

CMPSCI 250: Introduction to Computation

Lecture #36: State Elimination
David Mix Barrington
23 April 2014

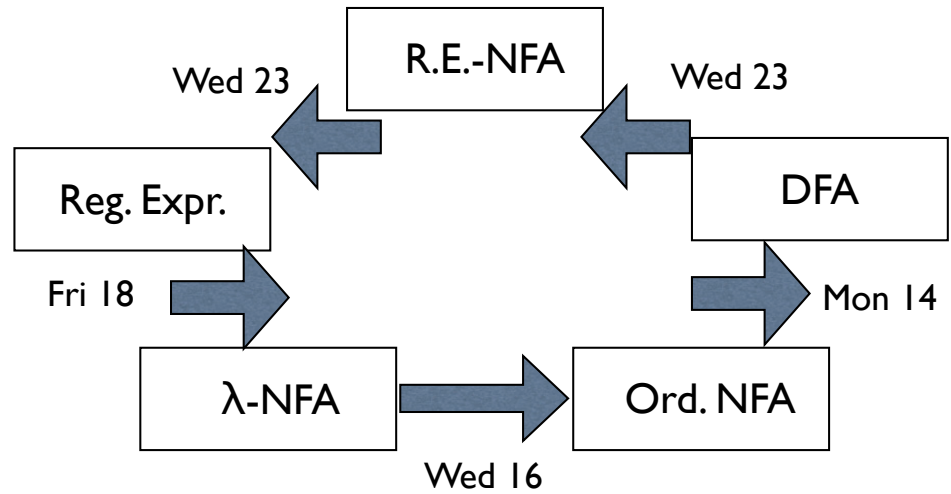
State Elimination

- Kleene's Theorem Overview
- Another New Model: The r.e.-NFA
- Overview of the Construction
- Eliminating a State
- Example: The Language EE
- Example: The Language No-aba
- Example: Number of a's Divisible by 3

Kleene's Theorem Overview

- We are finally ready to finish Kleene's Theorem, proving that a language has a regular expression if and only if it has a DFA.
- We have shown how to take a regular expression, produce a λ -NFA from it by the recursive construction, kill the λ -moves to get an ordinary NFA, use the Subset Construction to get a DFA, and then (if we want) minimize that DFA.

Kleene's Theorem Chart



Final Step of Kleene's Theorem

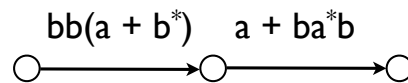
- The remaining step is to take a DFA and produce a regular expression for its language.
- As it turns out, the State Elimination Construction works equally well to get a regular expression for the language of any ordinary NFA or λ -NFA as well.

Final Step of Kleene's Theorem

- While the first two steps of converting a regular expression to a DFA roughly preserve the size, the Subset Construction in general takes an NFA with k states to a DFA with 2^k states.
- Though we won't prove this, State Elimination can also cause a large blowup, creating a long regular expression from a small DFA.
- (Excursion 14.11 in the text takes a closer look at this.)

Another Model: The R.E.-NFA

- The State Elimination Construction operates on yet another kind of NFA, which we will call an r.e.-NFA because the labels on its moves can be arbitrary regular expressions instead of just letters (as in an ordinary NFA) or either letters or λ (as in a λ -NFA).



Normal Form for R.E.-NFA's

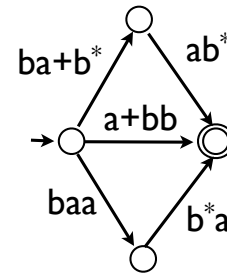
- Not every diagram with regular expressions on its edges is an r.e.-NFA -- we need to satisfy some rules.
- The first three are the same as the rules in our construction of λ -NFA's from regular expressions:
 - (1) Exactly one final state, not equal to the start state,
 - (2) No moves into the start state, and
 - (3) No moves out of the final state.

Normal Form for R.E.-NFA's

- The last rule is new: (4) no **parallel edges**, that is, no two edges with the same start node and end node.
- We have to redefine the Δ^* relation. We still have $\forall s: \Delta^*(s, \lambda, s)$, but now we have the rule $[\Delta^*(s, v, u) \wedge \Delta(u, R, t) \wedge (w \in L(R))] \rightarrow \Delta^*(s, vw, t)$.
- This rule isn't very useful for computing, as we have no equivalent top-down form for it.

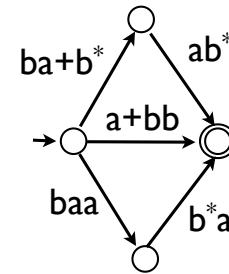
Clicker Question #1

- Which of these strings is *not* in the language of the r.e.-NFA pictured at right?
- (a) bbbba
- (b) babba
- (c) abbbb
- (d) bbabb



Answer #1

- Which of these strings is *not* in the language of the r.e.-NFA pictured at right?
- (a) bbbba
- (b) *babba*
- (c) abbbb
- (d) bbabb



Overview of the Construction

- The basic idea is to take our original DFA (or NFA, or λ -NFA), modify it so that it obeys the r.e.-NFA rules but still has the same language (how?) and then **eliminate states** one by one until there are only two left.
- Each elimination will preserve the language of the automaton and ensure that the r.e.-NFA rules still hold.

Overview of the Construction

- An r.e.-NFA with two states must have one of them as the start state and the other as the only final state, by rule (1).
- By rules (2), (3), and (4), there can be only one edge, going from the start state to the final state, and the only possible path from the start state to a final state has exactly one edge, this one.
- This edge is labeled by a regular expression R , and the language of the r.e.-NFA is exactly $L(R)$.

Overview of the Construction

- Thus $L(R)$ is also the language of the original DFA.
- The states we eliminate are every state except the start state and final state.
- We can eliminate them in any order and get a *correct* final regular expression, but if we choose the order wisely we may get a simpler regular expression.

Eliminating a State

- Suppose we have a state q that is neither initial nor final, and we want to eliminate it.
- We don't care about paths that *start or end* at q , because the language is defined only in terms of paths that start at the initial state and end at the final state.
- To safely delete q , we have to *replace* any two-step path, that had q as its middle node, by a single edge.

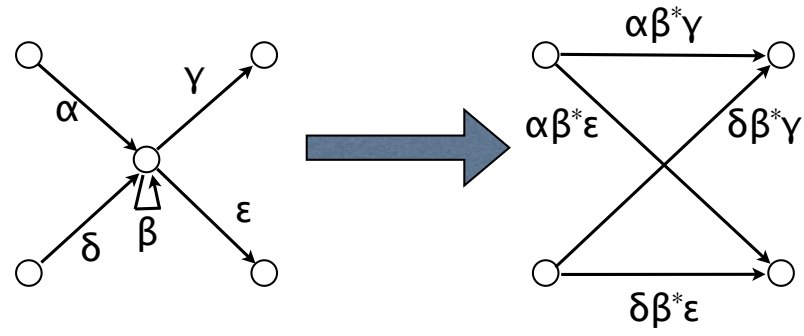
Eliminating a State

- If (p, α, q) and (q, β, r) are any two edges, and (q, γ, q) is the loop on q , then when we delete q we must add a new edge $(p, \alpha\gamma^*\beta, r)$.
- (Here α , β , and γ are regular expressions. Note also that $p = r$ is possible.)
- If there is already an edge from p to r , though, we add the new edge by changing the existing (p, δ, r) to $(p, \delta + \alpha\gamma^*\beta, r)$.

Eliminating a State

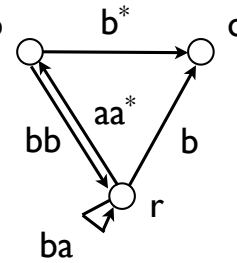
- (Note that if there is no loop on q we can take γ to be \emptyset and then $\gamma^* = \emptyset^*$ which is the identity for concatenation, so that $\alpha\gamma^*\beta = \alpha\beta$.)
- When we delete q , we should count all the m edges into q and all the n edges out of q , and make sure that we have added mn new edges. The loop on q , if it exists, does not count toward either m or n .

A General Example



Clicker Question #2

- Which transition *will* be present in the new r.e.-NFA if p we eliminate state p ?



- (a) (r, aa^*bb, r)
- (b) $(r, (ba)^*b, q)$
- (c) $(r, b + aa^*b^*, q)$
- (d) (r, ba, r)

Answer #2

- Which transition *will* be present in the new r.e.-NFA if we eliminate state p ?

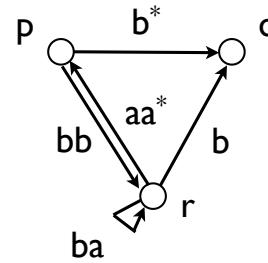
- (a) (r, aa^*bb, r)

- (b) $(r, (ba)^*b, q)$

- (c) $(r, b + aa^*b^*, q)$

- (d) (r, ba, r)

- (the merger of (a) and (d) is present, but not either one)

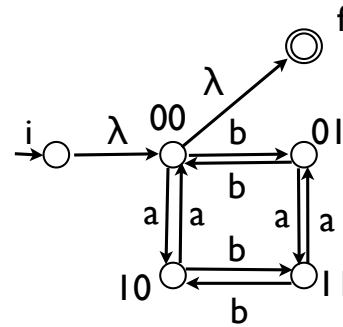


Example: The Language EE

- In Discussion #8 we designed a regular expression for the language EE, of strings over $\{a, b\}$ that have both an even number of a's and an even number of b's.
- We'll now use State Elimination to get such an expression from a DFA.
- The natural DFA has state set $\{00, 01, 10, 00\}$. Here 00 is the start state and the only final state, a's change the first bit of the state, and b's change the second bit.

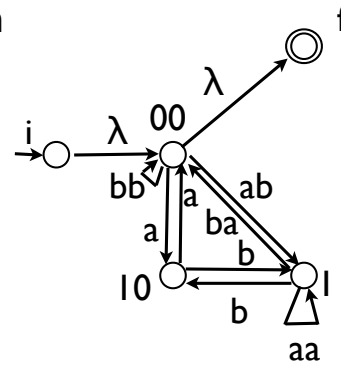
Example: The Language EE

- But this DFA violates the rules for an r.e.-NFA -- we have to add a new start state i and a new final state f , and add transitions $(i, \lambda, 00)$ and $(00, \lambda, f)$.
- Now all we have to do is eliminate four states to get our regular expression.



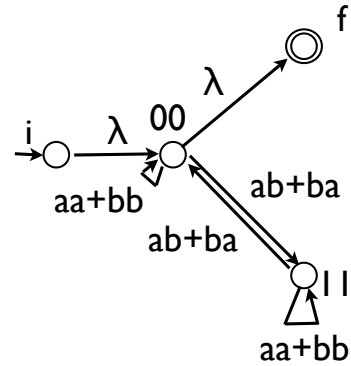
Example: The Language EE

- We begin by killing 01, which has two edges in and two out.
- We need four new edges: (00, bb, 00), (00, ba, 11), (11, ab, 00), and (11, aa, 11).



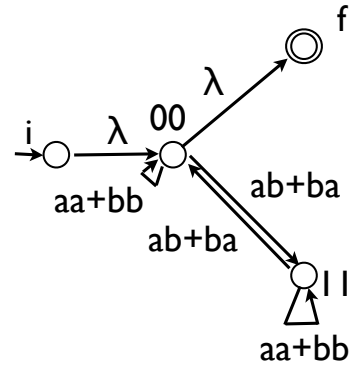
Example: The Language EE

- Next we eliminate λ (which looks like a good idea as it has no loop and fewer overall edges).
- Again we get four new edges, each of which is parallel to an existing edge, making $(00, aa+bb, 00)$, $(00, ab+ba, 11)$, $(11, ab+ba, 00)$, and $(11, aa+bb, 00)$.
- This gives us four states.



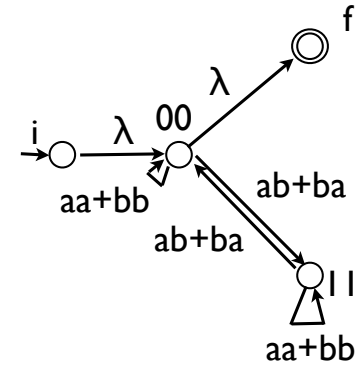
Clicker Question #3

- If we now eliminated 00, we would create four transitions. Which of these four would *not* appear in the new r.e.-NFA?
- (a) $(i, (aa+bb)^*, f)$
- (b) $(i, (aa+bb)^*(ab+ba)(aa+bb)^*, | |)$
- (c) $(| |, (ab+ba)(aa+bb)^*, f)$
- (d) $(| |, aa + bb + (ab+ba)(aa+bb)^*(ab+ba), | |)$



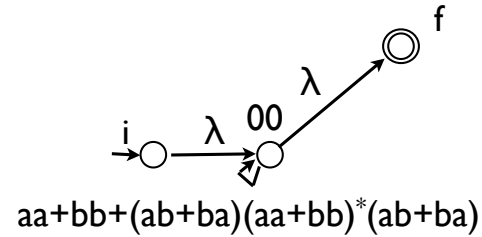
Answer #3

- If we now eliminated 00, we would create four transitions. Which of these four would *not* appear in the new r.e.-NFA?
- (a) $(i, (aa+bb)^*, f)$
- (b) $(i, (aa+bb)^*(ab+ba)(aa+bb)^*, ll)$ (no second loop)
- (c) $(ll, (ab+ba)(aa+bb)^*, f)$
- (d) $(ll, aa + bb + (ab+ba)(aa+bb)^*(ab+ba), ll)$



Finishing the EE Example

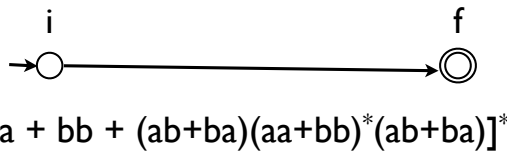
- The four remaining states are i , 00 , 11 , and f .
- State 11 now has one edge in and one edge out, along with a loop.
- When we eliminate 11 we create only one edge, $(00, (ab+ba)(aa+bb)^*(ab+ba), 00)$.



Finishing the EE Example

- The last state to eliminate is now 00, which also has one edge in, one edge out, and one loop.

- (Note that *any* three-state r.e.-NFA must have a form similar to this, maybe with another edge from initial to final state.)



Finishing the EE Example

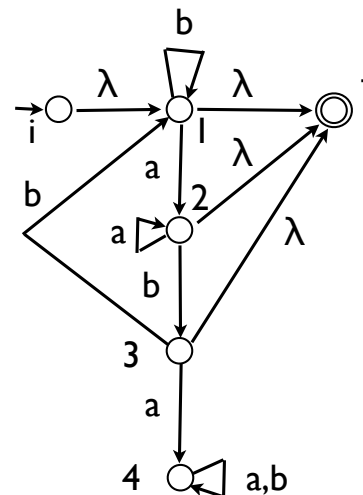
- The one edge that we create is $(i, [aa+bb+(ab+ba)(aa+bb)^*(ab+ba)]^*, f)$, and our final regular expression is the label of this edge.
- We would get a grubbier, equivalent regular expression by eliminating the states in a different order.
- The expression $aa+bb+(ab+ba)(aa+bb)^*(ab+ba)$ represents the language EEP of “primitive” (non-factorable) strings in EE.

Example: The Language No-aba

- We've seen the language Yes-aba = $\Sigma^* \text{aba} \Sigma^*$ and its complement No-aba several times now. We have a four-state DFA for No-aba -- let's turn this into a regular expression.
- The state set is $\{1,2,3,4\}$, the start state 1, final state set $\{1,2,3\}$, and edges $(1,a,2)$, $(1,b,1)$, $(2,a,2)$, $(2,b,3)$, $(3,a,4)$, $(3,b,1)$, $(4,a,4)$, and $(4,b,4)$.
- Again we need new start states i and f , with new edges $(i,\lambda,1)$, $(1,\lambda,f)$, $(2,\lambda,f)$, and $(3,\lambda,f)$.

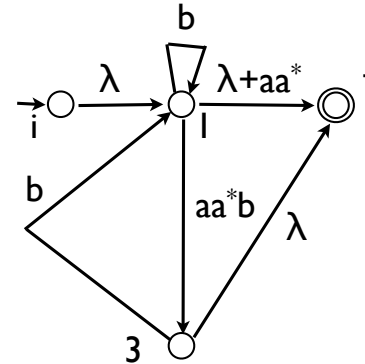
Example: The Language No-aba

- The first thing to do is to kill state 4, which requires adding no new edges, because it has no paths through it from i to f .
- Next, state 2 looks like a good target. It has one edge in and two out, for two new edges.



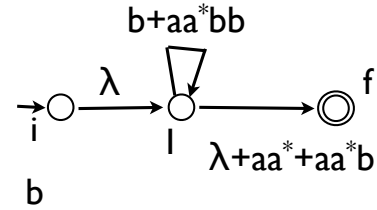
Example: The Language No-aba

- Killing state 2 produces two new edges, $(1, aa^*b, 2)$ and $(1, aa^*, f)$.
- The latter edge is merged with the existing edge $(1, \lambda, f)$.

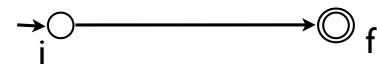


Example: The Language No-aba

- Now if we kill state 3 we create $(1, aa^*bb, 1)$ which becomes $(1, b + aa^*bb, 1)$ and $(1, aa^*b, f)$ which becomes $(1, \lambda + aa^* + aa^*b, f)$.
- Killing state 1 gives the final expression $(b + aa^*bb)^*(\lambda + aa^* + aa^*b)$.



$$(b+aa^*bb)^*(\lambda+aa^*+aa^*b)$$

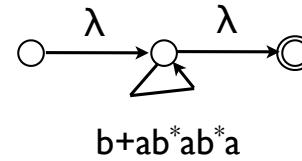
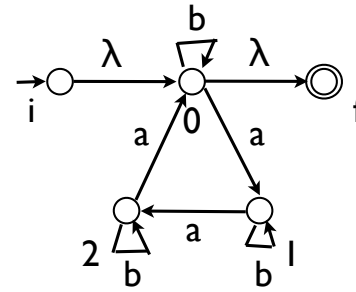


Example: # of a's Divisible by 3

- Here's another example (Exercise 14.10.3 in the text). Let D be the language of strings over $\{a, b\}$ where the number of a's is divisible by 3.
- It's clear how to make a DFA for this: states $\{0, 1, 2\}$, start state and only final state 0, edges (p, b, p) for each state p , and edges $(0, a, 1)$, $(1, a, 2)$, and $(2, a, 0)$.
- To make an r.e.-NFA, we once again add a new start state i and new final state f , with edges $(i, \lambda, 0)$ and $(0, \lambda, f)$. We have five states now and must kill three.

Example: # of a's Divisible by 3

- We first kill 2, creating one new edge $(1, ab^*a, 0)$.
- Then killing 1 creates a new edge $(0, ab^*ab^*a, 0)$, which adds to the existing $(0, b, 0)$ to get $(0, b + ab^*ab^*a, 0)$.



Example: # of a's Divisible by 3

- Finally, killing 0 gives the expression $[b + ab^*ab^*a]^*$, which makes sense because we can break any string in D into pieces that are either b's or have exactly three a's.
- A more challenging problem is the language of strings where *both* the number of a's and the number of b's are divisible by three.
- How about the strings where the number of a's and the number of b's are congruent to one another modulo 3?