Recall: Regular Languages

The language recognized by a finite automaton M is denoted by **L(M).**

A <u>regular language</u> is a language for which there exists a recognizing finite automaton.

Terminology: closure

- A set is defined to be closed under an operation if that operation on members of the set always produces a member of the same set. (adapted from wikipedia)

E.g.:

- The integers are closed under addition, multiplication.
- The integers are not closed under division
- Σ^* is closed under concatenation

- A set can be defined by closure -- Σ^* is called the (Kleene) closure of Σ under concatenation.

Terminology: Regular Operations

Pages 44-47 (Sipser, 3rd edition)

Chapter 2, p13 (Goddard, 1st edition)

The regular operations are:

1. Union
2. Concatenation
3. Star (Kleene Closure): For a language A,

   $A^* = \{w_1w_2w_3\ldots w_k \mid k \geq 0, \text{ and each } w_i \in A\}$

Closure Properties

- Set of regular languages is closed under
  - -- Complementation
  - – Union
  - – Concatenation
  - – Star (Kleene Closure)

Complement of a regular language

- Swap the accepting and non-accept states of **D**FA M to get M'.
- Note: Doesn't necessarily work on an NFA! Find an example where it does not.

- The complement of a regular language is regular.

Other closure properties

Union: Can be done with DFA, but using a complicated construction.

Concatenation: We tried and failed

Star: ???

We introduced non-determinism in FA

Recall: NFA drawing conventions

- Not all transitions are provided

(E.g., what to do on a ‘b’ when in state 1)

- missing transitions are assumed to go to a reject state (‘dead state’) from which the automaton cannot escape

Closure under regular operations

Union (new proof):

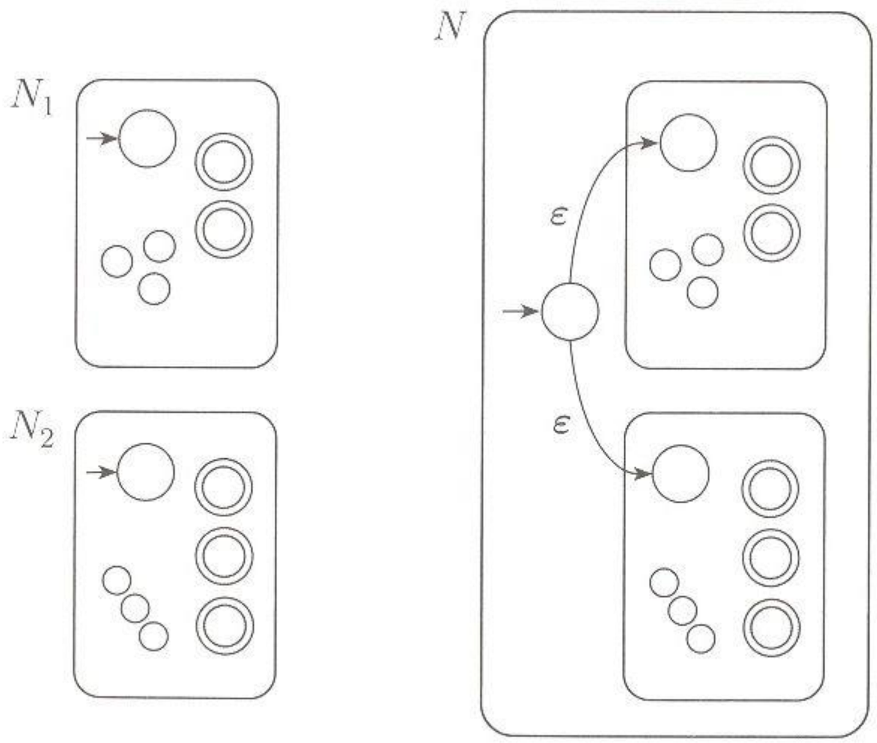


FIGURE 1.46
Construction of an NFA N to recognize $A_1 \cup A_2$

Closure under regular operations

Concatenation:

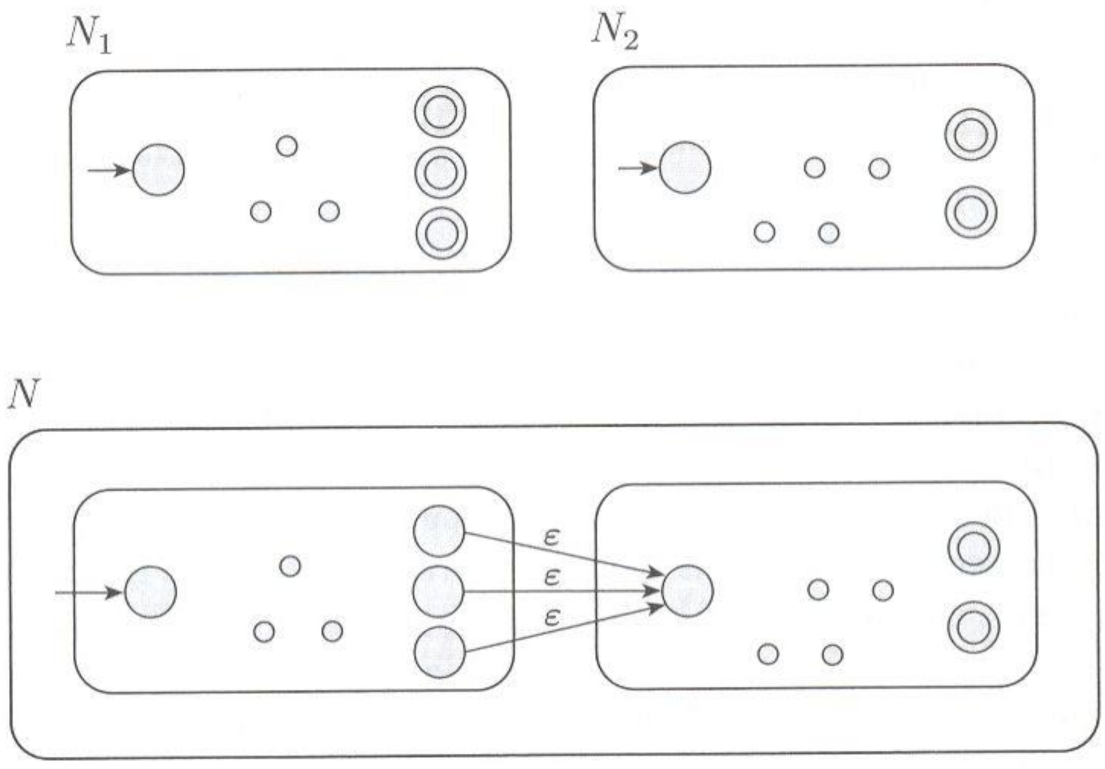


FIGURE 1.48
Construction of N to recognize $A_1 \circ A_2$

Closure under regular operations

Star:

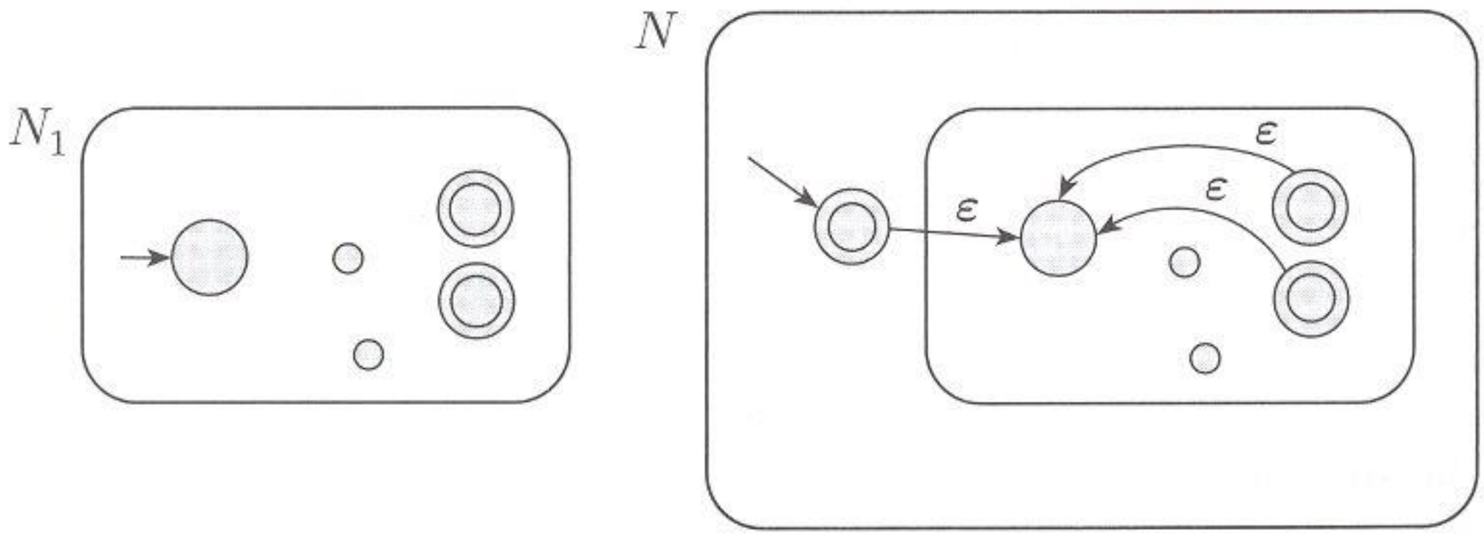


FIGURE 1.50
Construction of N to recognize A^*

Incorrect reasoning about RL

- Since $L_1 = \{w \mid w=a^n, n \in \mathbf{N}\}$,
  $L_2 = \{w \mid w = b^n, n \in \mathbf{N}\}$ are regular,
  therefore $L_1 \bullet L_2 = \{w \mid w=a^n b^n, n \in \mathbf{N}\}$ is regular
- If L_1 is a regular language, then
  $L_2 = \{w^R \mid w \in L_1\}$ is regular, and
  Therefore $L_1 \bullet L_2 = \{w\, w^R \mid w \in L_1\}$ is regular

Are NFA more powerful than DFA?

- NFA can solve every problem that DFA can (DFA are also NFA)
- Can DFA solve every problem that NFA can?

Equivalence of NFA, DFA

- Pages 54-58 (Sipser, 2nd ed)
- We will prove that every NFA is equivalent to a DFA (with upto exponentially more states).
- Non-determinism does not help FA's to recognize more languages!

Epsilon Closure

- Let $N=(Q,\Sigma,\delta,q_0,F)$ be any NFA
- Consider any set $R \subseteq Q$
- $E(R) = \{q|q$ can be reached from a state in R by following 0 or more ε-transitions$\}$
- $E(R)$ is the epsilon closure of R under ε-transitions

Proving equivalence

For all languages $L \subseteq \Sigma^*$

$$L = L(N) \quad \textit{iff} \quad L = L(M)$$

for some NFA N for some DFA M

One direction is easy:

A DFA M is also a NFA N. So N does not have to be `constructed' from M

Proving equivalence – contd.

The other direction:

Construct M from N

- N = (Q,Σ,δ,q_0,F)
- Construct M= $(Q',\Sigma,\delta',q'_0,F')$ such that,
  - for any string $w \in \Sigma^*$,
  - w is accepted by N iff w is accepted by M

Special case

- Assume that ε is not used in the NFA N.
  - Need to keep track of each subset of N
  - So $Q' = \mathcal{P}(Q)$, $q'_0 = \{q_0\}$
  - $\delta'(R,a) = \cup(\delta(r,a))$ over all $r \in R$, $R \in Q'$
  - $F' = \{R \in Q' \mid R$ contains an accept state of N$\}$
- Now let us assume that ε is used.

Construction (general case)

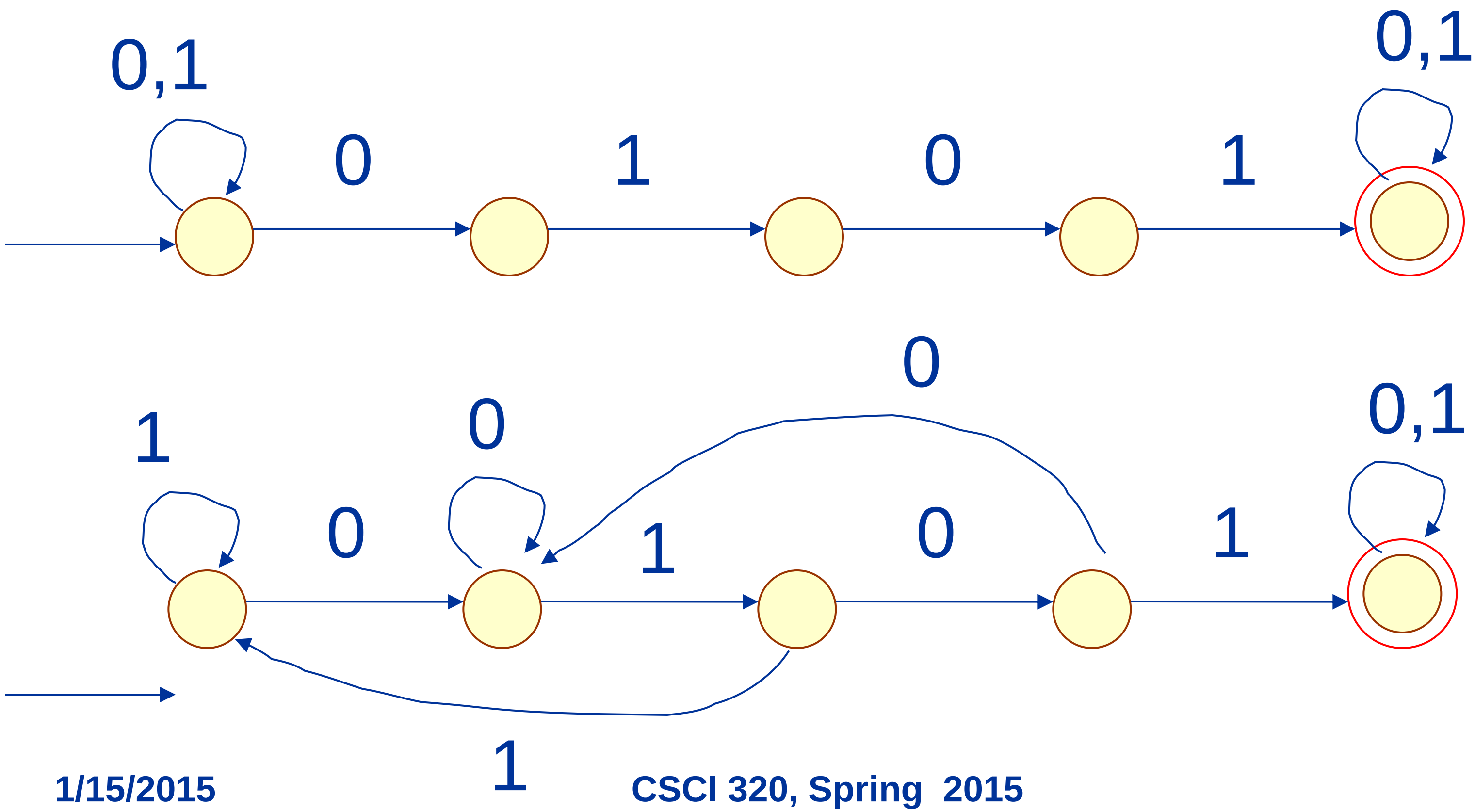
1. $Q' = \mathcal{P}(Q)$
2. $q'_0 = E(\{q_0\})$
3. for all $R \in Q'$ and $a \in \Sigma$
   $\delta'(R, a) = \{q \in Q | q \in E(\delta(r,a))$ for some $r \in R\}$
4. $F' = \{ R \in Q' |$ R contains an accept state of N$\}$

Why the construction works

- for any string $w \in \Sigma^*$,
- w is accepted by N iff w is accepted by M
- Can prove using induction on the number of steps of computation…

State minimization

It may be possible to design DFA's without the exponential blowup in the number of states. Consider the NFA and DFA below.



Characterizing FA languages

- Regular expressions

Regular Expressions (Def. 1.52)

Given an alphabet Σ, R is a regular expression if:
(INDUCTIVE DEFINITION)

- $R = a$, with $a \in \Sigma$
- $R = \varepsilon$
- $R = \varnothing$
- $R = (R_1 \cup R_2)$, with R_1 and R_2 regular expressions
- $R = (R_1 \bullet R_2)$, with R_1 and R_2 regular expressions
- $R = (R_1{}^*)$, with R_1 a regular expression

Precedence order: $*$, $\bullet$, $\cup$

Regular Expressions

- Unix ‘grep’ command: Global Regular Expression and Print
- Lexical Analyzer Generators (part of compilers)
- Both use regular expression to DFA conversion

Examples

- $e_1 = a \cup b$, $L(e_1) = \{a,b\}$
- $e_2 = ab \cup ba$, $L(e_2) = \{ab,ba\}$
- $e_3 = a^*$, $L(e_3) = \{a\}^*$
- $e_4 = (a \cup b)^*$, $L(e_4) = \{a,b\}^*$
- $e_5 = (e_m \,.\, e_n)$, $L(e_5) = L(e_m) \bullet L(e_n)$
- $e_6 = a^*b \cup a^*bb$,

  $L(e_6) = \{w \mid w \in \{a,b\}^*$ and w has 0 or more a's followed by 1 or 2 b's$\}$

Characterizing Regular Expressions

- We prove that Regular expressions (RE) and Regular Languages are the same set, i.e.,

RE = RL

Thm 1.54: RL ~ RE

We need to prove both ways:

• If a language is described by a regular expression, then it is regular (Lemma 1.55)
(We will show we can convert a regular expression R into an NFA M such that L(R)=L(M))

• The second part:
If a language is regular, then it can be described by a regular expression (Lemma 1.60)

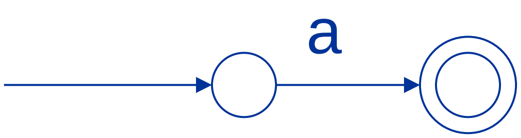
Regular expression to NFA

Claim: If L = L(e) for some RE *e*, then L = L(M) for some NFA M

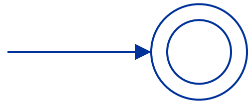
Construction: Use inductive definition

1. $R = a$, with $a \in \Sigma$
2. $R = \varepsilon$
3. $R = \varnothing$
4. $R = (R_1 \cup R_2)$, with R_1 and R_2 regular expressions
5. $R = (R_1 \bullet R_2)$, with R_1 and R_2 regular expressions
6. $R = (R_1{}^*)$, with R_1 a regular expression

1.



2.



3.

4,5,6: similar to closure of RL under regular operations.

Examples of RE to NFA conv.

$L = \{ab, ba\}$

$L = \{ab, abab, ababab, \ldots\ldots\}$

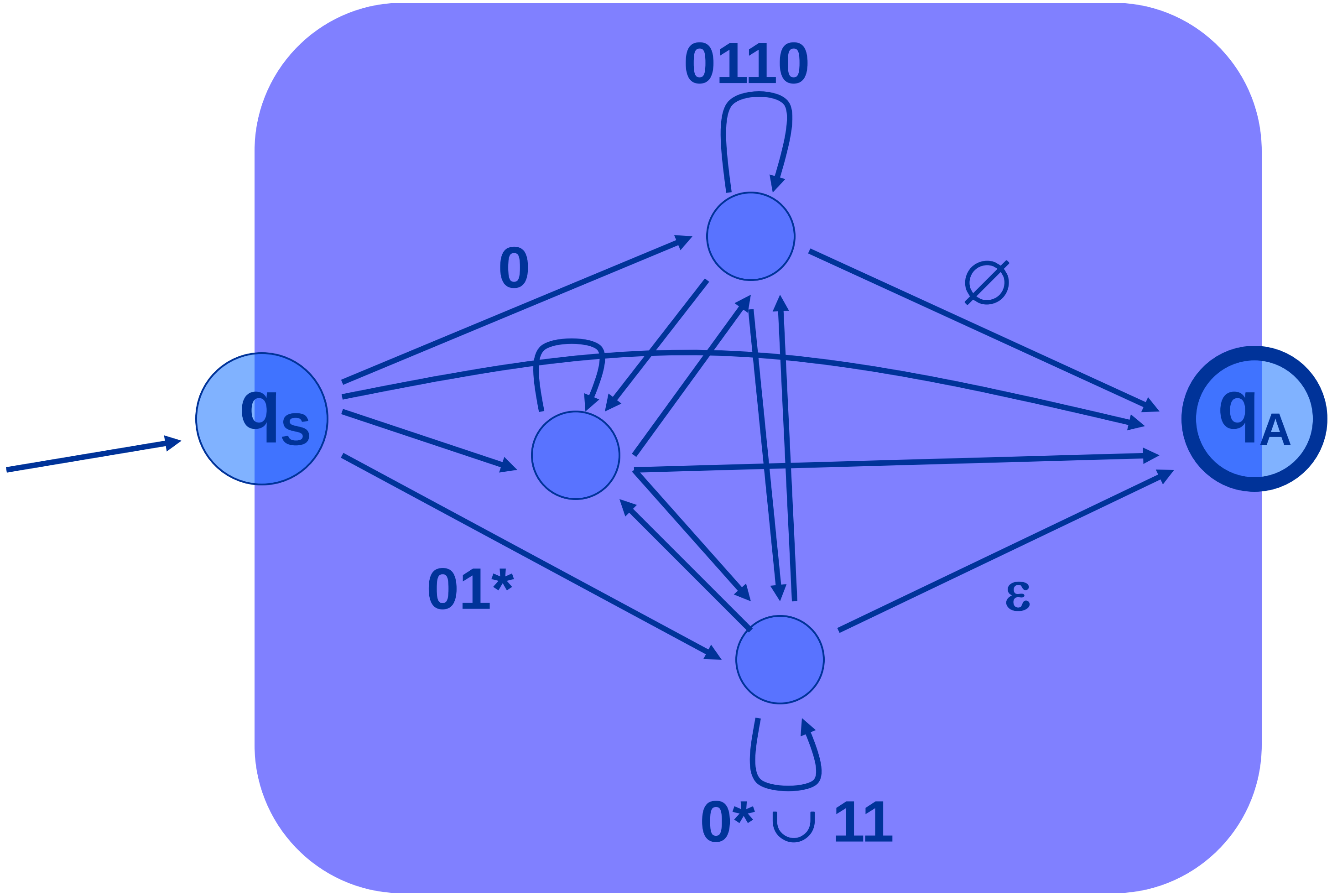
$L = \{w \mid w = a^m b^n,\ m<10,\ n>10\}$

Back to RL ~ RE

- The second part (Lemma 1.60):
  If a language is regular, then it can be described by a regular expression.
- Proof strategy:
  - regular implies equivalent DFA.
  - convert DFA to GNFA (generalized NFA)
  - convert GNFA to NFA.

GNFA: NFA that have regular expressions as transition labels

Example GNFA



Generalized NFA - defn

Generalized non-deterministic finite automaton $M=(Q, \Sigma, \delta, q_{start}, q_{accept})$ with

- Q finite set of states
- Σ the input alphabet
- q_{start} the start state
- q_{accept} the (unique) accept state
- $\delta:(Q - \{q_{accept}\})\times(Q - \{q_{start}\}) \to \mathcal{R}$ is the transition function

($\mathcal{R}$ is the set of regular expressions over Σ)

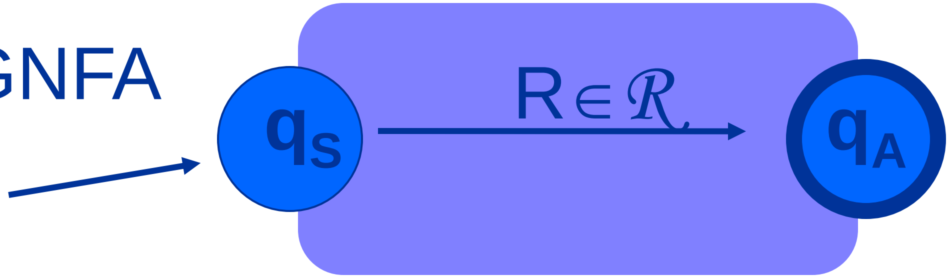
(NOTE THE NEW DEFN OF δ)

Characteristics of GNFA's δ

- $\delta:(Q\backslash\{q_{accept}\})\times(Q\backslash\{q_{start}\}) \rightarrow \mathcal{R}$

The interior $Q\backslash\{q_{accept},q_{start}\}$ is fully connected by δ
From q_{start} only 'outgoing transitions'
To q_{accept} only 'ingoing transitions'
Impossible $q_i \rightarrow q_j$ transitions are labeled "$\delta(q_i,q_j) = \varnothing$"

Observation: This GNFA recognizes the language L(R)



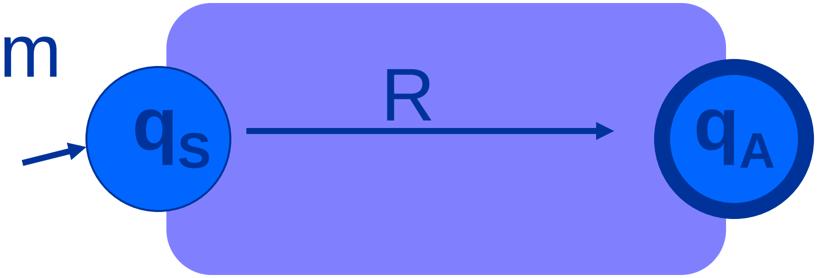
Proof Idea of Lemma 1.60

Proof idea (given a DFA M):

Construct an equivalent GNFA M' with $k \geq 2$ states

Reduce one-by-one the internal states until $k=2$

This GNFA will be of the form



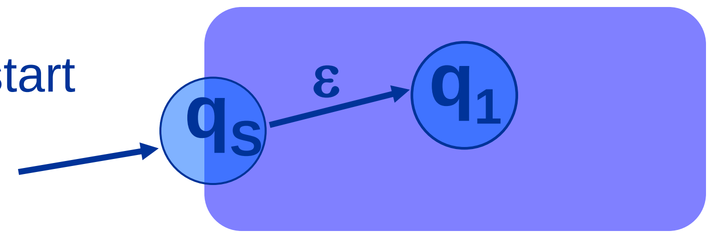
This regular expression R
will be such that $L(R) = L(M)$

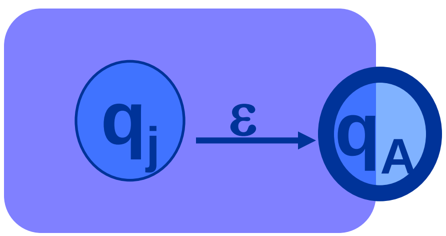
DFA M → Equivalent GNFA M’

Let M have k states $Q=\{q_1,\ldots,q_k\}$

- Add two states q_{accept} and q_{start}

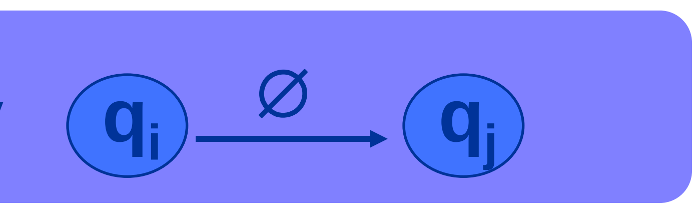
- Connect q_{start} to earlier q_1:



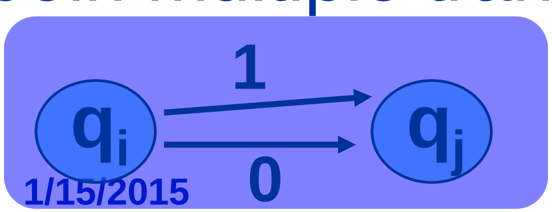


- Connect old accepting states to q_{accept}

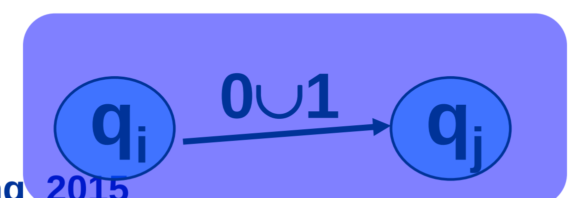
- Complete missing transitions by



- Join multiple transitions:



becomes



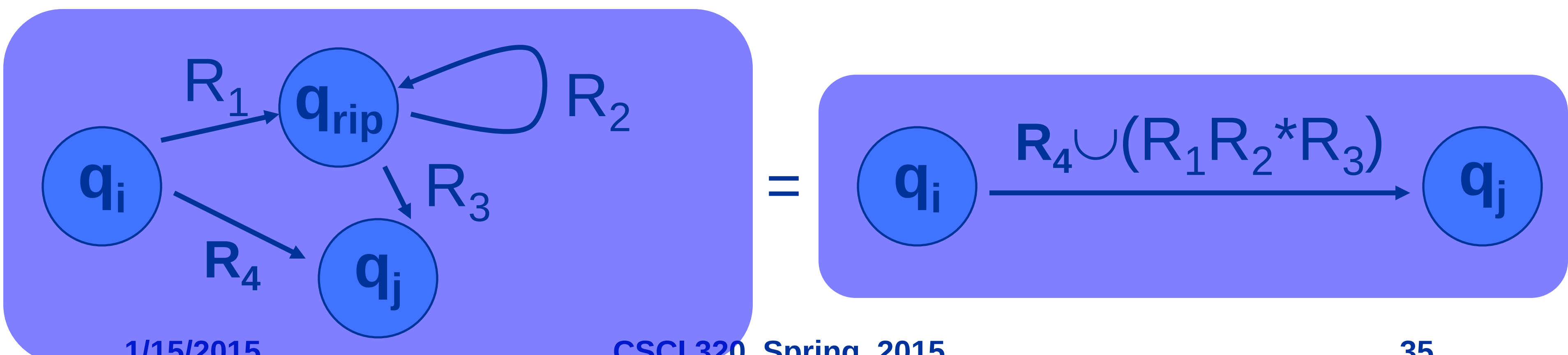
Remove Internal state of GNFA

If the GNFA M has more than 2 states, ‘rip’ internal q_{rip} to get equivalent GNFA M’ by:

- Removing state q_{rip}: $Q' = Q \setminus \{q_{rip}\}$
- Changing the transition function δ by

$$\delta'(q_i,q_j) = \delta(q_i,q_j) \cup (\delta(q_i,q_{rip})(\delta(q_i,q_j))^*\delta(q_{rip},q_j))$$

for every $q_i \in Q' \setminus \{q_{accept}\}$ and $q_j \in Q' \setminus \{q_{start}\}$



Proof Lemma 1.60

Let M be DFA with k states

Create equivalent GNFA M' with k+2 states

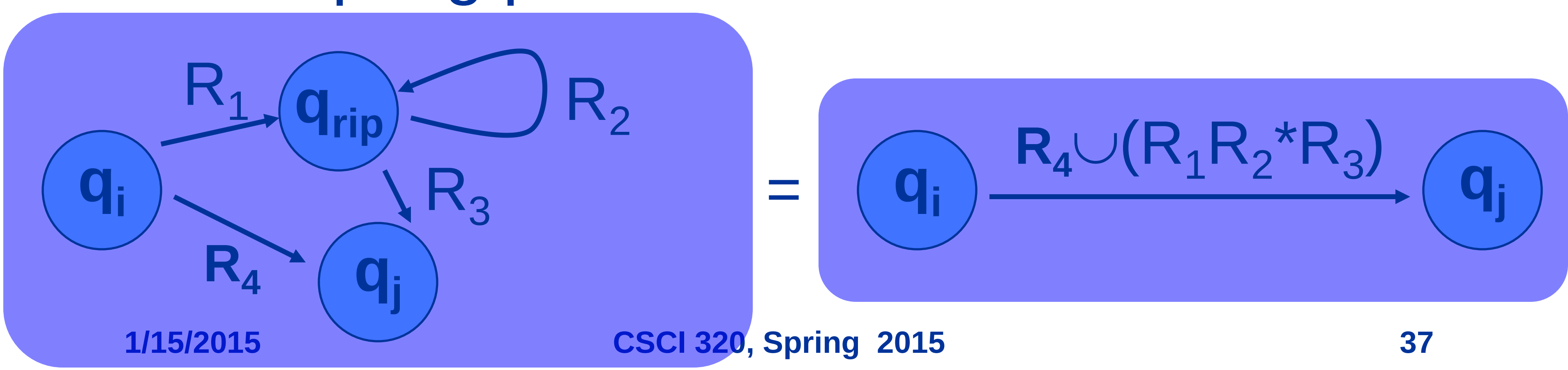
Reduce in k steps M' to M'' with 2 states

The resulting GNFA describes a single regular expressions R

The regular language L(M) equals the language L(R) of the regular expression R

Proof Lemma 1.60 - continued

- Use induction (on number of states of GNFA) to prove correctness of the conversion procedure.
- Base case: k=2.
- Inductive step: 2 cases – q_{rip} is/is not on accepting path.



Recap RL = RE

Let R be a regular expression, then there exists an NFA M such that L(R) = L(M)

The language L(M) of a DFA M is equivalent to a language L(M') of a GNFA = M', which can be converted to a two-state M''

The transition $q_{start} \xrightarrow{R} q_{accept}$ of M'' obeys L(R) = L(M'')

Hence: RE $\subseteq$ NFA = DFA $\subseteq$ GNFA $\subseteq$ RE

Example

L = {w| the sum of the bits of w is odd}