Lecture 3: Closure Properties & Regular Expressions

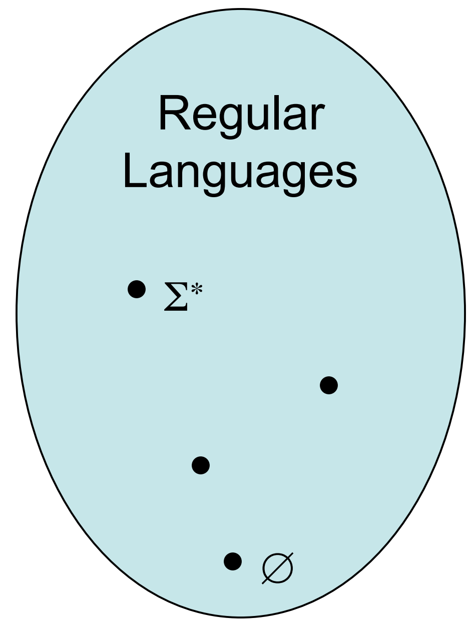
Jim Hook
Tim Sheard
Portland State University

Last Time

- Defined DFA, regular languages
- Defined NFA, showed equivalent to DFA
- Showed closure properties of Regular Languages

Why do we care about closure properties?

- One course objective is to "map the world"
- Closure properties tell us how to build new regular languages from old



Can properties define a Class of Languages?

What is the smallest class of languages:

- That contains
  - the empty language
  - the universal language
  - every singleton character of the alphabet
- And is closed under
  - union
  - concatenation
  - iteration (Kleene star)

Regular languages?

- Can the regular languages from last lecture be this smallest class?
- Since they meet the requirements they must at least contain this smallest class
  - Discuss
- How can we tell if there are regular languages not in this class?

Regular expressions

- Kleene introduced regular expressions (REGEXP) to name the languages in this "smallest class"
  - a is a REGEXP for every a in Σ
  - ε is a REGEXP
  - $\varnothing$ is a REGEXP
  - if R_1 and R_2 are REGEXPs then the following are REGEXPs
    - $R_1 + R_2$
    - $R_1 R_2$
    - R_1^*

Regular expressions and Regular Languages

- Thm [1.54] A language is regular iff it is described by a regular expression
- Lemma [1.55] If a language is descried by a regular expression then it is regular
- Proof sketch:
  - For each REGEXP we must show that a corresponding NFA can be constructed
  - We've done the hard work by proving the closure properties
  - We just have to complete the base cases for {a}, {ε}, and ∅.

Regular expressions and Regular Languages

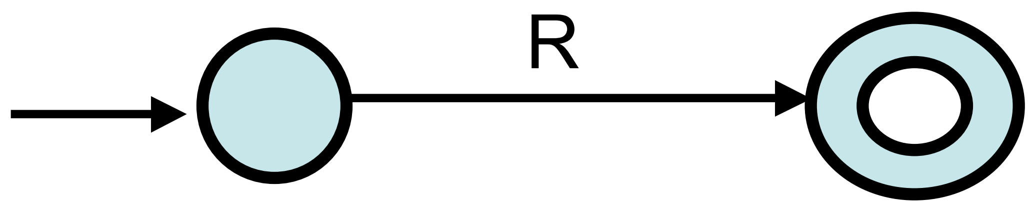
- More interesting: can we convert a DFA M into a regular expression?
- Lemma [1.60] If a language is regular then it is described by a regular expression
  - How do we prove this?
  - Can we calculate a REGEXP from a DFA?

DFA -> REGEXP

- One construction:
  - Draw a graph labeled essentially like the DFA
  - Find a way to remove states from the DFA systematically, replacing labels with regular expressions
  - Set things up so that when we are done the resulting regular expression describes the language accepted by the DFA

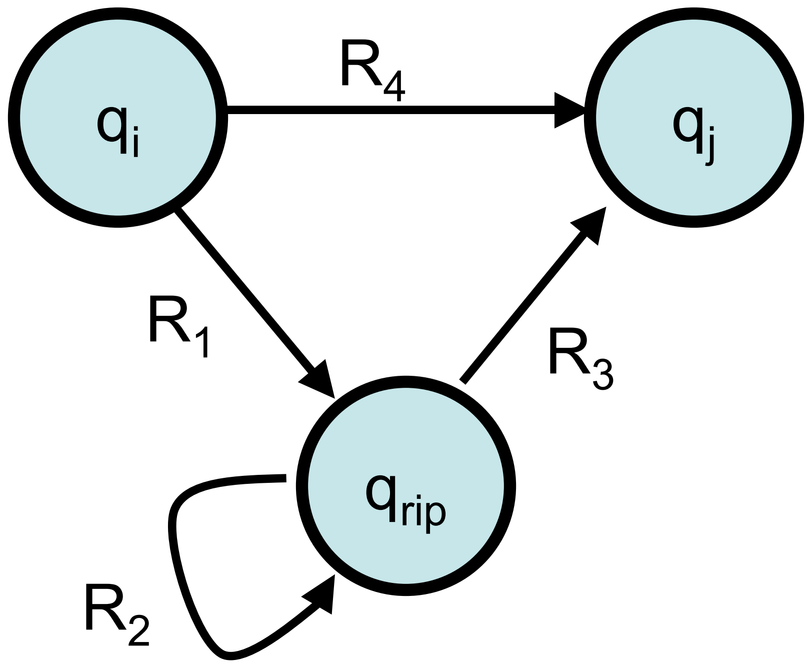
Generalized NFAs

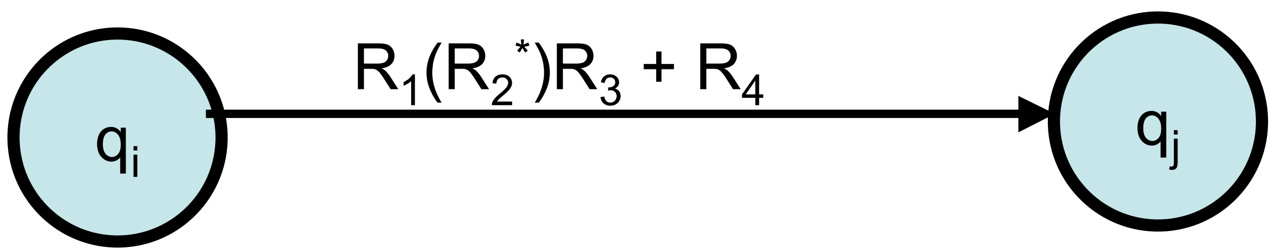
- Generalize an NFA to have regular expressions labeling transitions
- Goal is to simplify an automaton to:



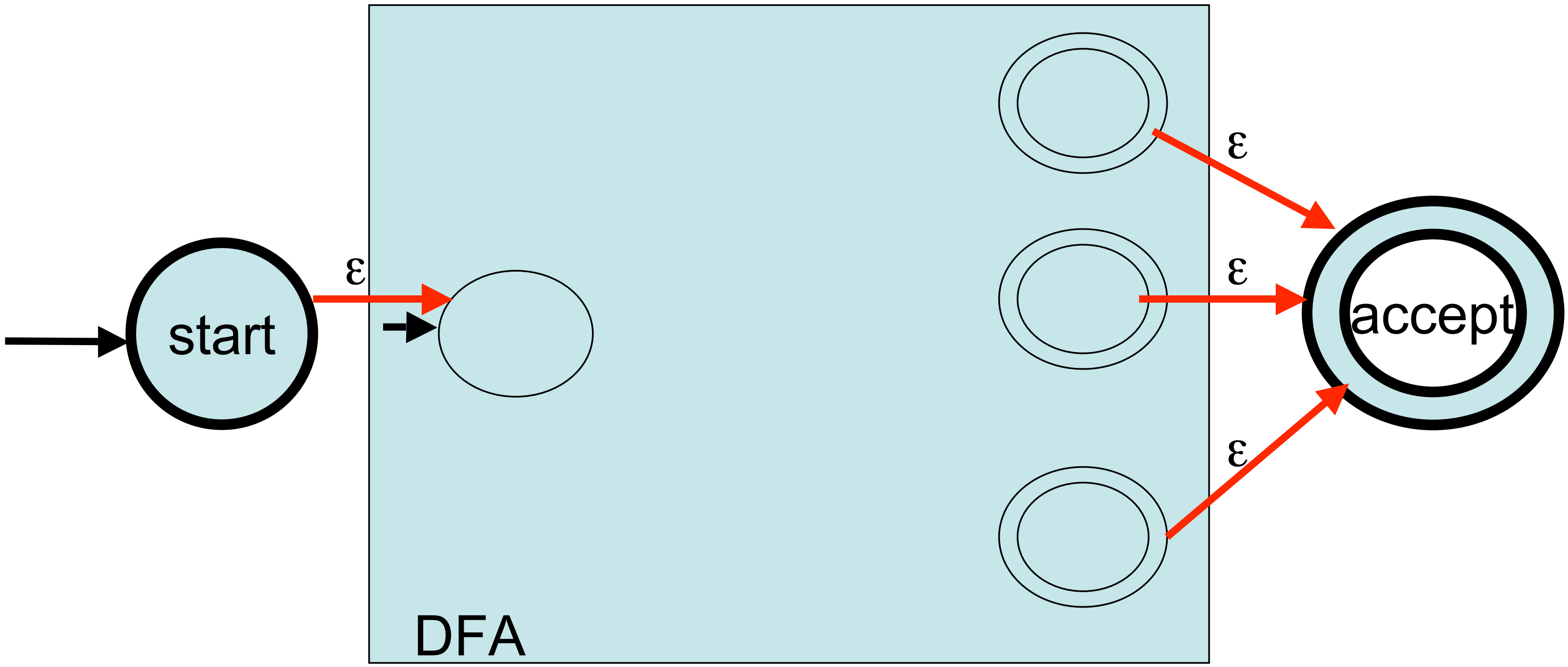
- Helpful to have single start and final state

Simplification (remove q_{rip})





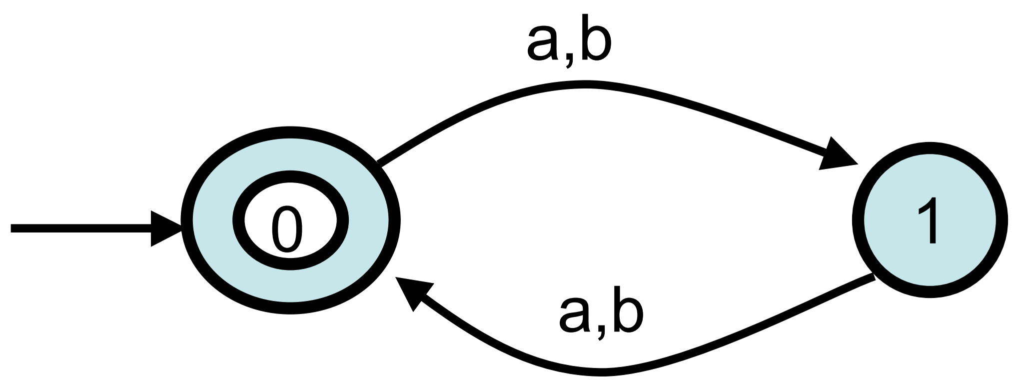
Initial Construction



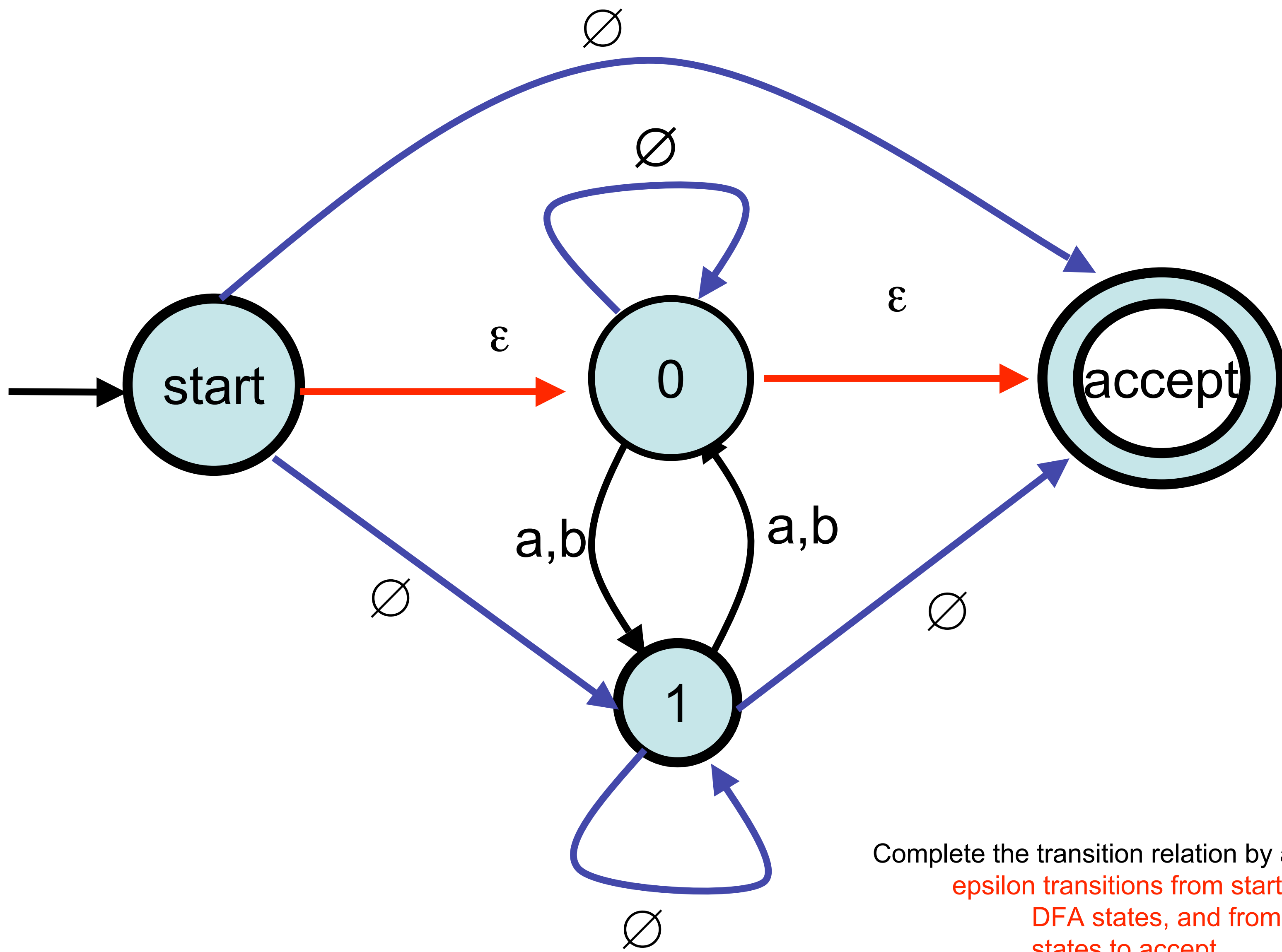
Complete the transition relation by adding

1. epsilon transitions from start to all initial DFA states, and from all DFA final states to accept.
2. null transitions between all unlabeled DFA states,

Example (DFA)



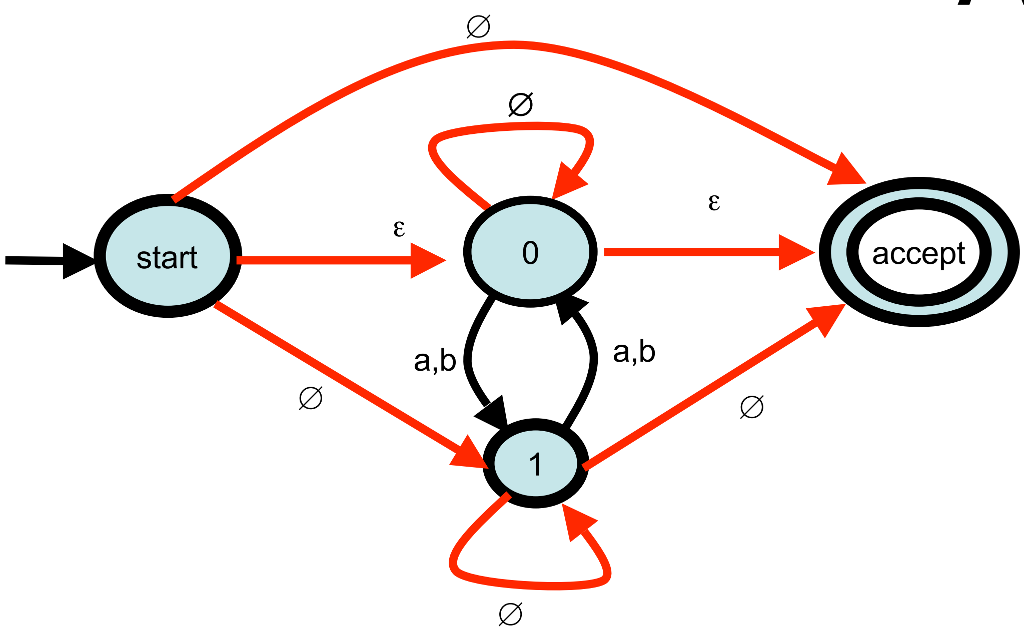
Example (GNFA)



Complete the transition relation by adding

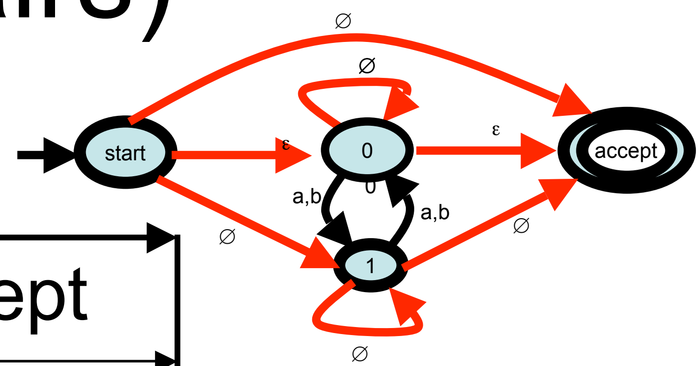
- epsilon transitions from start to all initial DFA states, and from all DFA final states to accept.
- null transitions between all unlabeled DFA states,

As Table



	0	1	accept
start	ε	∅	∅
0	∅	a+b	ε
1	a+b	∅	∅

Cut 1 (w.r.t all pairs)



	0	1	accept
start	ε	∅	∅
0	∅	a+b	ε
1	a+b	∅	∅

	0	accept
start	ε+ ∅ ∅*(a+b)	∅+ ∅ ∅* ∅
0	∅+(a+b) ∅*(a+b)	ε+(a+b) ∅* ∅

Simplifying

	0	accept
start	$\varepsilon + \emptyset\ \emptyset^*$**(a+b)**	$\emptyset$**+** $\emptyset\ \emptyset^*\ \emptyset$
0	$\emptyset$**+(a+b)** $\emptyset^*$**(a+b)**	ε+(a+b) $\emptyset^*\ \emptyset$

	0	accept
start	ε	$\emptyset$
0	**(a+b)(a+b)**	ε

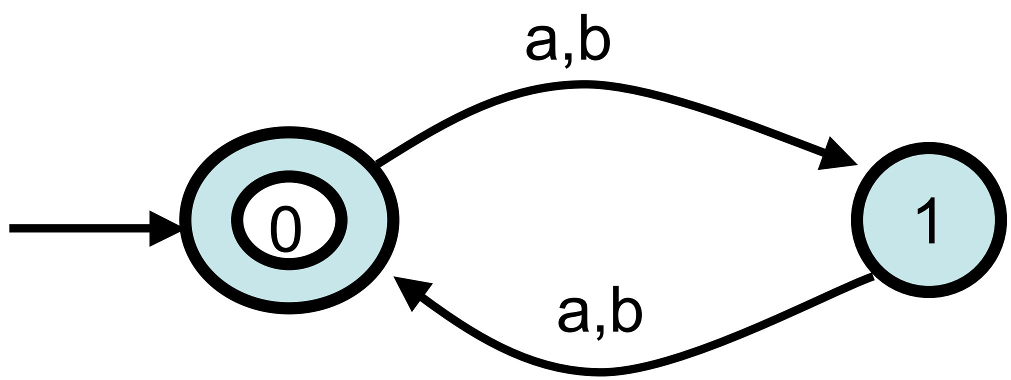
Cut 0 and simplify

	0	accept
start	ε	$\varnothing$
0	$(a+b)(a+b)$	ε

	accept
start	$\varnothing + \varepsilon((a+b)(a+b))^* \varepsilon$

	accept
start	$((a+b)(a+b))^*$

Example (conclusion)



$$((\mathbf{a+b})(\mathbf{a+b}))^*$$

Proof Sketch

- Formalize GNFA
  - adjust delta to give REGEXP
  - define acceptance for GNFA
    - this will give a sequence of states visited on acceptance
- Show “ripping a state” preserves language accepted
  - Let G’ be obtained from G by ripping q_{rip}
  - Show: $w \in L(G') \Rightarrow w \in L(G)$
  - Show: $w \in L(G) \Rightarrow w \in L(G')$

Proof Sketch (cont)

- Let G' be obtained from G by ripping q_{rip}
- Show: $w \in L(G') \Rightarrow w \in L(G)$
  - $w \in L(G')$ implies there is a sequence of states: $q_{start}, q_1, \ldots, q_{accept}$ and substrings $w_1, w_2, \ldots, w_n$ satisfying the acceptance conditions
  - Look at each w_i, either
    - w_i comes from an "R4" rule, or
    - w_i comes from an R1 R2* R3 rule
  - If w_i comes from an R4 rule then G can make a corresponding step
  - If w_i comes from an R1 R2* R3 rule, then w_i is of the form $y_1 \ldots y_m$, where
    - $y_1 \in R1$,
    - $y_i \in R2$ (1<I<m)
    - $y_m \in R3$
  - In this case G transitions from q_i-1 to q_i with m-2 intermediate instances of q_{rip} on input $w_i = y_1 \ldots y_m$

Proof Sketch (cont)

- Let G' be obtained from G by ripping q_{rip}
- Show: $w \in L(G) \Rightarrow w \in L(G')$
- $w \in L(G)$ impiles there are states $q_{start}, q_1, \ldots, q_{accept}$ and strings $w_1, \ldots, w_n$ satisfying conditions of acceptance
- Cases:
  - q_{rip} not used in computation: w clearly in L(G') (use only R4 rules)
  - q_{rip} is used:
    - every occurrence of q_{rip} is in a context of the form:
      - $q_i\ q_{rip}\ q_{rip} \ldots q_{rip}\ q_j$, in which there is one or more occurrences of q_{rip} between non rip states i and j.
      - In this case
        - » w_{i+1} will be an "R1" string
        - » $w_{I+2}, \ldots, w_{j-1}$ will be "R2" strings (there may be 0 of these)
        - » w_j will be an "R3" string

Proof Sketch (cont)

- Let G' be obtained from G by ripping q_rip
- Show: w $\in$ L(G) => w $\in$ L(G')
- Cases:
  - q_{rip} is used:
    - every occurrence of q_{rip} is in a context of the form:
      - q_i q_{rip} q_{rip} … q_{rip} $q_{j,}$ in which there is one or more occurrences of q_{rip} between non rip states i and j.
      - In this case
        - » w_{i+1} will be an "R1" string
        - » w_{I+2}, …, w_{j-1} will be "R2" strings (there may be 0 of these)
        - » w_j will be an "R3" string
      - Consequently, G' will transition from q_i to q_j on w_{i+1} … w_j by an R1 R2* R3 transition

Next time

- Non-regular languages (pumping lemma)