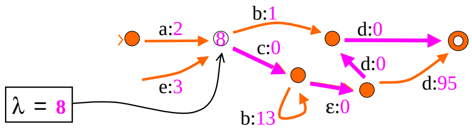
Mohri's Algorithms (1997, 2000)

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



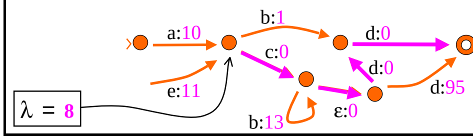
Mohri's Algorithms (1997, 2000)

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



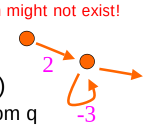
Mohri's Algorithms (1997, 2000)

- Mohri treated two versions of (K,⊗)
- (K,⊗) = (strings, concatenation)
  - λ(q) = longest common prefix of all paths from q
  - Rather tricky to find
- (K,⊗) = (nonnegative reals, addition)
  - λ(q) = minimum weight of any path from q
  - Find it by Dijkstra's shortest-path algorithm
- In both cases:
  - λ(q) = a "sum" over infinite set of path weights
  - must define this "sum" and an algorithm to compute it
  - doesn't generalize automatically to other (K,⊗) …

Mohri's Algorithms (1997, 2000)

(real weights, multiplication)?
(score vectors, addition)?
(objective func & gradient,
product-rule multiplication)?

e.g., what if we allowed negative reals?
Then minimum might not exist!



- (K,⊗) = (nonnegative reals, addition)
  - λ(q) = minimum weight of any path from q
  - Find it by Dijkstra's algorithm
- In both cases:
  - λ(q) = a "sum" over infinite set of path weights
  - must define this "sum" and an algorithm to compute it
  - doesn't generalize automatically to other (K,⊗) …

End of background material.
Now we can sketch the new results!

Want to minimize DFAs in any (K,⊗)

Generalizing the Strategy

- Given (K,⊗)
- Just need a definition of λ … then use general alg.
- λ should extract an appropriate "left factor" from state q's suffix function F_q: $\Sigma^* \rightarrow K$

  Remember, F_q is the function that the automaton would compute if state q were the start state

- What properties must λ have to guarantee that we get the minimum equivalent machine?

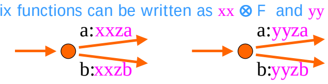
Generalizing the Strategy

- What properties must the λ function have?
- For all $F: \Sigma^* \to K$, $k \in K$, $a \in \Sigma$:
  - **Shifting:** $\lambda(k \otimes F) = k \otimes \lambda(F)$
  - **Quotient:** $\lambda(F)$ is a left factor of $\lambda(a^{-1}F)$
  - **Final-quotient:** $\lambda(F)$ is a left factor of $F(\varepsilon)$
- Then pushing + merging is guaranteed to minimize the machine.

Generalizing the Strategy

- What properties must the λ function have?
- For all $F: \Sigma^* \to K$, $k \in K$, $a \in \Sigma$:
  - **Shifting:** $\lambda(k \otimes F) = k \otimes \lambda(F)$

Suffix functions can be written as xx $\otimes$ F and yy $\otimes$ F:

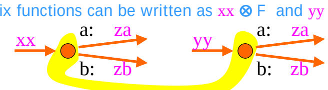


Shifting property says:
When we remove the prefixes λ(xx $\otimes$ F) and λ(yy $\otimes$ F)
we will remove xx and yy respectively

Generalizing the Strategy

- What properties must the λ function have?
- For all $F: \Sigma^* \to K$, $k \in K$, $a \in \Sigma$:
  - **Shifting:** $\lambda(k \otimes F) = k \otimes \lambda(F)$

Suffix functions can be written as xx $\otimes$ F and yy $\otimes$ F:



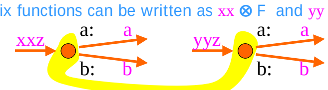
Shifting property says:
When we remove the prefixes λ(xx $\otimes$ F) and λ(yy $\otimes$ F)
we will remove xx and yy respectively
leaving behind a common residue.

Actually, remove xx $\otimes$ λ(F) and yy $\otimes$ λ(F).

Generalizing the Strategy

- What properties must the λ function have?
- For all $F: \Sigma^* \to K$, $k \in K$, $a \in \Sigma$:
  - **Shifting:** $\lambda(k \otimes F) = k \otimes \lambda(F)$

Suffix functions can be written as xx $\otimes$ F and yy $\otimes$ F:

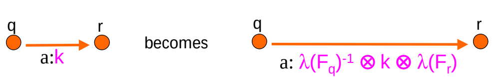


Shifting property says:
When we remove the prefixes λ(xx $\otimes$ F) and λ(yy $\otimes$ F)
we will remove xx and yy respectively
leaving behind a common residue.

Actually, remove xx $\otimes$ λ(F) and yy $\otimes$ λ(F).

Generalizing the Strategy

- What properties must the λ function have?
- For all $F: \Sigma^* \to K$, $k \in K$, $a \in \Sigma$:
  - **Shifting:** $\lambda(k \otimes F) = k \otimes \lambda(F)$
  - **Quotient:** $\lambda(F)$ is a left factor of $\lambda(a^{-1}F)$



$= \lambda(F_q)^{-1} \otimes \lambda(k \otimes F_r)$
$= \lambda(F_q)^{-1} \otimes \lambda(a^{-1}F_q)$

Quotient property says that this quotient exists even if $\lambda(F_q)$ doesn't have a multiplicative inverse.

Generalizing the Strategy

- What properties must the λ function have?
- For all $F: \Sigma^* \to K$, $k \in K$, $a \in \Sigma$:
  - **Shifting:** $\lambda(k \otimes F) = k \otimes \lambda(F)$
  - **Quotient:** $\lambda(F)$ is a left factor of $\lambda(a^{-1}F)$
  - **Final-quotient:** $\lambda(F)$ is a left factor of $F(\varepsilon)$

Guarantees we can find final-state stopping weights.
If we didn't have this base case, we couldn't prove:
$\lambda(F)$ is a left factor of every output in range(F).

Then pushing + merging is guaranteed to minimize.

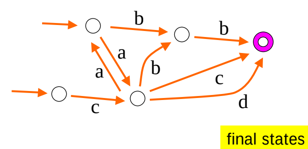
A New Specific Algorithm

- Mohri's algorithms instantiate this strategy.
- They use particular definitions of λ.
  - λ(q) = longest common string prefix of <u>all paths</u> from q
  - λ(q) = minimum numeric weight of <u>all paths</u> from q
  - interpreted as infinite sums over path weights; ignore input symbols
  - dividing by λ makes suffix func canonical: path weights sum to <u>1</u>

- Now for a new definition of λ !
  - λ(q) = weight of the <u>shortest</u> path from q,
    breaking ties lexicographically by input string
  - choose just **one** path, based **only** on its input symbols;
    computation is simple, well-defined, independent of (K, ⊗)
  - dividing by λ makes suffix func canonical: shortest path has weight <u>1</u>

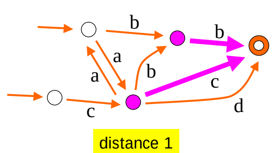
A New Specific Algorithm

- New definition of λ :
  - λ(q) = weight of the <u>shortest</u> path from q,
    breaking ties lexicographically by input string
- Computation is simple, well-defined, independent of (K, ⊗)
- **Breadth-first search back from final states:**



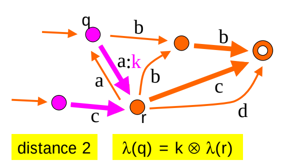
A New Specific Algorithm

- New definition of λ :
  - λ(q) = weight of the <u>shortest</u> path from q,
    breaking ties lexicographically by input string
- Computation is simple, well-defined, independent of (K, ⊗)
- **Breadth-first search back from final states:**



A New Specific Algorithm

- New definition of λ :
  - λ(q) = weight of the <u>shortest</u> path from q,
    breaking ties alphabetically on input symbols
- Computation is simple, well-defined, independent of (K, ⊗)
- **Breadth-first search back from final states:**



Compute λ(q)
in O(1) time
as soon as we visit q.
Whole alg. is linear.

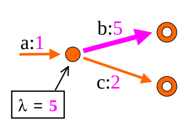
Faster than finding
min-weight path
à la Mohri.

Requires Multiplicative Inverses

- Does this definition of λ have the necessary properties?
  - λ(q) = weight of the <u>shortest</u> path from q,
    breaking ties alphabetically on input symbols
- If we regard λ as applying to suffix functions:
  - λ(F) = F(min domain(F)) with appropriate defn of "min"
- **Shifting:** λ(k ⊗ F) = k ⊗ λ(F)
  - Trivially true
- **Quotient:** λ(F) is a left factor of λ(a^{-1}F)
- **Final-quotient:** λ(F) is a left factor of F(ε)
  - These are true provided that (K,⊗) contains multiplicative inverses.
  - i.e., okay if (K,⊗) is a semigroup; (K,⊕,⊗) is a division semiring.

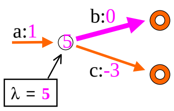
Requires Multiplicative Inverses

- So (K,⊗) must contain multiplicative inverses (under ⊗).
- Consider (K,⊗) = (nonnegative reals, addition):



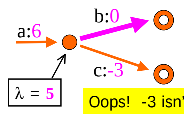
Requires Multiplicative Inverses

- So (K,⊗) must contain multiplicative inverses (under ⊗).
- Consider (K,⊗) = (nonnegative reals, addition):



Requires Multiplicative Inverses

- So (K,⊗) must contain multiplicative inverses (under ⊗).
- Consider (K,⊗) = (~~nonnegative~~ reals, addition):

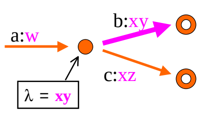


Oops! -3 isn't a legal weight.

Need to say (K,⊗) = (reals, addition).
Then subtraction always gives an answer.
Unlike Mohri, we might get negative weights in the output DFA ...
But unlike Mohri, we can handle negative weights in the input DFA
(including negative weight cycles!).

Requires Multiplicative Inverses

- How about transducers?
- (K,⊗) = (strings, concatenation)
- Must add multiplicative inverses, via inverse letters.

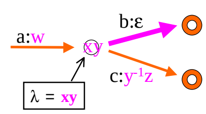


ab → wxy

ac → wxz

Requires Multiplicative Inverses

- How about transducers?
- (K,⊗) = (strings, concatenation)
- Must add multiplicative inverses, via inverse letters.

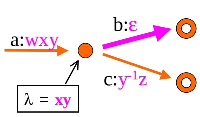


ab → wxy

ac → wxz

Requires Multiplicative Inverses

- How about transducers?
- (K,⊗) = (strings, concatenation)
- Must add multiplicative inverses, via inverse letters.



ab → wxy

ac → wxz

- Can actually make this work, though ⊗ no longer O(1)
  - Still arguably simpler than Mohri
  - But this time we're a bit slower in worst case, not faster as before
- Can eliminate inverse letters *after* we minimize

Real Benefit – Other Semirings!

- Other (K,⊗) of current interest do have mult inverses ...
- So we now have an easy minimization algorithm for them.
- No algorithm existed before.

(real weights, multiplication)?	**conditional random fields, rational kernels** (Lafferty/McCallum/Pereira; Cortes/Haffner/Mohri)
(score vectors, addition)?	**OT phonology** (Ellison)
(objective func & gradient, product-rule multiplication)?	**training the parameters of a model** (Eisner – expectation semirings)

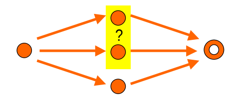
Back to the General Strategy

- What properties must the λ function have?
- For all $F: \Sigma^* \to K$, $k \in K$, $a \in \Sigma$:
  - **Shifting:** $\lambda(k \otimes F) = k \otimes \lambda(F)$
  - **Quotient:** $\lambda(F)$ is a left factor of $\lambda(a^{-1}F)$
  - **Final-quotient:** $\lambda(F)$ is a left factor of $F(\varepsilon)$
- New algorithm and Mohri's algs are special cases

> - What if we don't have mult. inverses?
> - Does this strategy work in every $(K,\otimes)$?
> - Does an appropriate λ always exist?
> - No! No strategy always works.
> - Minimization isn't always well-defined!

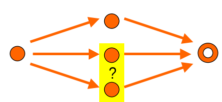
Minimization Not Unique

- In previously studied cases, all minimum-state machines equivalent to a given DFA were essentially the same.
- But the paper gives several $(K,\otimes)$ where this is not true!



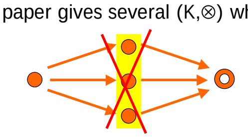
Minimization Not Unique

- In previously studied cases, all minimum-state machines equivalent to a given DFA were essentially the same.
- But the paper gives several $(K,\otimes)$ where this is not true!



Minimization Not Unique

- In previously studied cases, all minimum-state machines equivalent to a given DFA were essentially the same.
- But the paper gives several $(K,\otimes)$ where this is not true!



- Mergeability may not be an equivalence relation on states.
- "Having a common residue" may not be an equivalence relation on suffix functions.
- Has to do with the uniqueness of prime factorization in $(K,\otimes)$.
  - (But had to generalize notion so didn't assume $\otimes$ was commutative.)
  - Paper gives necessary and sufficient conditions …

Non-Unique Minimization Is Hard

- Minimum-state automaton isn't always unique.
- But can we find **one** that has min # of states?
- No: unfortunately NP-complete.
  - (reduction from Minimum Clique Partition)
- Can we get close to the minimum?
  - No: Min Clique Partition is inapproximable in polytime to within any constant factor (unless P=NP).
  - So we can't even be sure of getting within a factor of 100 of the smallest possible.

Summary of Results

- Some weight semirings are "bad":
  - Don't let us minimize uniquely, efficiently, or approximately [even in (bit vectors, conjunction)]
- Characterization of "good" weight semirings
- General minimization strategy for "good" semirings
  - Find a λ … Mohri's algorithms are special cases
- Easy minimization algorithm for division semirings
  - For additive weights, simpler & faster than Mohri's
  - Can apply to transducers, with "inverse letters" trick
  - Applies in the other semirings of present interest
    - fancy machine learning; parameter training; optimality theory

FIN

- New definition of λ :
  - λ(q) = weight of the shortest path from q, breaking ties alphabetically on input symbols

Ranking of accepting paths by input string:

ε < b < bb < aab < aba < abb

"geneaological order on strings"

we pick the minimum string accepted from state q