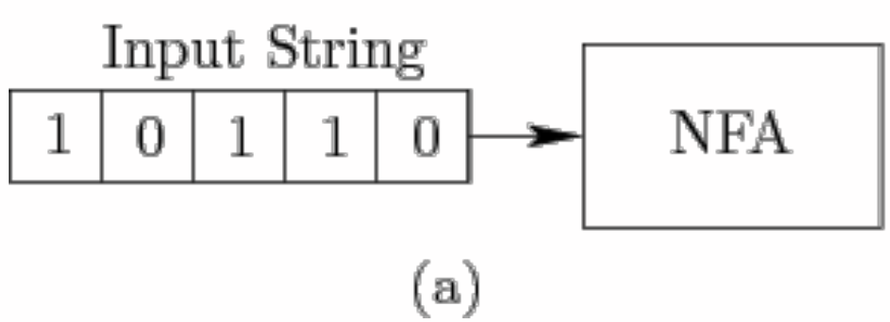
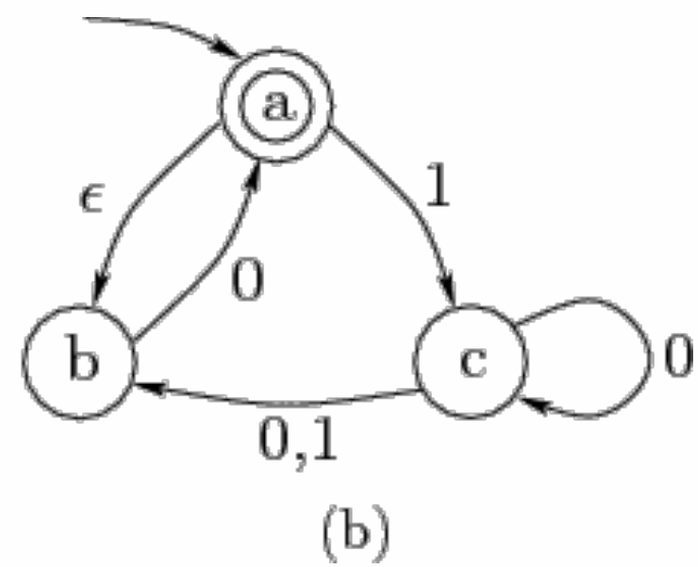
CS 321

Theory of Computation

Part I: Finite Automata and Regular Languages

Sipser 0, 1; Linz 1, 2, 3.1, 3.2, 4.1, 4.3



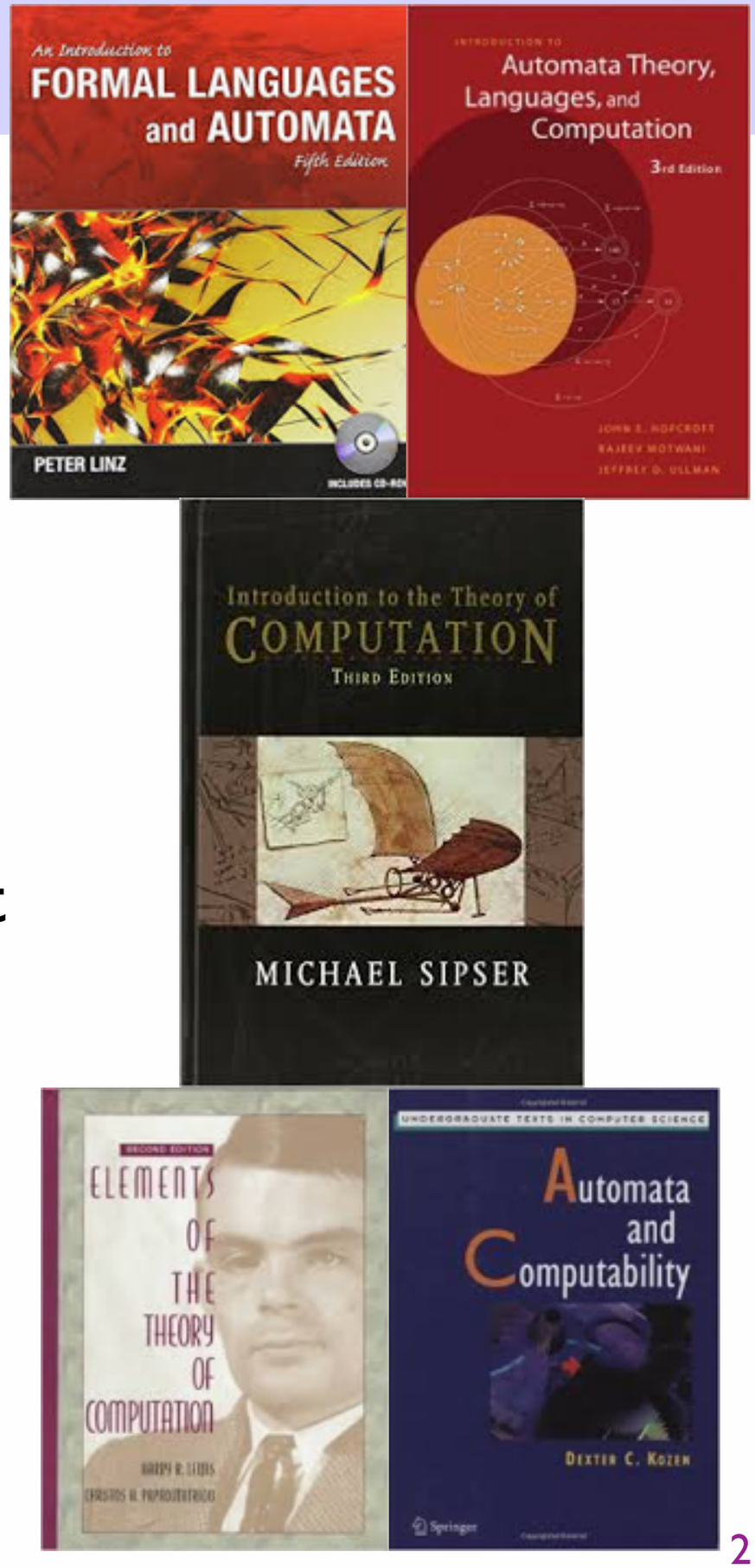


Instructor: Liang Huang

(some slides courtesy of J. Ullman and H. Potter)

Logistics

- Instructor: Liang Huang
  - Office Hours: MW 4-5pm, 2069 KEC
- TAs: Kai Zhao and Mingbo Ma
  - Office Hours: TRF 4-5pm, 4130 KEC
- Textbooks:
  - Sipser, 3rd edi. (or 2nd edi.) -- main text
  - *Linz, 5th edi. (or 4th) -- very helpful*
  - *Hopcroft, Motwani, & Ullman, 3rd edi.*
  - *Lewis & Papadimitriou, 2nd edi.*
  - *Kozen*



Logistics (2)

- Grading
  - Weekly Quizzes (5%x9=45%) -- 1st quiz: Wed 9/30
  - Weekly Homework (10%) -- due in class on a quiz day
    - graded for completeness, but not correctness; no late submissions
  - Midterm (20%)
  - Final (25%)
- Questions/Discussions -- on Canvas
  - first check if a similar question has been asked on Canvas
  - if not, ask it there; we'll answer it
  - for non-technical questions email cs321-fall15-orst@googlegroups.com

What is Computer Science

Computer Science is no more about computers than astronomy is about telescopes.

— (Mis)attributed to Edsger Dijkstra, 1970.

Computer science is not really about computers -- and it's not about computers in the same sense that physics is not really about particle accelerators, and biology is not about microscopes and Petri dishes... and geometry isn't really about using surveying instruments... when some field is just getting started and you don't really understand it very well, it's very easy to confuse the essence of what you're doing with the tools that you use.

— Harry Abelson (1986), Intro to Computer Science Course, MIT

What is (Theoretical) Computer Science

- CS is the study of computation; but what's computation?
- this course (TCS) studies abstract models of computation
  - finite state machines and regular expressions
    - REs in Unix/Linux tools, FSMs model protocols & circuits
  - context-free grammars and pushdown automata
    - programming language syntax, XML DTDs, natural language syntax
  - Turing machines -- (roughly) as powerful as real computers

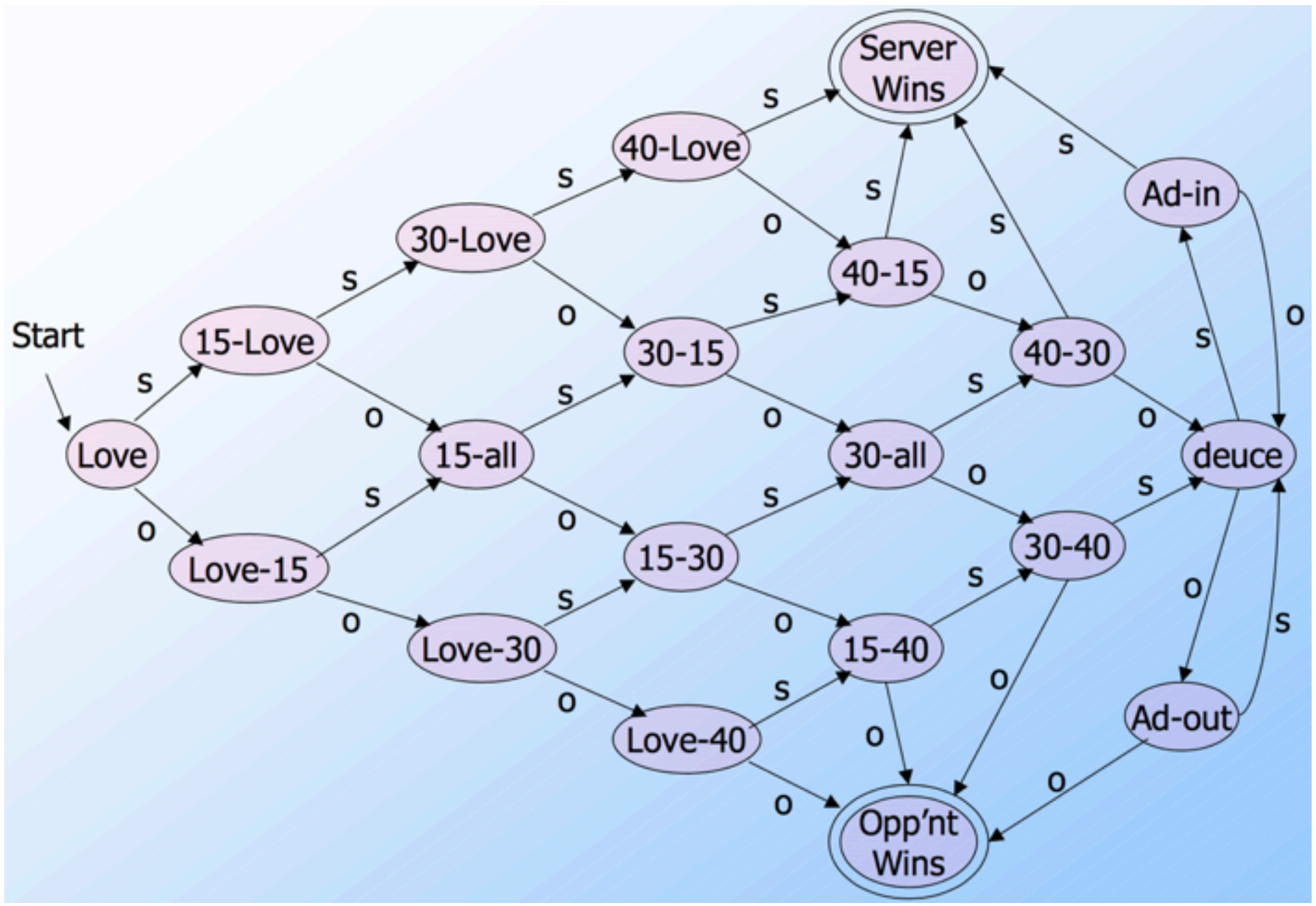
machine	language	grammar
finite-state	regular (RegExps)	regular
pushdown	context-free	context-free
Turing	recursively enumerable	unrestricted

Deterministic Finite-State Machines (DFA)

Sipser 1.1, Linz 2.1

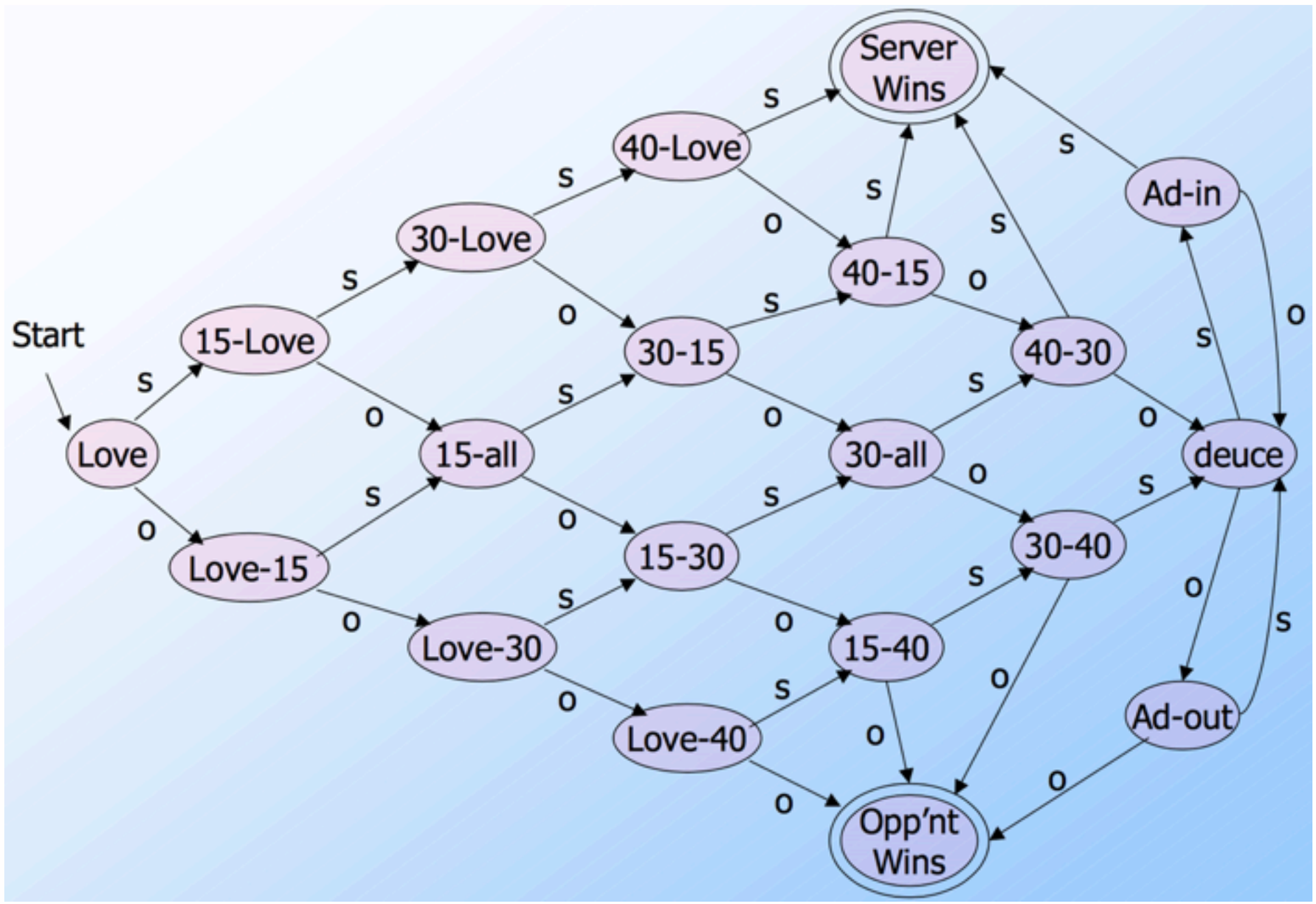
Finite Automata Example

- in Tennis, match = 3-5 sets; set = 6 or more games
- in one game, one person serves throughout
  - to win, you must win by at least 2 points



Finite Automata Example

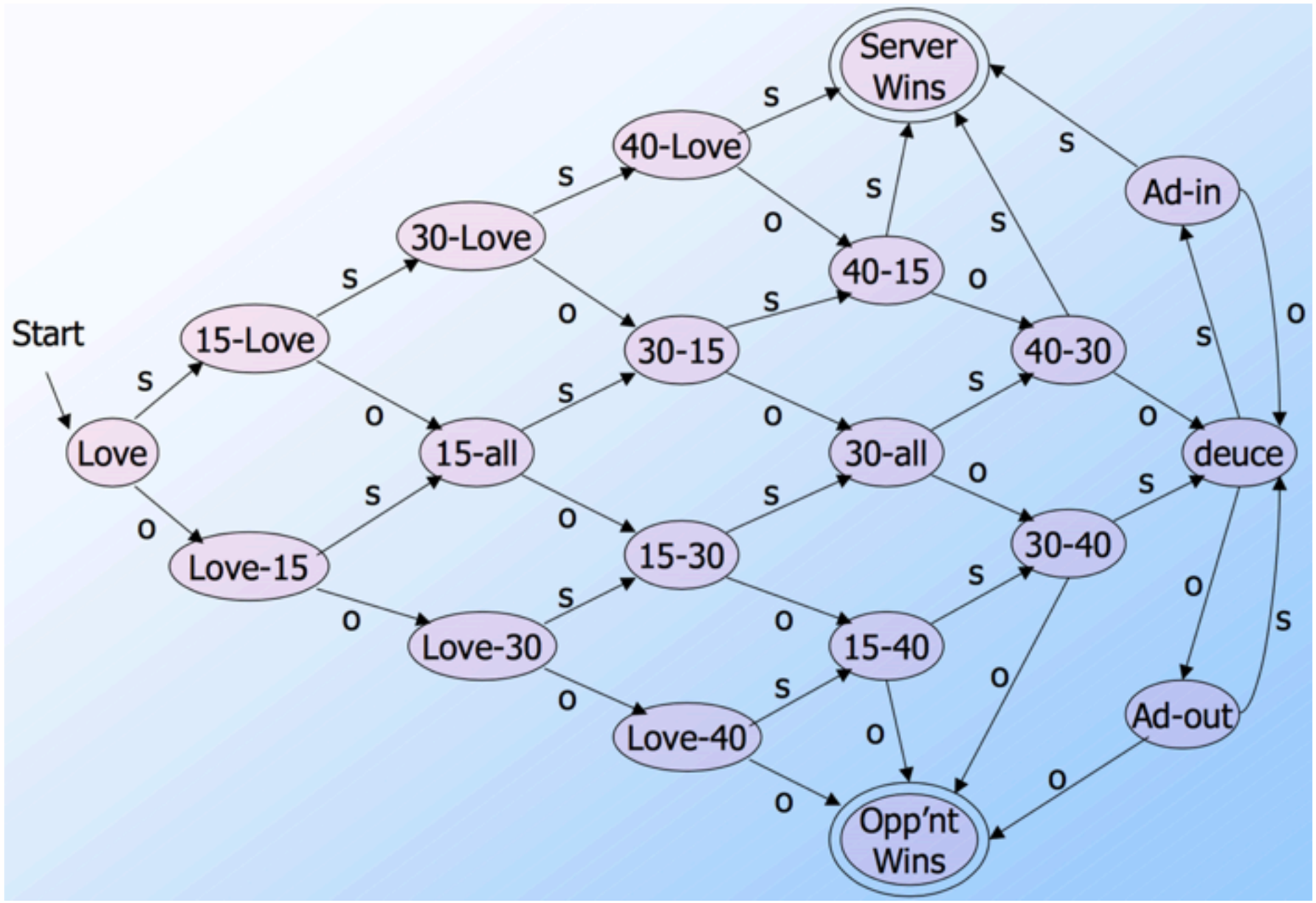
- states: encode status info. finite: only knows current state
  - state changes in response to inputs (s, o in this example)
  - transitions: rules for state change. *at state x, on input a, goto state y*



trap state not shown

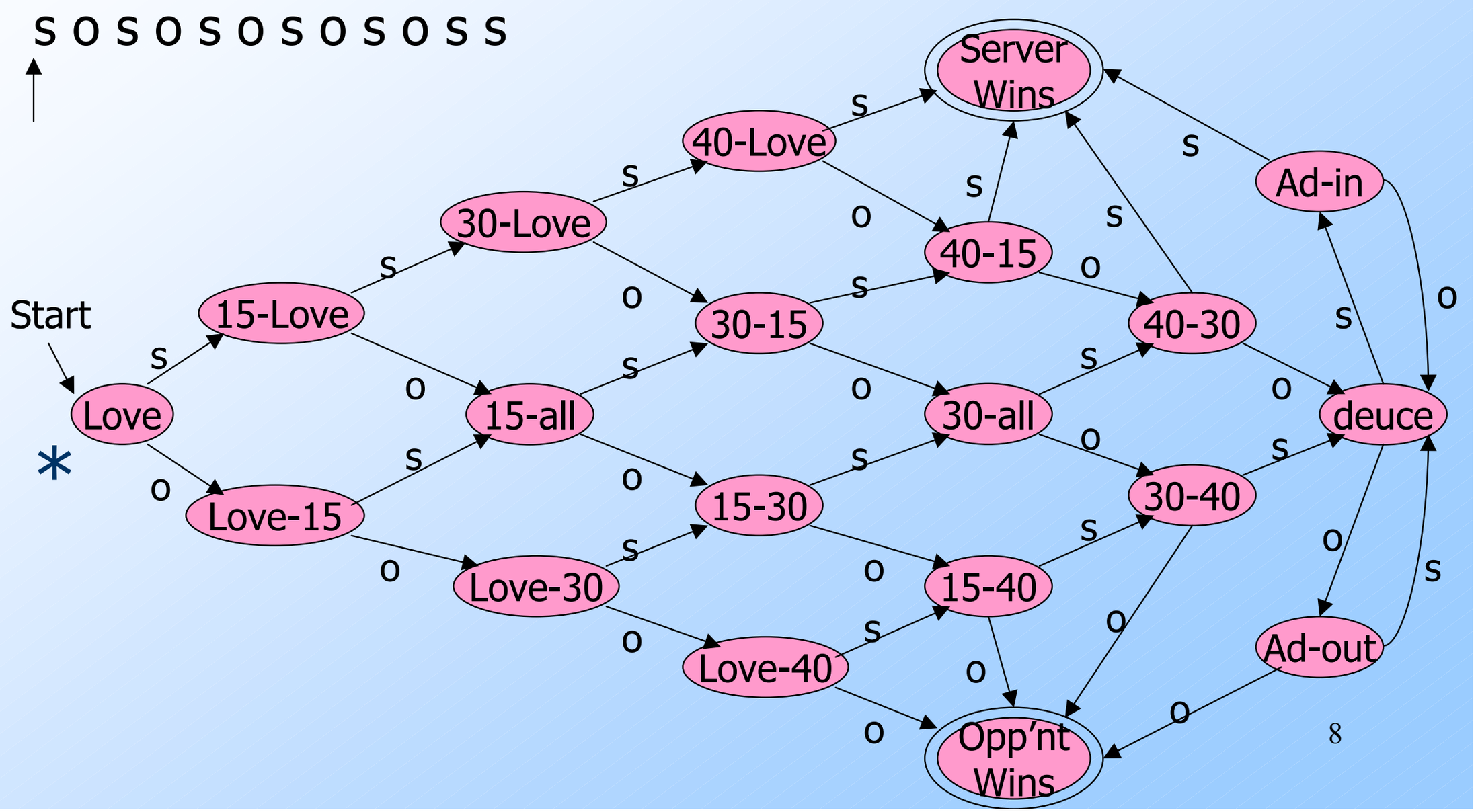
Acceptance/Rejection of Inputs

- given an input string, start at the start state and follow transitions
- input is accepted if you wind up in a final (accepting) state after all inputs have been read; otherwise rejected

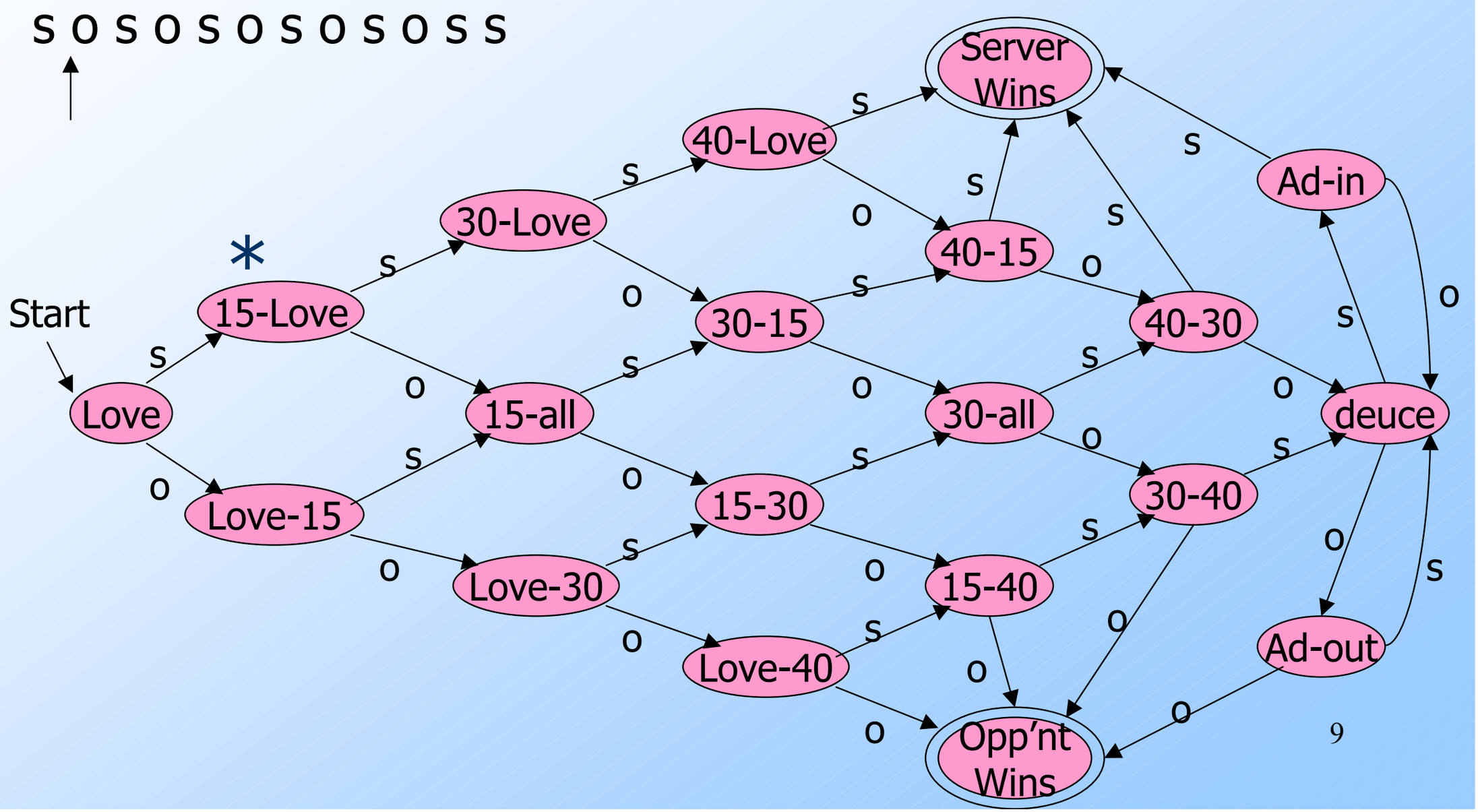


trap state not shown

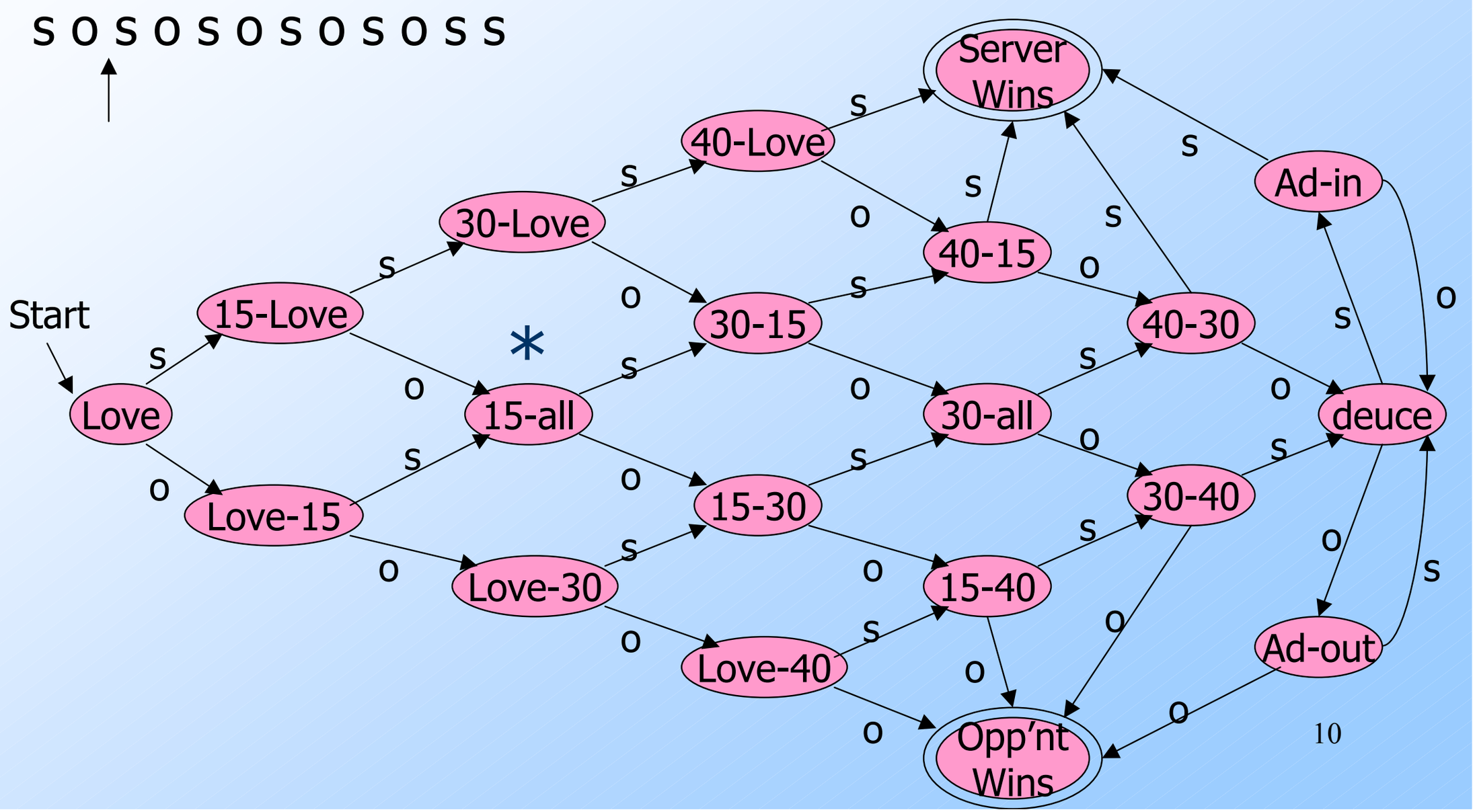
Example: Processing a String



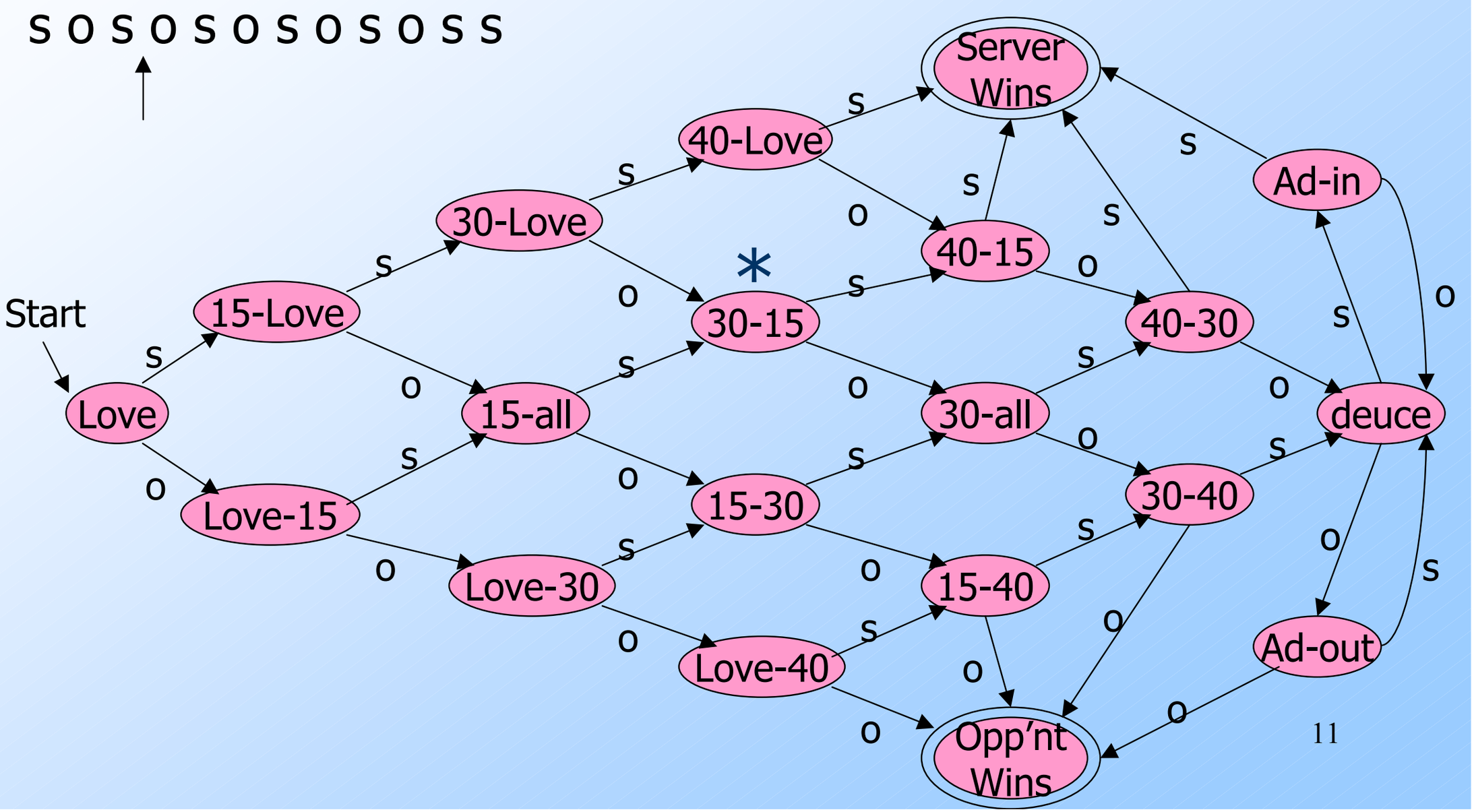
Example: Processing a String



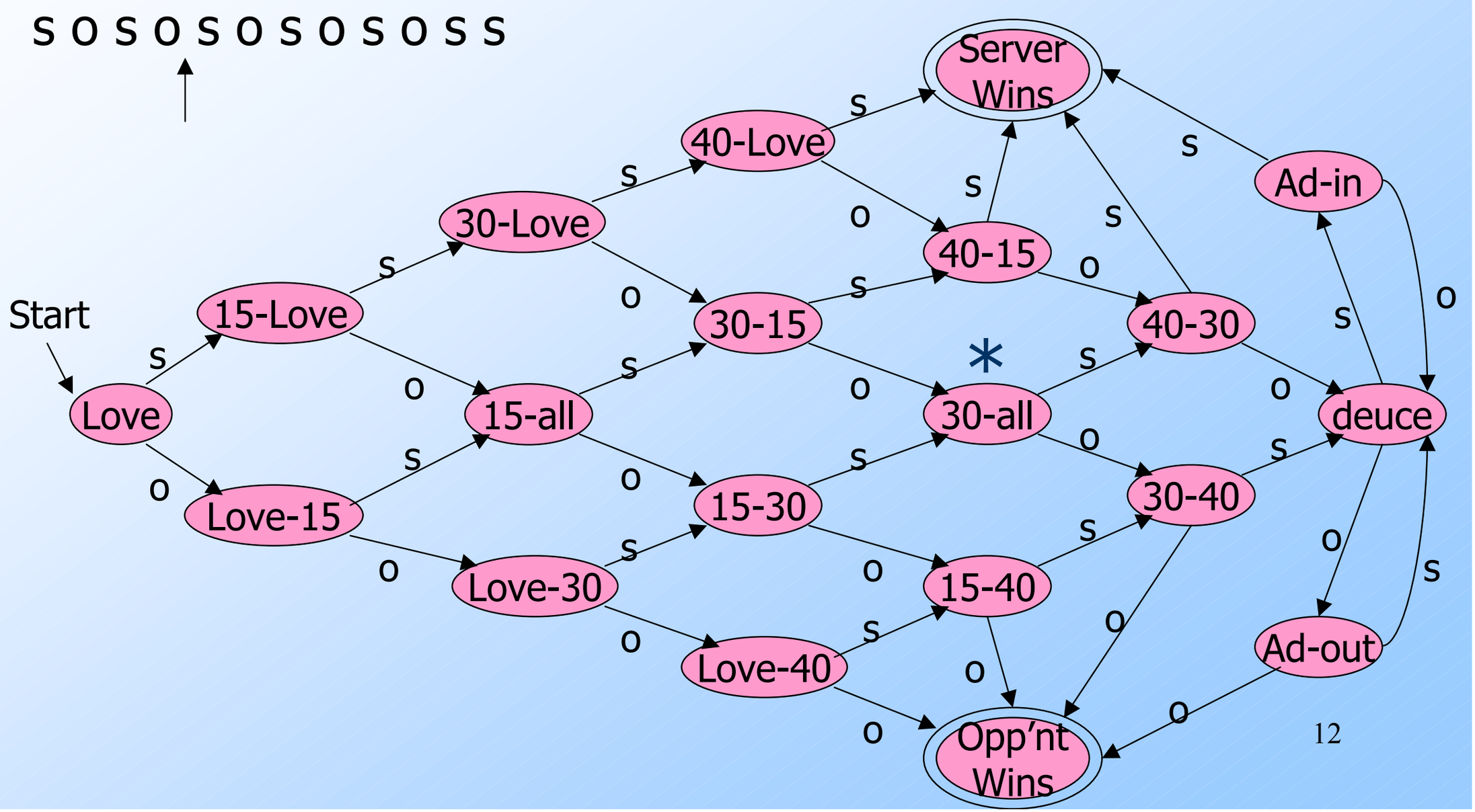
Example: Processing a String



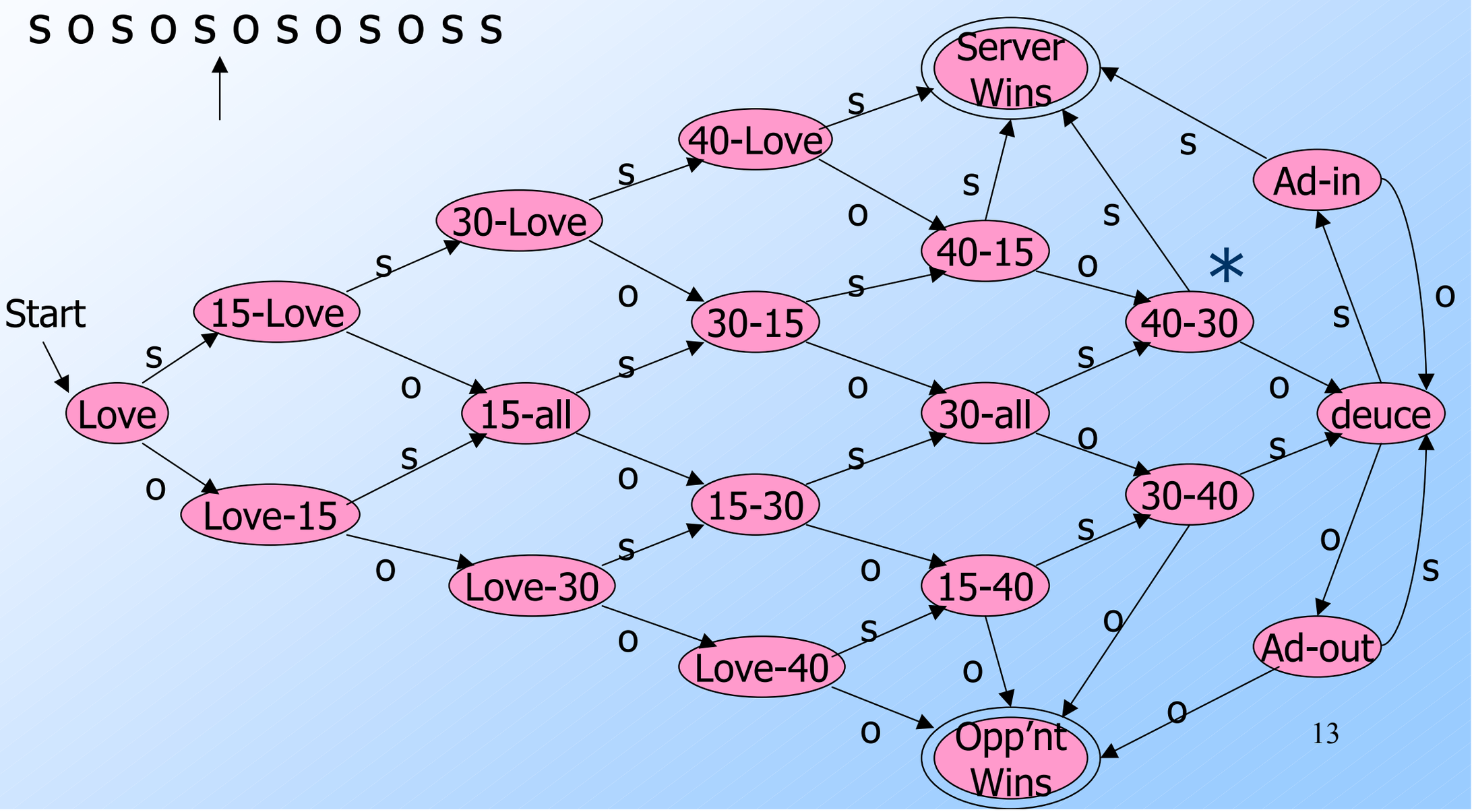
Example: Processing a String



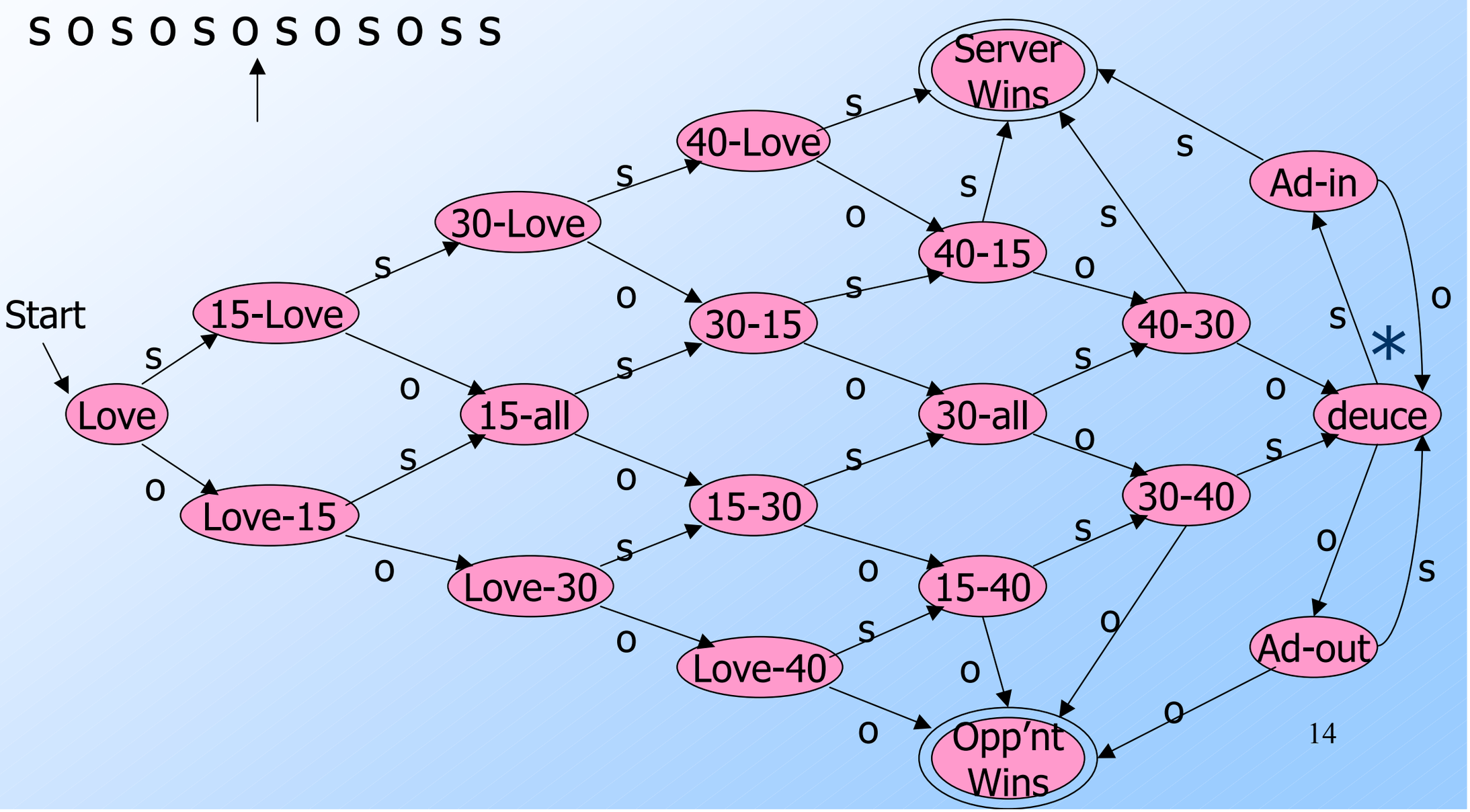
Example: Processing a String



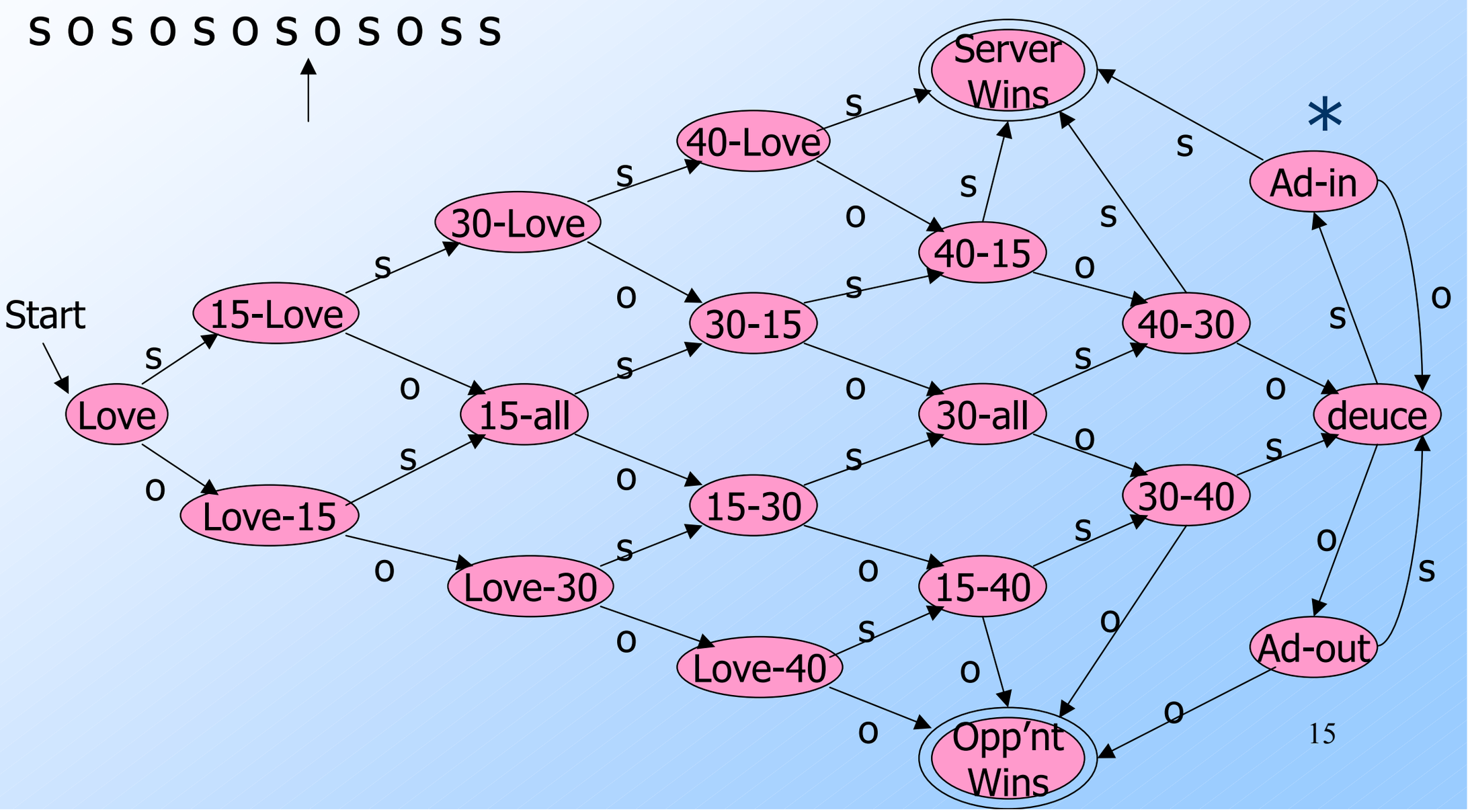
Example: Processing a String



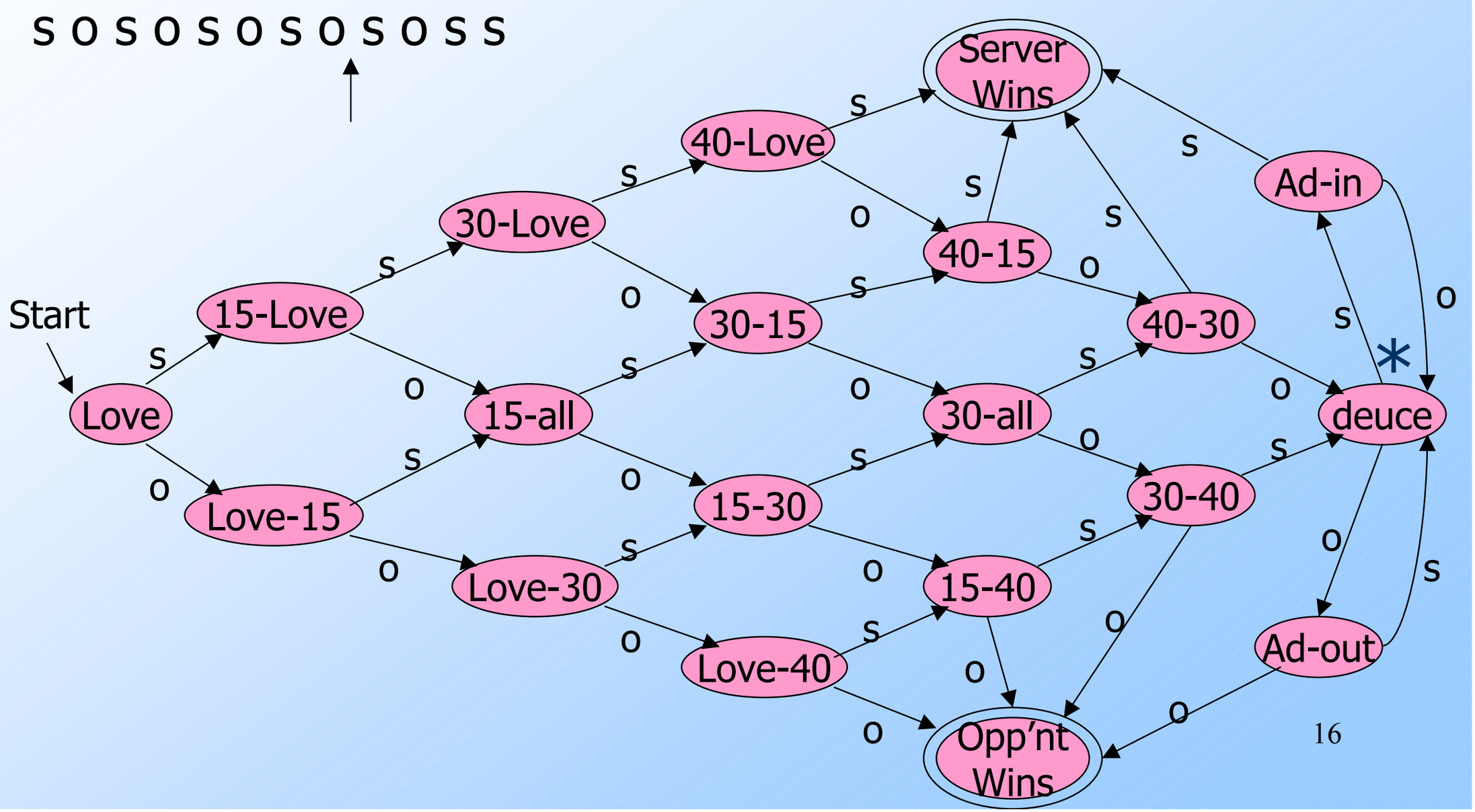
Example: Processing a String



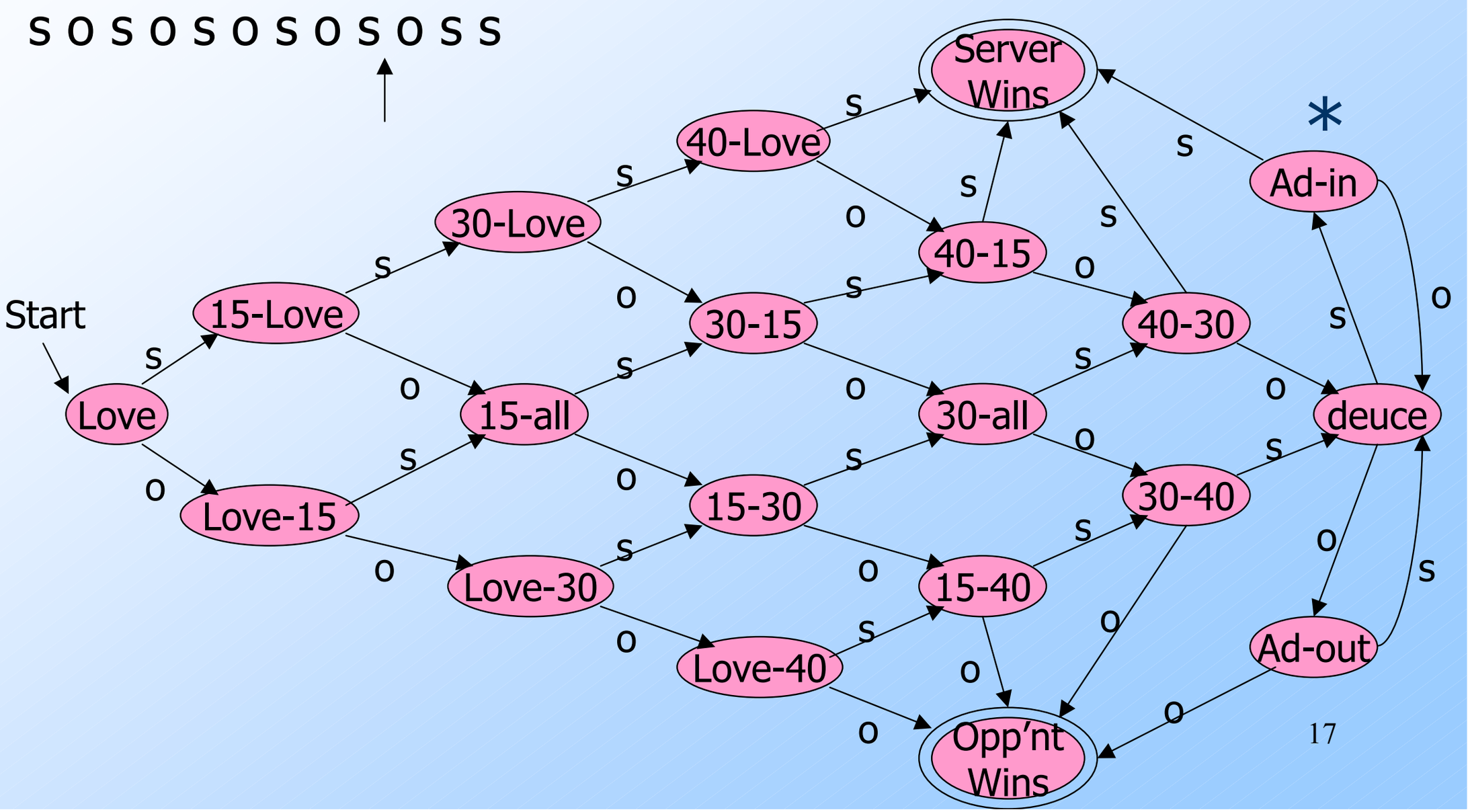
Example: Processing a String



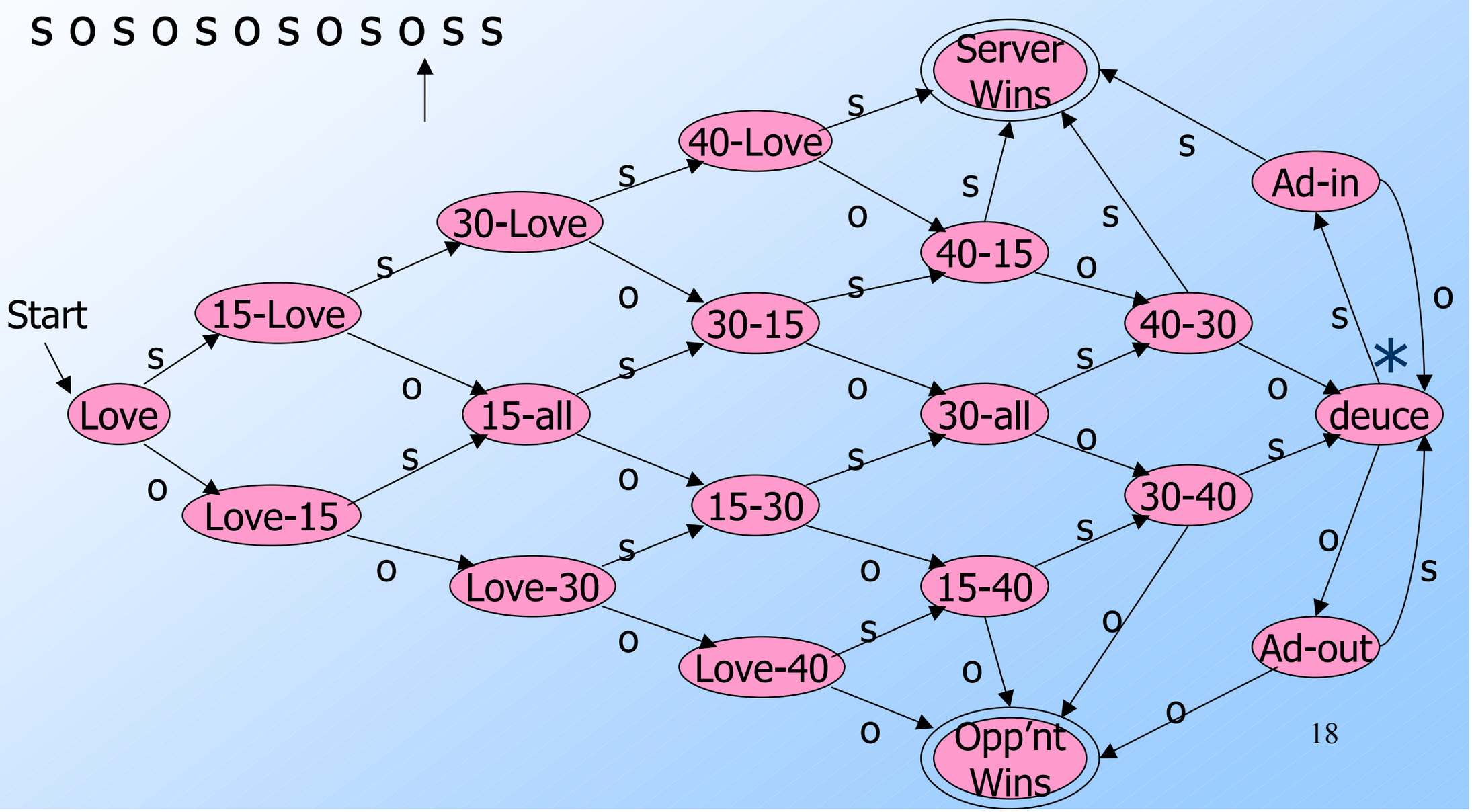
Example: Processing a String



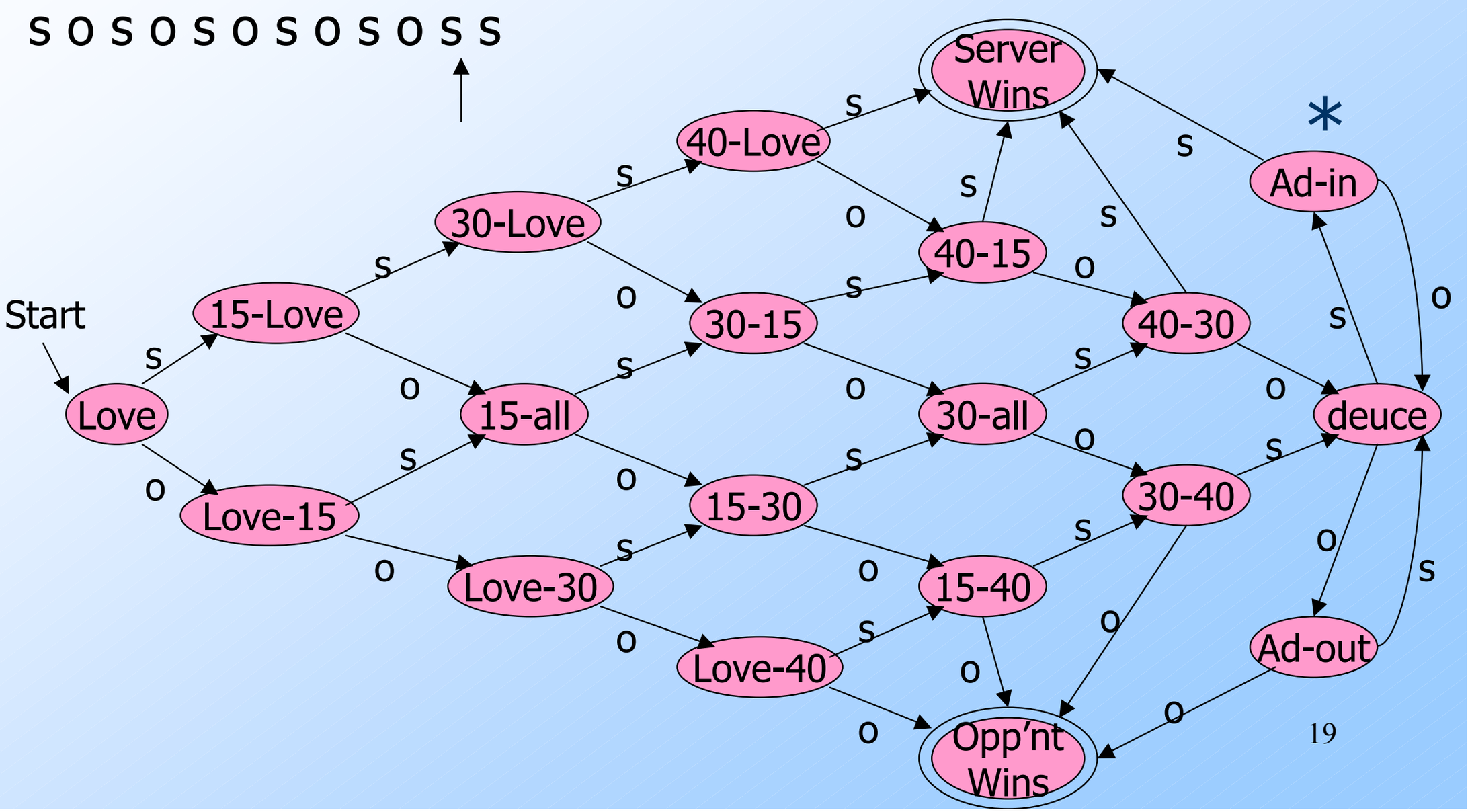
Example: Processing a String



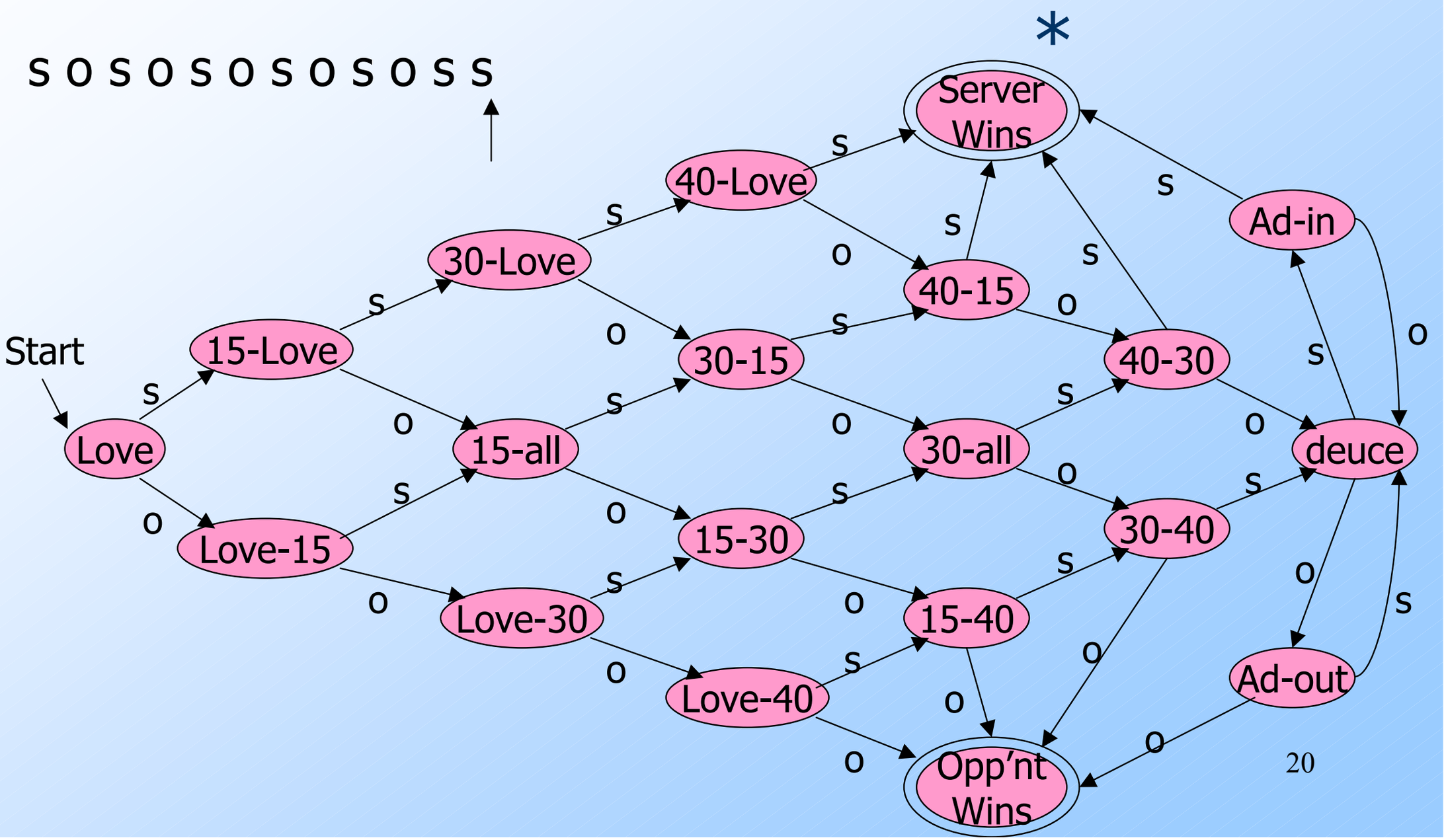
Example: Processing a String



Example: Processing a String



Example: Processing a String



Language of an Automaton

- the set of strings accepted by an automaton A is the language of A, denoted L(A)
- L(Tennis) = strings of {s, o} that determine the winner

STRINGS

Σ = Alphabet = Set of symbols.

$\Sigma = \{a, b, c, d\}$

(Always a finite set!)

STRING = A finite sequence of symbols.

baccada

ϵ = Empty string = Epsilon (also ε)

Length of a string

w = baccada

$|w| = |\text{baccada}| = 7$

$|\epsilon| = 0$

xy = Concatenation

x^R = Reverse of x

x = bac

y = cada

xy = baccad

x^R = cab

Language

"A language is a set of strings."

$L_1 = \{ab, bc, ac, dd\}$

$L_2 = \{\epsilon, ab, abab, ababab, \dots\}$

The empty string:

ϵ

The empty language:

$\{\} = \emptyset$

The language containing only ϵ:

$L_3 = \{\epsilon\}$

How to describe languages?

Enumeration $\{b, ac, da\}$

Regular Expressions $c(ab)^*(d|c)$

Context-Free Grammars

$S \to dTd$

$T \to TT \mid aTb \mid \epsilon$

Set Notation

$\{x \mid x \in \dots\}$

Construct Finite Automata for ...

- L1 = { aa, ab, ac, ..., ba, bb, ..., zz } (finite)
  - start state, final states

- L2 = *all letter sequences* (infinite)
  - (cycle)

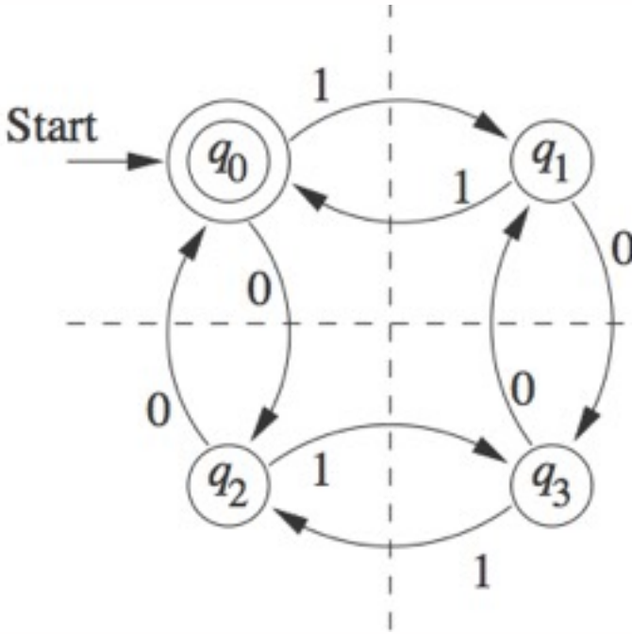
- L3 = *all alphanumeric strings that start with a letter*

Construct Finite Automata for ...

- L4 = *all letter strings with at least a vowel*

- L5 = *all letter strings with vowels in order*

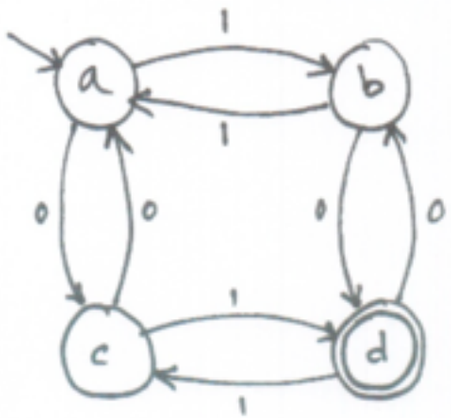
- L6 = *all 01 strings with even number of 0's and even number of 1's*



Formal Definition of DFA

Described by a 5-tuple:

$$M = (Q, \Sigma, \delta, q_0, F)$$



Q = Set of States

<u>Finite</u> Number of states

$Q = \{a, b, c, d\}$

Σ = Alphabet, a <u>Finite</u> Set of Symbols

$\Sigma = \{0, 1\}$

<u>by default, function means "total" function!</u>

δ = The TRANSITION FUNCTION

$$\delta : Q \times \Sigma \rightarrow Q$$

$q_0 = a$

$F = \{d\}$

$\delta =$

States	Symbol 0	Symbol 1
a	c	b
b	d	a
c	a	d
d	b	c

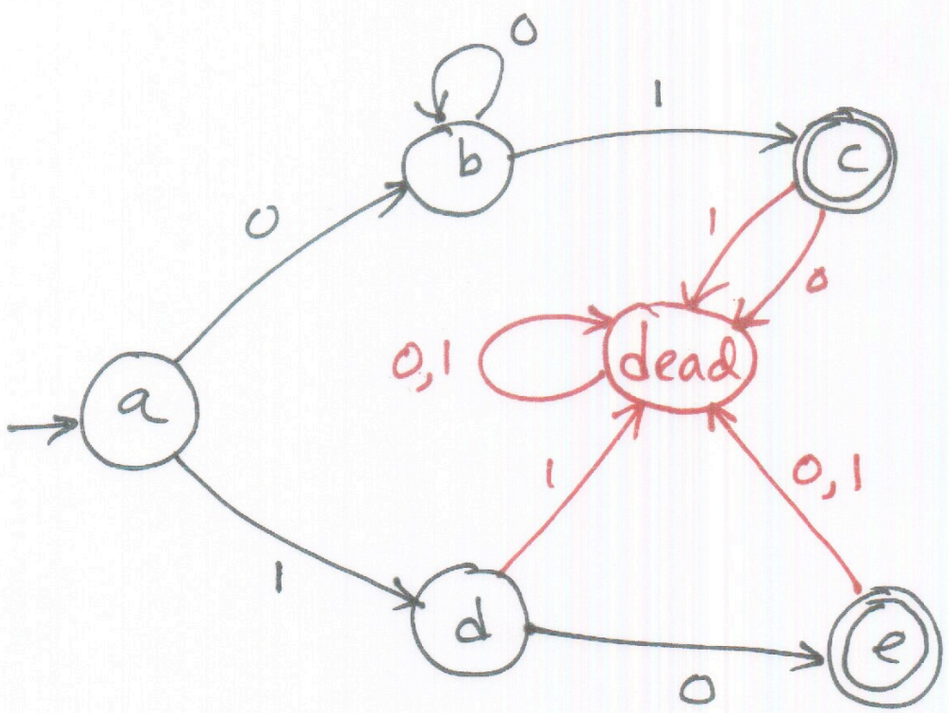
q_0 = The STARTING STATE

$q_0 \in Q$ (or "INITIAL" STATE)

F = The set of ACCEPT states

(or "FINAL" STATES) $F \subseteq Q$

Alternative Definition of Transition



if we change the transition function δ from a total function to a *partial* function then we don't need to include trap state because whenever $\delta(s, a)$ is undefined, it goes to the trap state.

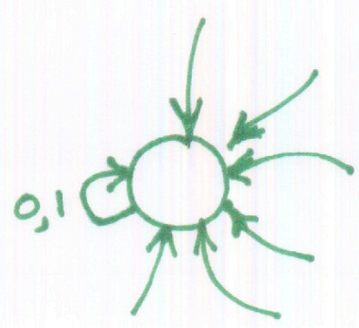
δ is a function (total)

FORMALLY, must be defined

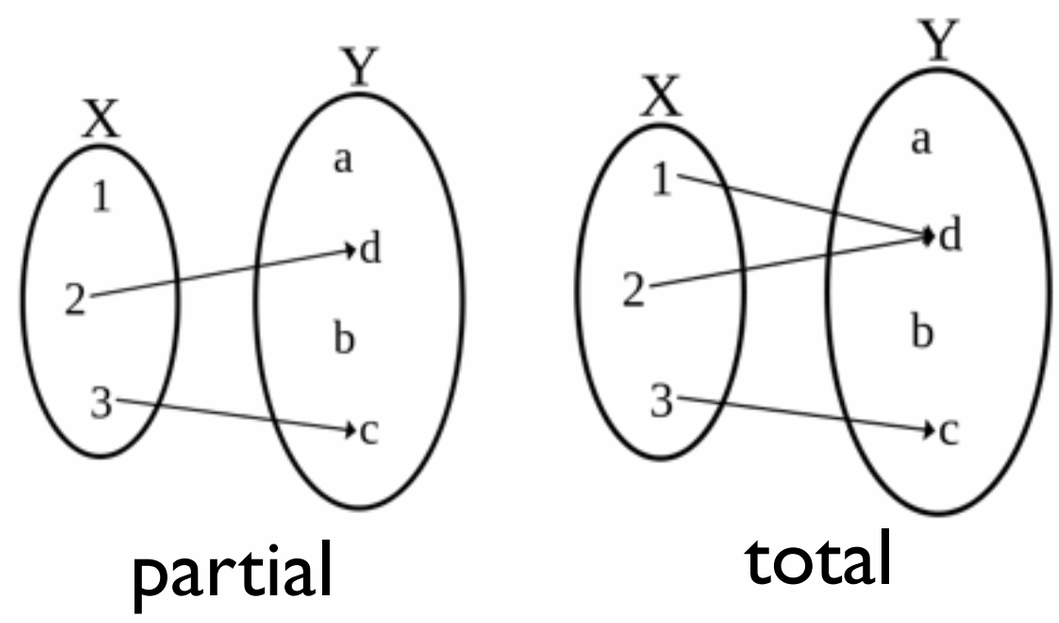
$\delta(c, 1) = ?$

If some transitions are missing, add a dead state.

(Often, prefer not to show dead state.)



note that the definition of computation remains unchanged.



https://en.wikipedia.org/wiki/Partial_function

Formal Definition of Computation

Let $M = (Q, \Sigma, \delta, q_0, F)$

Let $w = w_1 w_2 \cdots w_N$ be a string where $w_i \in \Sigma$

M accepts w iff. there is a sequence of states $r_0, r_1, r_2, \ldots, r_N$ in Q

such that

$r_0 = q_0$

$\delta(r_i, w_{i+1}) = r_{i+1}$ for $0 \leq i \leq N$

$r_N \in F$

We say...

M "recognizes" Language A if $A = \{ w \mid M \text{ accepts } w \}$

Redefine computation: Extend δ to δ^*

Defining the computation of an FA $M=(Q,\Sigma,q_0,A,\delta)$.

$\delta(q,a) =$ which state to go to from q after reading a

$\delta^*(q,s) =$ which state we end in after starting in q and reading the string s

e.g. aabba

Extended transition function $\delta^* : Q \times \Sigma^* \to Q$:

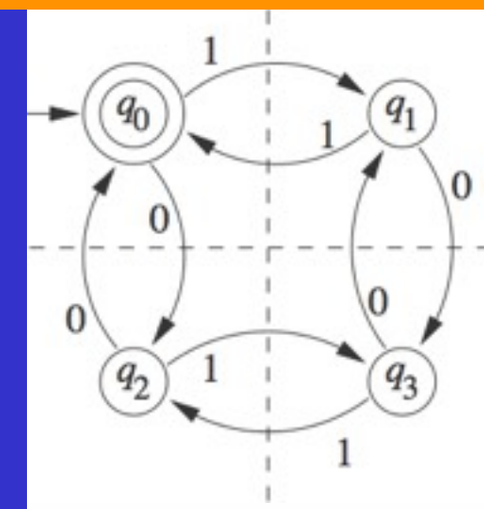
a string

1) For every $q \in Q$, let $\delta^*(q,\Lambda) = q$

2) For every $q \in Q$, $y \in \Sigma^*$, and $\sigma \in \Sigma$, let $\delta^*(q,y\sigma)$

$= \delta(\underbrace{\delta^*(q,y)}_{r},\sigma)$

We say that a string $x \in \Sigma^*$ is **accepted by M** iff $\delta^*(q_0,x) \in A$.



$\delta^*(q_0, 10) = q_3$
$\delta^*(q_3, 111) = q_2$
$\delta^*(q_2, \varepsilon) = q_2$
$\delta^*(q_1, 1) = q_0$

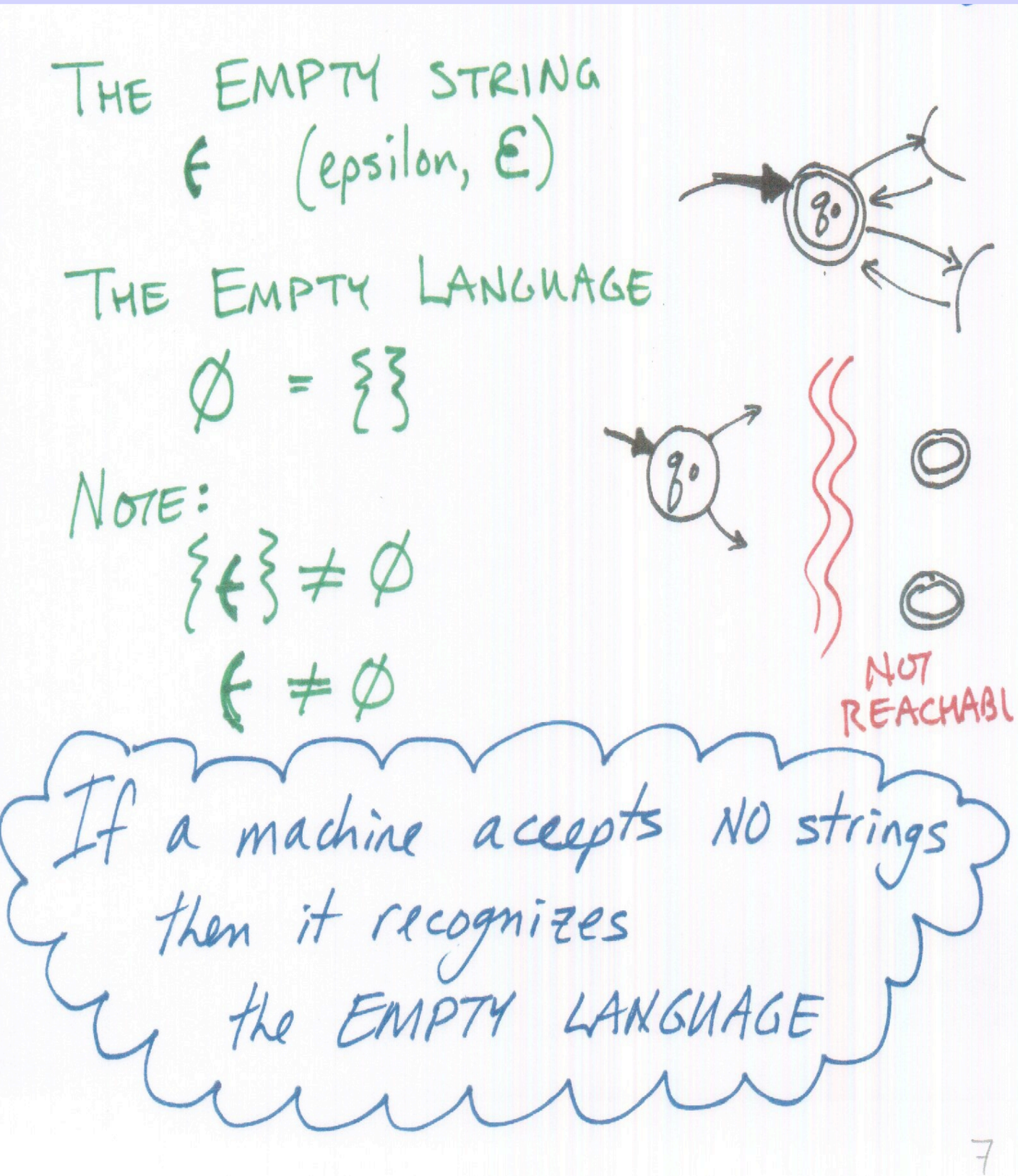
$$\delta^*(q,w) = \begin{cases} \delta(\delta^*(q,x),a) & \text{where } w = xa \text{ and } x \in \Sigma^*, a \in \Sigma \\ q & \text{where } w = \epsilon \end{cases}$$

$$L((Q,\Sigma,\delta,q_0,F)) = \{w \in \Sigma^* \mid \delta^*(q_0,w) \in F\}$$

Q1: what about decomposing w = ax or even w = xy?
Q2: what about partial function δ?

δ^* is not defined in Sipser, but is in all other textbooks. This is probably one of the biggest flaws of Sipser book.

Language, String, Machine

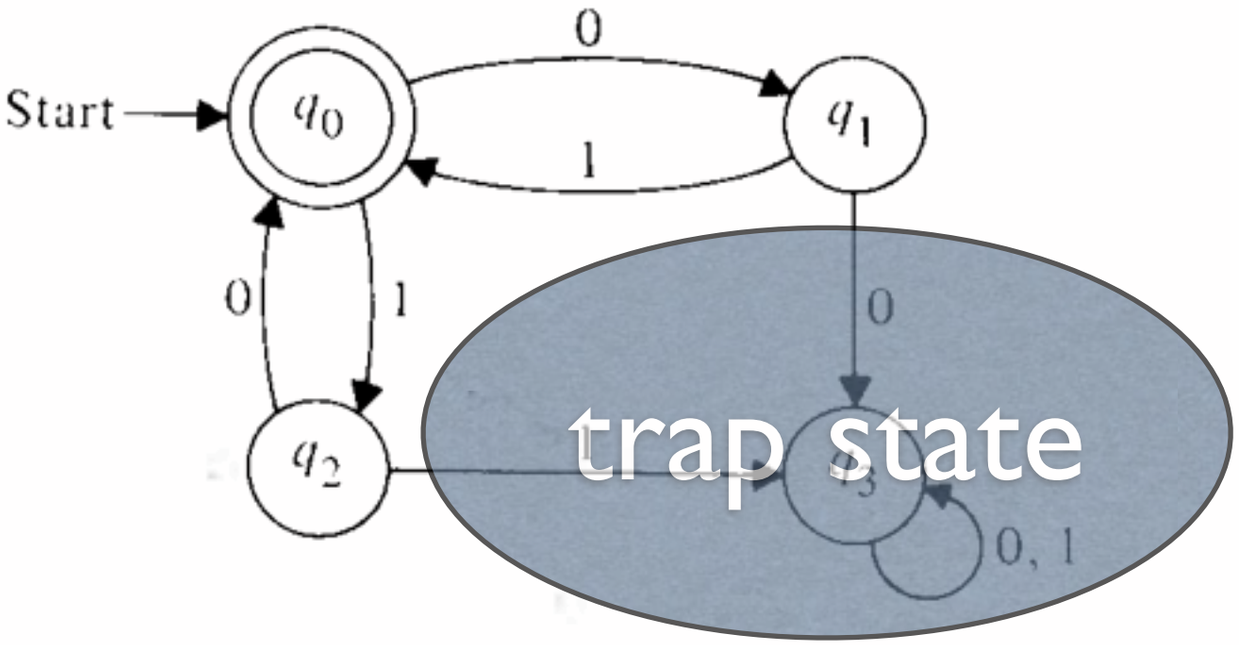


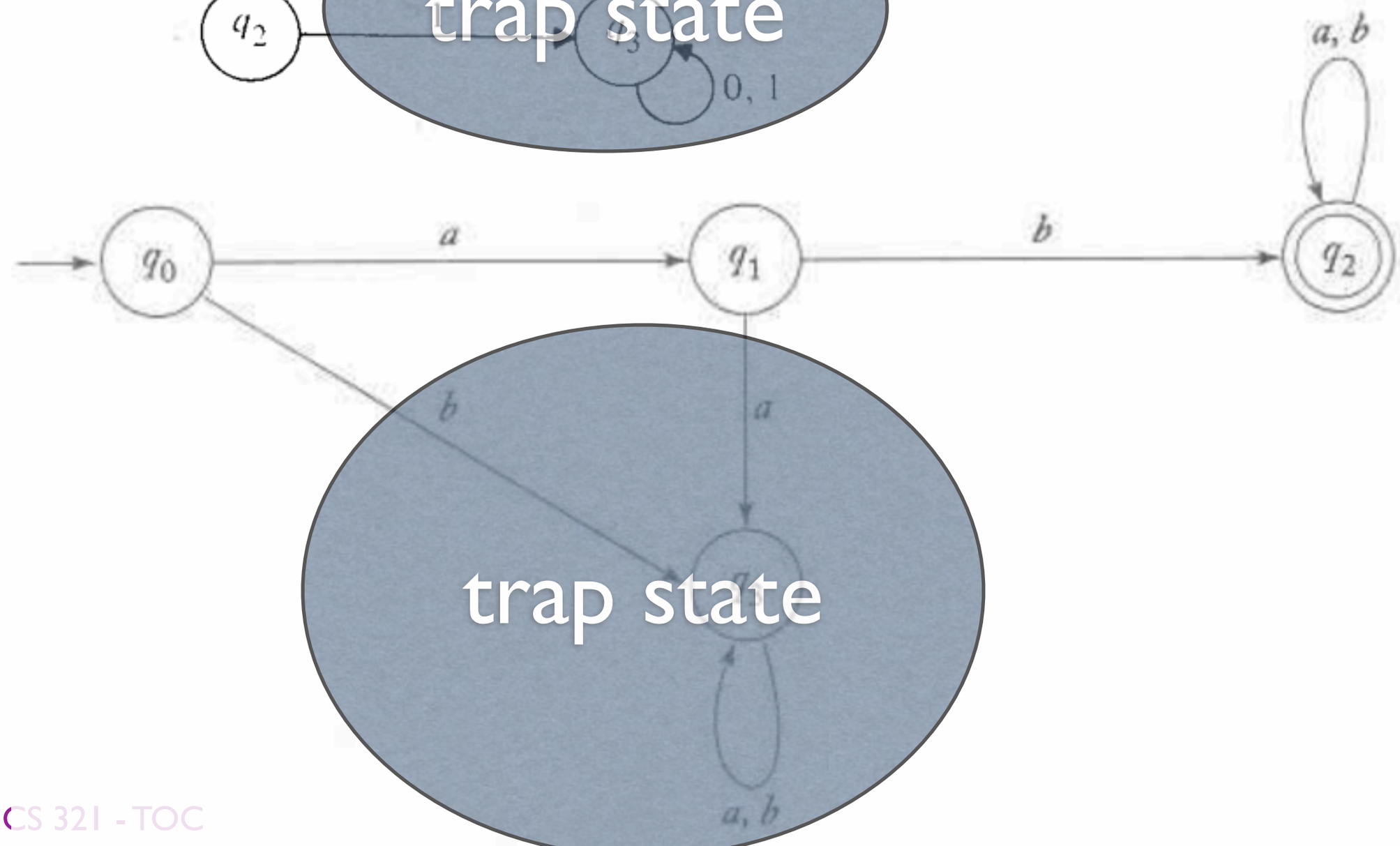
does M accept any string?

does M accept empty string?

does M recognizes the empty language?

What's the language of ...





Divisible by 3

Q1: language over {a,b,c} s.t. the # of a's is divisible by 3

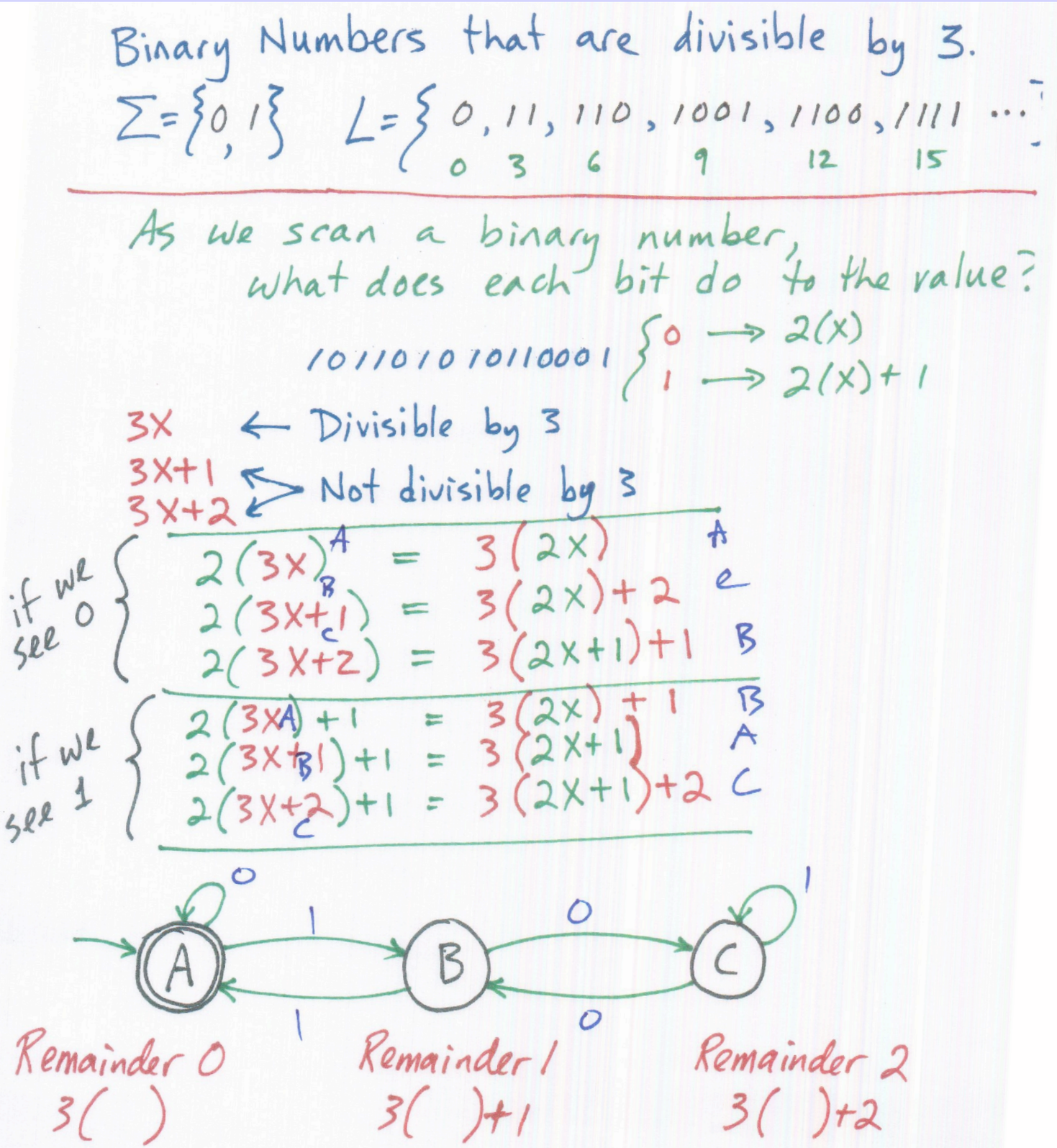
hint: need 3 states:
state 0: (#a's) % 3 == 0
state 1: (#a's) % 3 == 1
state 2: (#a's) % 3 == 2

wait... what about b/c?

Q2: language over {a,b,c} s.t. (the # of a's) - (the # of b's) is divisible by 3

hint: still 3 states:
state 0: ((#a's) - (#b's)) % 3 == 0
state 1: ((#a's) - (#b's)) % 3 == 1
state 2: ((#a's) - (#b's)) % 3 == 2

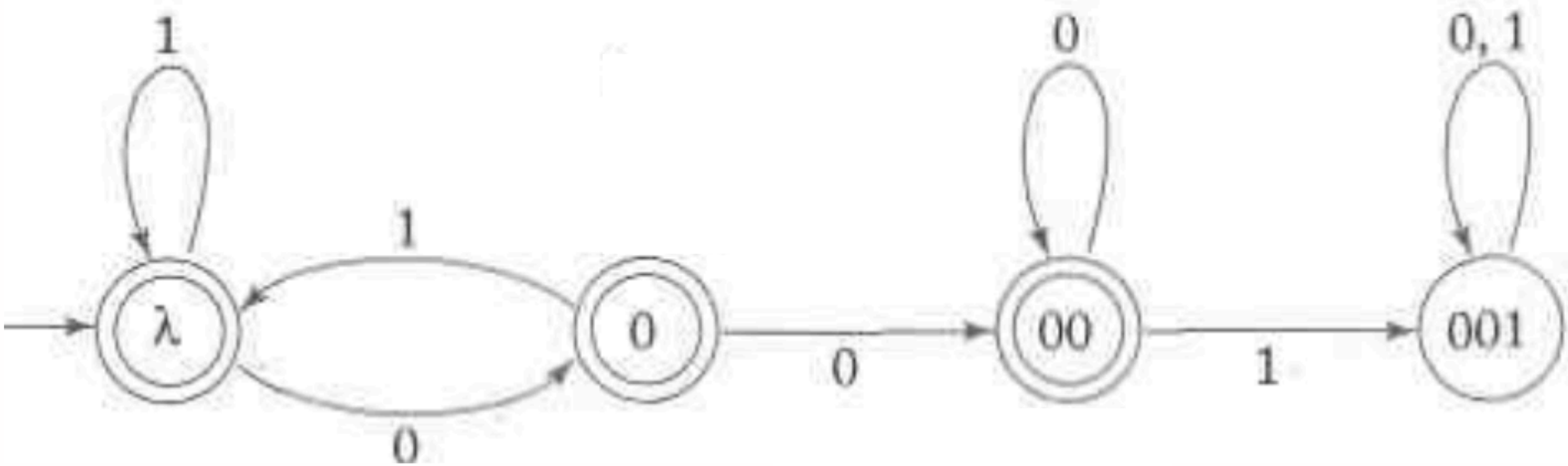
Binary Number Divisible by 3



- still just need 3 states
- decimal num divisible by 3?
  - again, 3 states
- general strategy in base *d*
  - divisible by *n* => *n* states
  - [0] [1] ... [*n*-1]
    - [*i*] = { *x* | *x* % *n* = *i* }
  - at state [*q*], on digit *i*
    - goto state [(*q* x *d* + *i*) % *n*]
- but it's possible to use *less* than *n* states!

Construct Finite Automaton for ...

- any string that does not contain 001 in it
- try simpler problem: a string that does contain 001

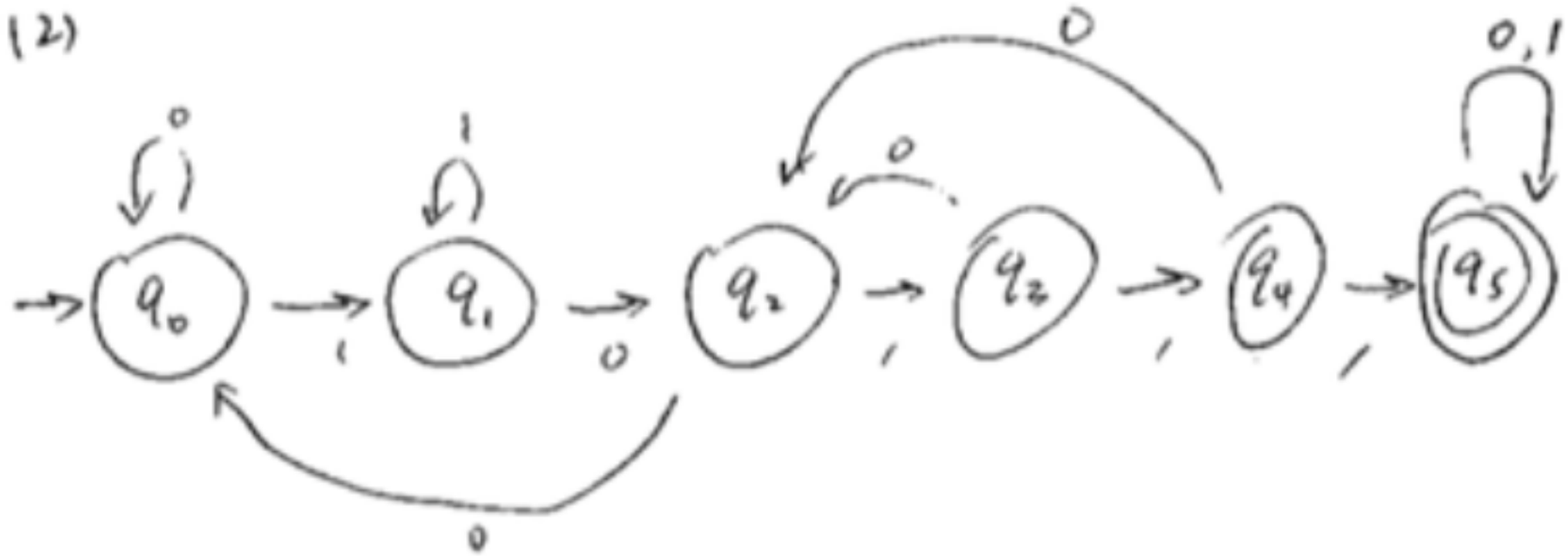


this reminds you of KMP string matching

https://www.ics.uci.edu/~eppstein/161/960222.html

Construct Finite Automaton for ...

- any string that does contain 10111 in it



general strategy for "contains pattern $a_0a_1...a_{n-1}$":

backbone: at state i on a_i goto state $i+1$ ($n+1$ states)

deviations: at state i, on any input b != a_i

go back to the rightmost such state j,

where prefix $a_0a_1...a_{j-1}$ == suffix $a_{i-j+1}...a_{i-1}b$

because we can reuse such suffix and

don't need to restart from the very beginning

https://www.ics.uci.edu/~eppstein/161/960222.html

Complement Language

Notation

$L(M_1)$ = The language that M_1 recognizes.
= The set of strings over $\{0,1\}^*$ that contain 0011 as a substring.

$L(M_2)$ = The set of strings over $\{0,1\}^*$ that do not contain 0011.

COMPLIMENTING A LANGUAGE

They are sets, after all.

$$L(M_1) = \overline{L(M_2)}$$

The "Universe"
All possible strings made with symbols from the alphabet.

$\Sigma = \{0,1\}$ Universe $= \{0,1\}^*$

Set compliment is always relative to some Universe; (implicitly).

$\overline{L} \triangleq \Sigma^* \setminus L = \{x \mid x \in \Sigma^*, x \notin L\}$

if $L = L(M)$ and $M = (Q, \Sigma, \delta, q, F)$ then

$\overline{L} = L(\overline{M})$ where $\overline{M} = (Q, \Sigma, \delta, q, \overline{F})$

where $\overline{F} = Q \setminus F$

just flip final and non-final!

Prove: Complement of Regular is Regular

Proof: For every regular language L, by definition of regular language, there must be a DFA M s.t. $L(M) = L$. let $M = (Q, \Sigma, \delta, q_0, F)$ where δ is a **total** function, let us construct another DFA $\overline{M} = (Q, \Sigma, \delta, q_0, \overline{F})$ where $\overline{F} = Q \backslash F$.

For every string $w \in L$, it will end up in a state $q \in F$ in M, and it will end up in the same state in $\overline{M}$ which rejects w since $q \notin \overline{F}$; similarly, for every string w' in the complement language, i.e., $w \in \Sigma^* \setminus F$, it will end up in a state $q' \notin F$ in M, and it will end up in the same state in $\overline{M}$ which accepts w since $q' \in \overline{F}$. So $\overline{M}$ accepts all strings in $\Sigma^* \setminus L$ **and only those**, which means the complement language $\Sigma^* \setminus L$ is recognized by DFA $\overline{M}$, thus regular. $\square$

Note, however, that if δ is a partial function (i.e., trap state omitted), this proof does not work (why?). You would have to add a trap state and all trap transitions to make δ a total function first.

Rewrite/Simplify using δ^* notation

Proof: For every regular language L, by the definition of regular language, there must be a DFA M s.t. $L(M) = L$. let $M = (Q, \Sigma, \delta, q_0, F)$ where δ is a **total** function, we construct another DFA $\overline{M} = (Q, \Sigma, \delta, q_0, \overline{F})$ where $\overline{F} = Q \backslash F$.

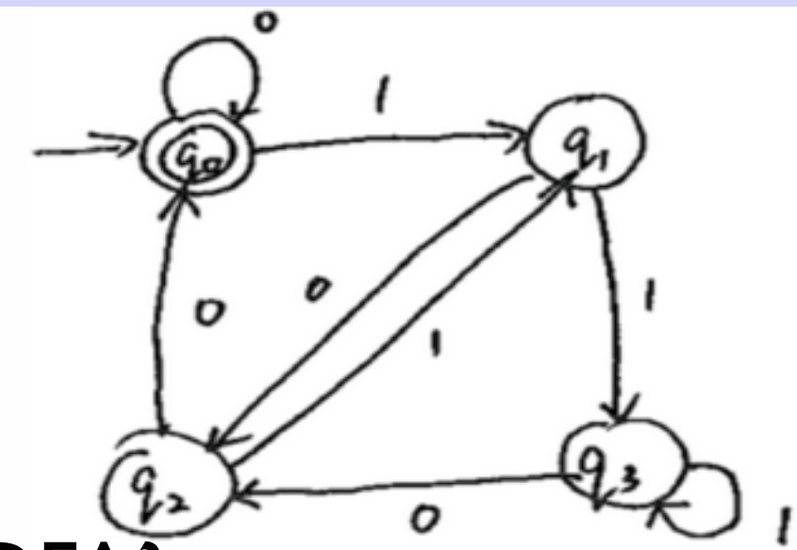
For every string $w \in L$, there exists $q \in Q$ s.t. $\delta^*(q_0, w) = q \in F$ in M, and $\delta^*(q_0, w) = q \notin \overline{F}$ in $\overline{M}$ which rejects w; similarly, for every string w' in the complement language, i.e., $w' \in \Sigma^* \setminus F$, there exists $q' \in Q$ s.t. $\delta^*(q_0, w') = q' \notin F$ in M, and $\delta^*(q_0, w') = q' \in \overline{F}$ in $\overline{M}$ which accepts w. So $\overline{M}$ accepts all strings in $\Sigma^* \setminus L$ and **nothing else**, which means the complement language $\Sigma^* \setminus L$ is recognized by DFA $\overline{M}$, thus regular. $\square$

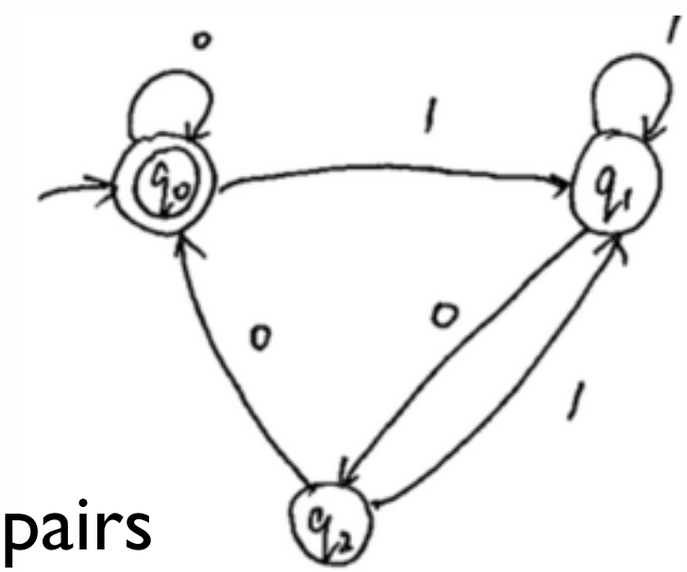
$$\boxed{\delta^*(q, w) = \begin{cases} \delta(\delta^*(q, x), a) & \text{where } w = xa \text{ and } x \in \Sigma^*, a \in \Sigma \\ q & \text{where } w = \epsilon \end{cases}}$$

$$\boxed{L((Q, \Sigma, \delta, q_0, F)) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}}$$

Binary Number Divisible by 4

- 4-state solution (trivial)
- 3-state solution (merge q1 w/ q3)
- in general, how do you:
  - reduce a DFA to a smaller but equivalent DFA?
    - see Linz 2.4 or Sipser problem 7.42 (p. 327)
    - will discuss later after NFA
  - test if two DFAs are equivalent?
    - follow pairs of states, check if all visited state-pairs agree on finality (both accept or both reject)
    - $O(n^2 \Sigma)$ time and space

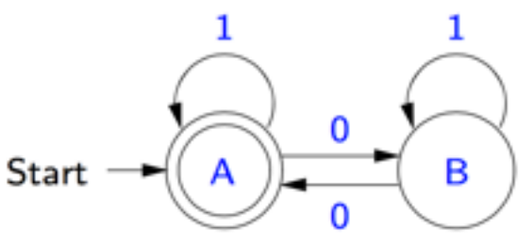


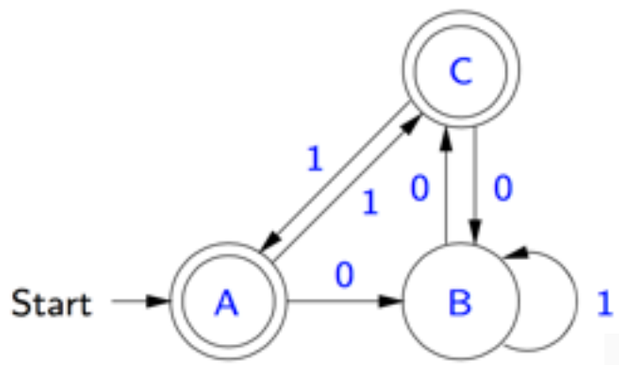


https://www.cse.iitb.ac.in/~trivedi/courses/cs208-spring14/lec05.pdf

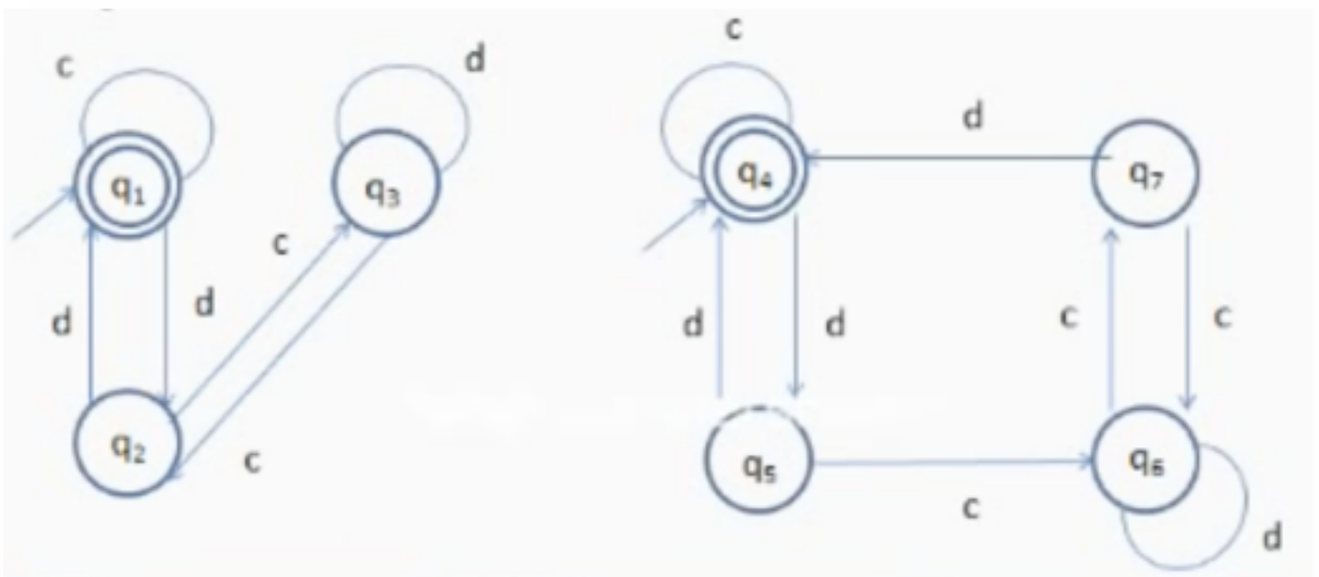
Test if two DFAs are equivalent

- traverse all state-pairs and make sure each pair agrees on "finality" (both accept or both reject)





pair	final?	on 0	on 1
(A, A)	(y, y)	(B, B)	(A, C)
(B, B)	(n, n)	(A, C)	(B, B)
(A, C)	(y, y)	(B, B)	(A, A)
no new pairs found			



Q Q'	Q_c Q_c'	Q_d Q_d'
q_1 , q_4	q_1 , q_4	q_2 , q_5
q_2 , q_5	q_3 , q_6	q_1 , q_4
q_3 , q_6	q_2 , q_7	q_3 , q_6
q_2 , q_7	q_3 , q_6	q_1 , q_4

Inductive Proofs

Sipser 0.4, Linz 1.1-1.2

Proof by Induction (Linz 1.1-1.2, Sipser 0.4)

- Theorem to prove: $|uv| = |u| + |v|$
- first define string length rigorously and inductively:
  - ~~$|a| = 1$~~, $|\varepsilon| = 0$, $|wa| = |w| + 1$
- now prove the Theorem by induction on $|v|$
  - base case: $|v| = 0$, then $v = \varepsilon$, so $|uv|=|u|=|u|+0=|u|+|v|$
  - inductive case: assume Theorem holds for any $|v|$ of length $0..n$ (IH)
    Now take any v of length $n+1$. Let $v = wa$, then $|v| = |w|+1$ (by def.)
    - then $|uv| = |uwa| = |uw|+1$ (by definition)
    - by induction hypothesis (applicable to any w of length n)
      - $|uw| = |u|+|w|$, so that $|uv| = |u|+|w|+1 = |u|+|v|$

HW: prove by induction on $|u|$

What's wrong with this proof?

Theorem(?!): All horses are the same color.

Proof: Let $P(n)$ be the predicate "in all non-empty collections of n horses, all the horses are the same color." We show that $P(n)$ holds for all n by induction on n (using 1 as the base case).

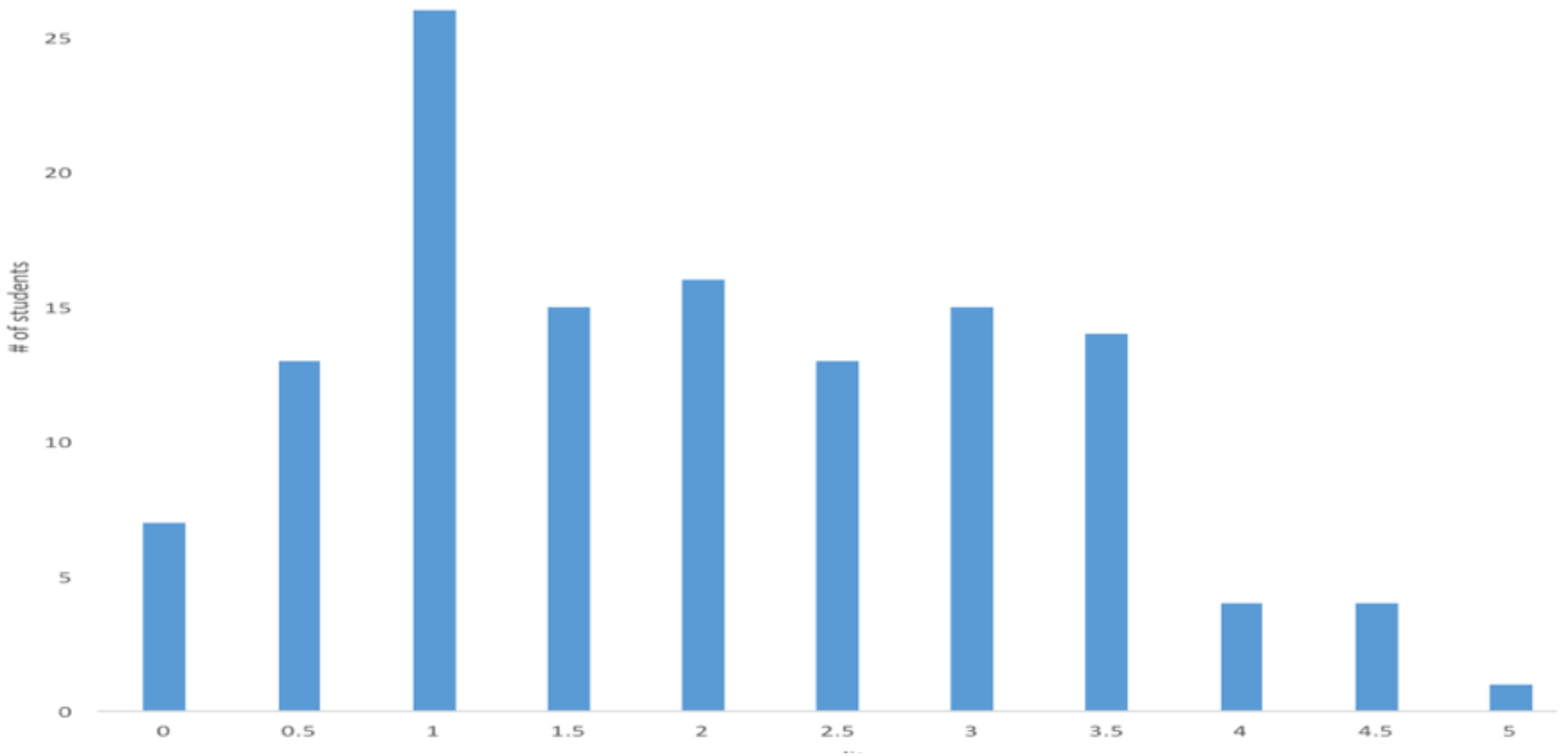
Base case: Clearly, $P(1)$ holds.

Induction case: Given $P(n)$, we must show $P(n+1)$.

Consider an arbitrary collection of $n+1$ horses. Remove one horse temporarily. Now we have n horses and hence, by the induction hypothesis, these n horses are all the same color. Now call the exiled horse back and send a different horse away. Again, we have a collection of n horses, which, by the induction hypothesis, are all the same color. Moreover, these n horses are the same color as the first collection. Thus, the horse we brought back was the same color as the second horse we sent away, and all the $n+1$ horses are the same color.

Quiz 1 scores and Projected Final Grade

- mean: 2.0, median: 2



projected final grade	0 0.5	1	1.5	2	2.5	3	3.5	4 4.5 5
	F	C/C-	C+	B-	B	B+	A-	A

Regular Language and Regular Operations

Sipser 1.1, Linz 2.1

Regular Language

DEFINITION

A language is a **REGULAR LANGUAGE** iff some Finite State Machine recognizes it.

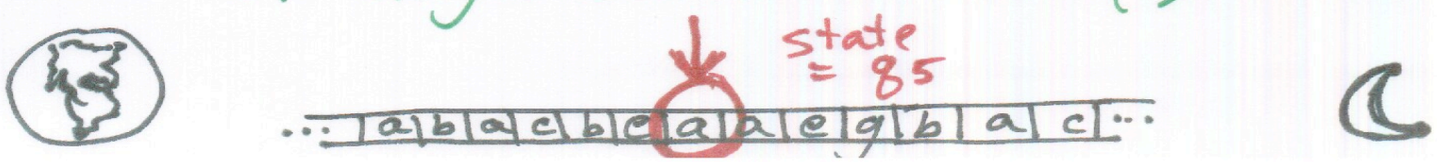
What languages are NOT regular?
Anything that requires memory.
The F.S.M. memory is <u>very</u> limited
Cannot store the string.
Cannot "count."

Not Regular:

ww $\quad \underbrace{01101}_{w}\underbrace{01101}_{w}$

$0^N 1^N$ $\quad \underbrace{000000}_{6}\underbrace{111111}_{6}$

Imagine a String from here to the Moon. You are trying to recognize it. Your only memory:
A single small number (i, # of states)



Regular Operations

UNION

$$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$$

CONCATENATION

$$A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$$

STAR "closure"

$$A^* = \{x_1 x_2 \cdots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$$

EXAMPLE

$$\Sigma = \{a, b, c, \ldots z\}$$

$$A = \{aa, b\}$$

$$B = \{x, yy\}$$

$$A \cup B = \{aa, b, x, yy\}$$

$$A \circ B = \{aax, aayy, bx, byy\}$$

$$A^* = \{\epsilon, aa, b, aaaa, aab, baa, bb, aaaaaa, aaaab, aabaa, aabb, \ldots$$

- other operations:
  - intersection
  - complement
  - difference
- regular languages are <u>closed</u> under all these operations

Union

THEOREM

The class of Regular Languages is CLOSED under UNION.

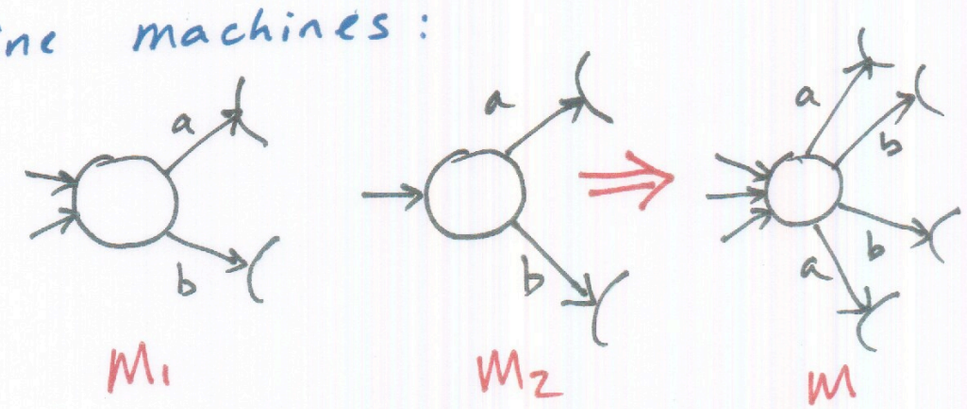
If L_1 and L_2 are regular languages then so is $L_1 \cup L_2$.

PROOF (BY CONSTRUCTION)

Assume $L_1 = L(M_1)$
$L_2 = L(M_2)$

Build M to recognize $L_1 \cup L_2$.

Combine machines:



WHOOPS!
Not a F.S.M.!

- the union of regular languages is still regular

Union by Cross-Product Construction

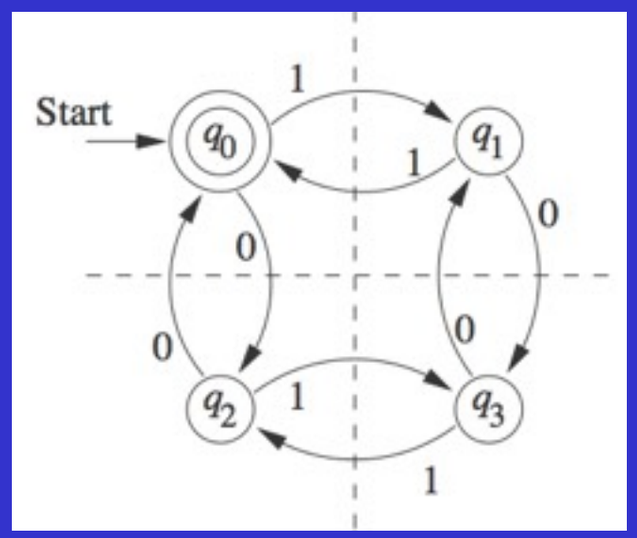
Let $M_1 = (Q_1,\Sigma,q_1,A_1,\delta_1)$ and $M_2 = (Q_2,\Sigma,q_2,A_2,\delta_2)$ be two FA's.

Union:

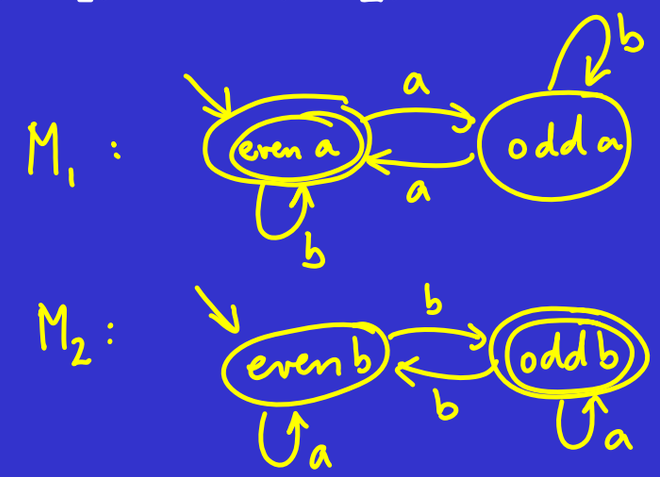
We know that $L(M_1) \cup L(M_2)$ is regular.
Construct an FA M such that $L(M) = L(M_1) \cup L(M_2)$.

Construction:

see next slide



example:



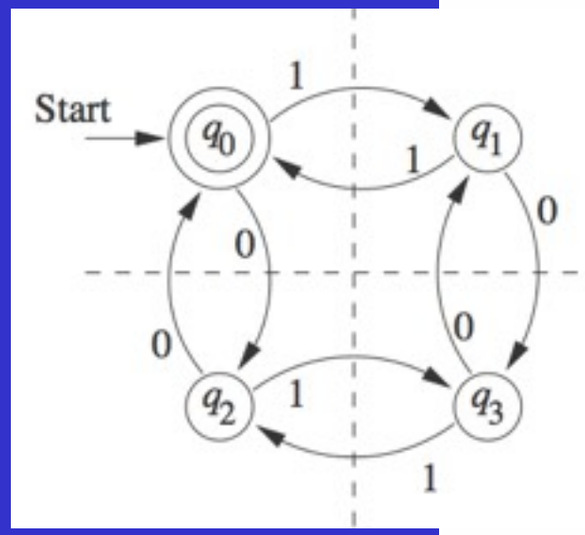
$L(M_1) \cup L(M_2)$ contains the strings over $\{a,b\}$ s.t. the string has an even # a's <u>or</u> an odd # b's

$aabbb \in L(M_1) \cup L(M_2)$

see Sipser 1.1, Linz

Intersection by Cross-Product Construct.

Let $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$ and $M_2 = (Q_2, \Sigma, q_2, A_2, \delta_2)$ be two FA's.



Intersection:

Is $L(M_1) \cap L(M_2)$ is regular ?
Construct an FA M such that $L(M) = L(M_1) \cap L(M_2)$.

$M = (Q, \Sigma, q_0, A, \delta)$ where

$Q = Q_1 \times Q_2$

$q_0 = (q_1, q_2)$

$A = A_1 \times A_2$

$\delta((p,q), \sigma) = (\delta_1(p,\sigma), \delta_2(q,\sigma))$

$\forall p \in Q_1 \ \forall q \in Q_2 \ \forall \sigma \in \Sigma$

IDEA:

automata "walk" together,

new states = pair of states,
one from M_1, one from M_2

see the "even a, odd b" example
a few slides back

for union: $A = A_1 \times Q_2 \cup Q_1 \times A_2$

Complement and Difference

Let $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$ be an FA.

Complement:

Is $L(M_1)'$ regular ?
Construct an FA M such that $L(M) = L(M_1)'$.

Idea: switch accepting for non-accepting states in M_1

$M = (Q, \Sigma, q_0, A, \delta)$ where $Q = Q_1$
$q_0 = q_1$
$\delta = \delta_1$
$A = A_1' = Q_1 - A_1$ ✓

Let $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$ and $M_2 = (Q_2, \Sigma, q_2, A_2, \delta_2)$ be two FA's.

Difference:

Construct an FA M such that $L(M) = L(M_1) - L(M_2)$.

Summary: Union, Intersection, Difference

- "cross-product" construction (also used in table-filling algorithm)

$$M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$$
$$M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$$
$$M = (Q_1 \times Q_2, \Sigma, \delta, (q_{01}, q_{02}), F)$$
$$\delta : (Q_1 \times Q_2) \times \Sigma \mapsto Q_1 \times Q_2,$$
$$\delta((p,q),a) = (\delta_1(p,a), \delta_2(q,a))$$
$$F_\cap = F_1 \times F_2$$
$$F_\cup = F_1 \times Q_2 \cup Q_1 \times F_2$$
$$F_- = F_1 \times \bar{F}_2$$

LaTeX source:

```
M_1 & = (Q_1, \Sigma, \delta_1, q_{01}, F_1)\\
M_2 & = (Q_2, \Sigma, \delta_2, q_{02}, F_2)\\
M & = (Q_1 \times Q_2, \Sigma, \delta, (q_{01}, q_{02}), F)\\
\delta & : (Q_1 \times Q_2) \times \Sigma \mapsto Q_1 \times Q_2, \\
& \delta  ((p,q), a) = (\delta_1(p,a), \delta_2(q,a))\\
F_\cap & = F_1 \times F_2   \\
F_\cup & = F_1 \times Q_2 \cup Q_1 \times F_2 \\
F_- & = F_1 \times \bar{F_2} \\
\\
\delta^* & : (Q_1 \times Q_2) \times \Sigma^* \mapsto Q_1 \times Q_2, \\
\delta^* & ((p,q), w) = \begin{cases}
                          (p,q) & w = \epsilon \\
                          \delta(\delta^*((p,q), x), a)  & w = xa
                        \end{cases}\\
&\text{alternatively, }\\
\delta^* & ((p,q), w) = (\delta_1^*(p, w), \delta_2^*(q, w))
```

$$\delta^* : (Q_1 \times Q_2) \times \Sigma^* \mapsto Q_1 \times Q_2,$$
$$\delta^*((p,q),w) = \begin{cases} (p,q) & w = \epsilon \\ \delta(\delta^*((p,q),x),a) & w = xa \end{cases}$$

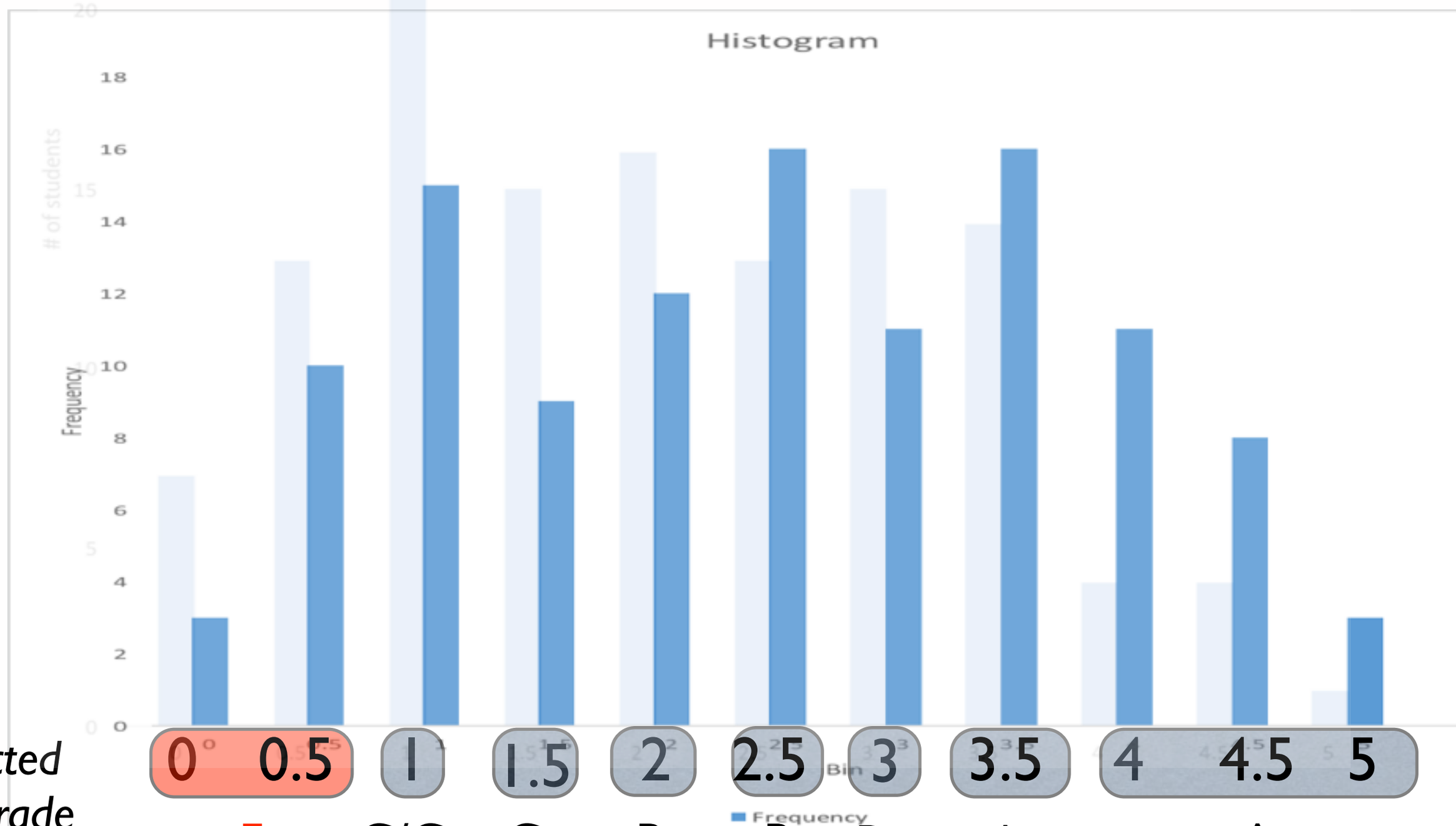
alternatively,

$$\delta^*((p,q),w) = (\delta_1^*(p,w), \delta_2^*(q,w))$$

HW: prove equivalence

Quiz 2 much better than Quiz 1

- quiz 2 mean: 2.47; quiz 1 mean: 2.0



projected final grade

F C/C- C+ B- B B+ A- A

Non-Deterministic Finite-State Machines (NFA)

Sipser 1.2, Linz 2.2-2.3

Concatenation - NFA

THEOREM

THE CLASS OF REGULAR LANGUAGES IS CLOSED UNDER CONCATENATION.

If L_1 and L_2 are regular then so is $L_1 \circ L_2$.

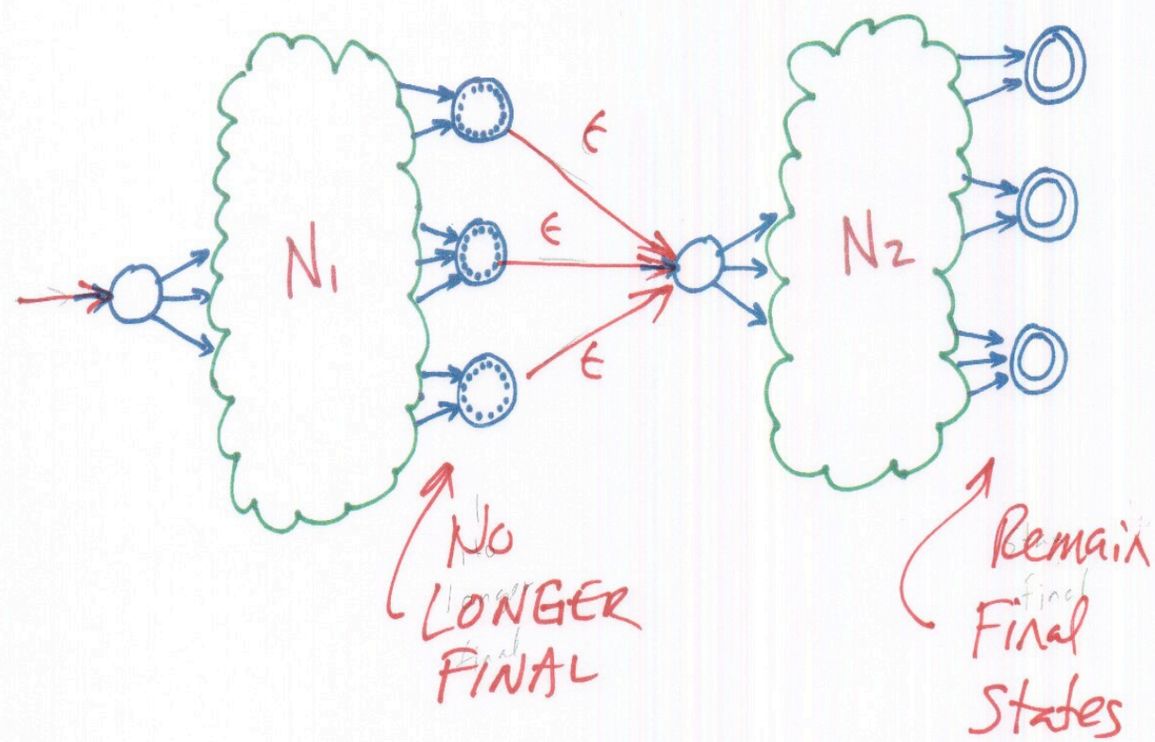
PROOF

CAN'T DO IT YET.

WE NEED...

NONDETERMINISM

- concatenation of regular languages is still regular



Non-Determinism

"GIVEN THE CURRENT STATE, THERE MAY BE MULTIPLE NEXT STATES."

- The next state is chosen at random.
- All next states are chosen in parallel and pursued simultaneously.

Same Thing

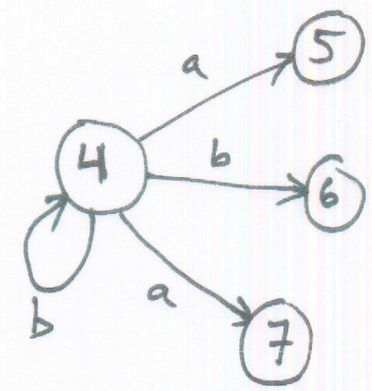
FSM means Deterministic Finite State Automaton/Machine

DFA means DETERMINISTIC FINITE STATE AUTOMATON

NFA means NONDETERMINISTIC FINITE STATE AUTOMATON

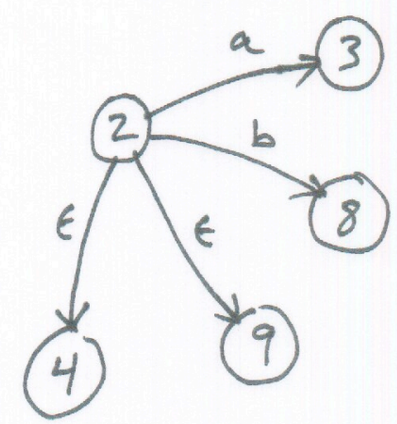
NOW WE WILL ALLOW

- MULTIPLE EDGES WITH THE SAME LABEL OUT OF A NODE.



Which edge should you take???

- EPSILON EDGES



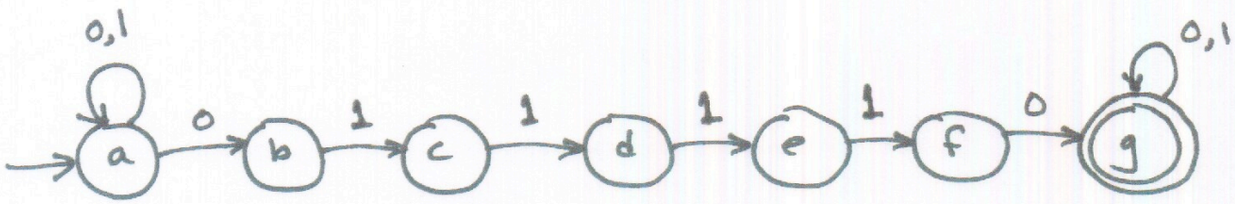
Can take an ε-edge without scanning a symbol.

It is "OPTIONAL"!

Any path to accept is accepted

EXAMPLE

All strings that contain 011110



EXAMPLE STRING: 010001111010101

Lots of bad choices
that don't work,
that don't reach an accept state

All we need is one way
to reach ACCEPT.

If there is any way to run
the machine that ends
with ACCEPT,
Then the ~~[illegible]~~ NFA accepts.

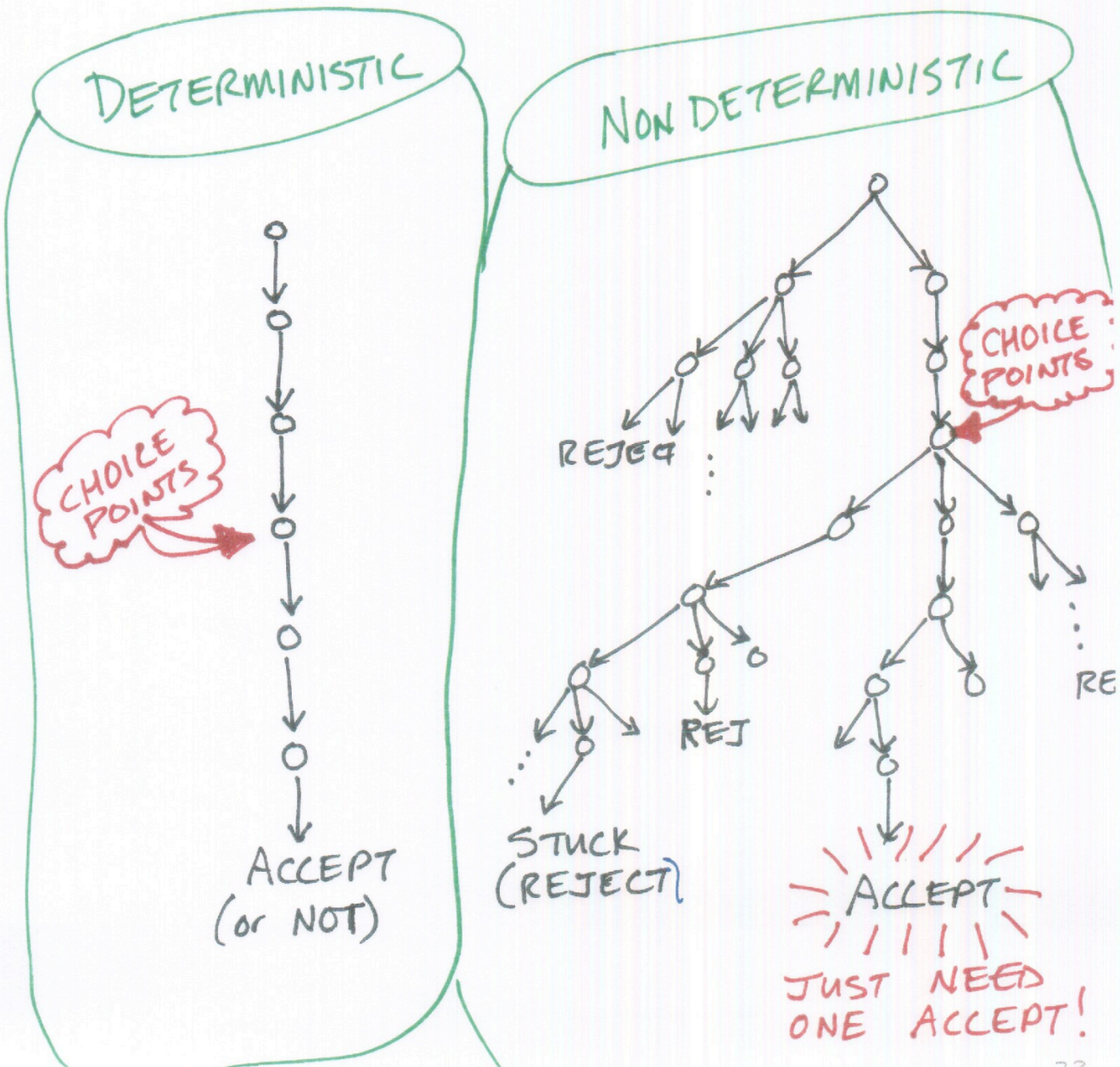
- DFA is much harder (see HW1)
- but does NFA help for the complement?

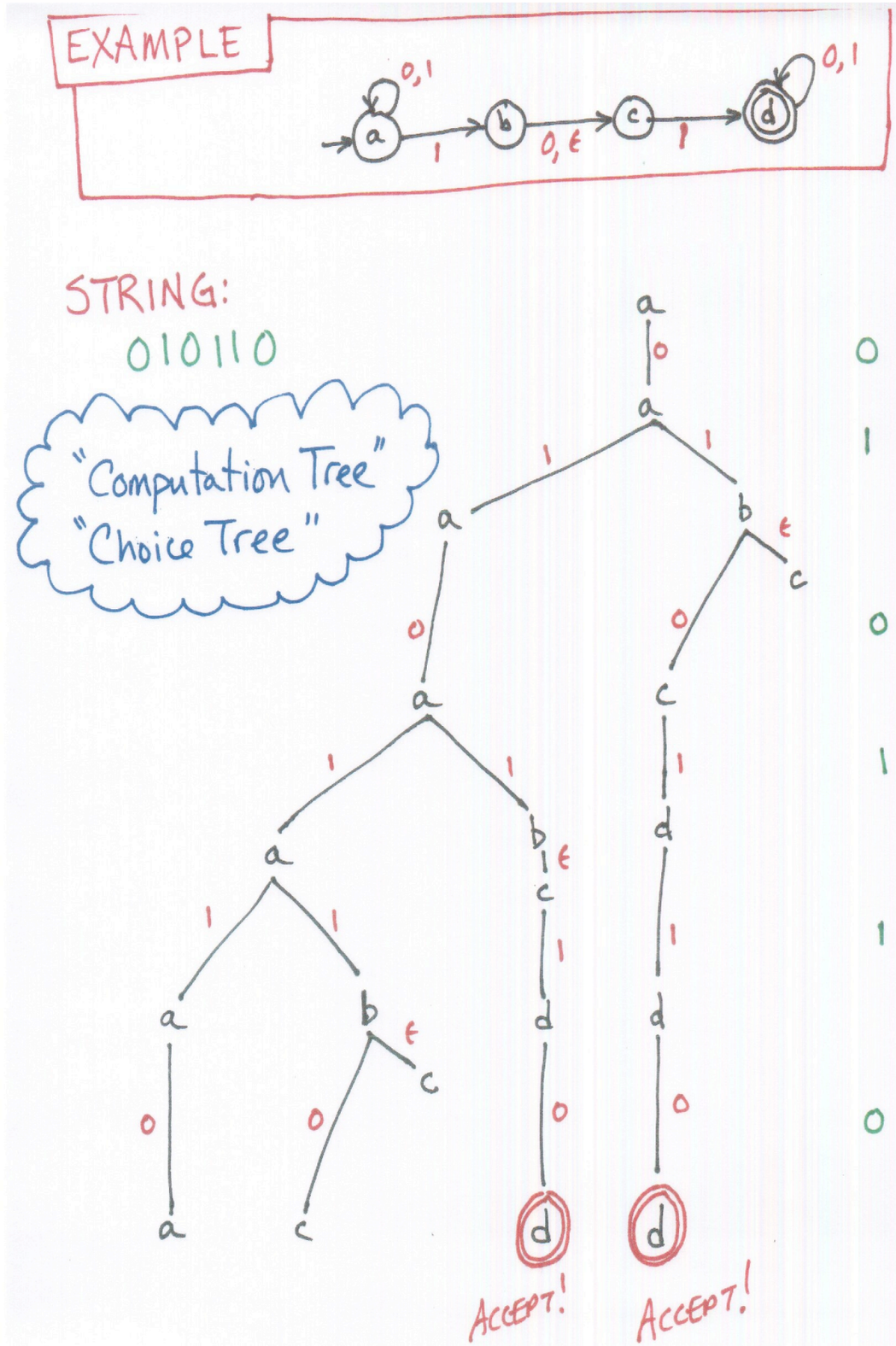
As long as there is (at least) one path...

LOTS OF CHOICES - WHICH ONE TO TRY?

* Try them all.
* Make the right choice at each point.

EQUIVALENT





NFA == DFA, but often a lot easier

THEOREM

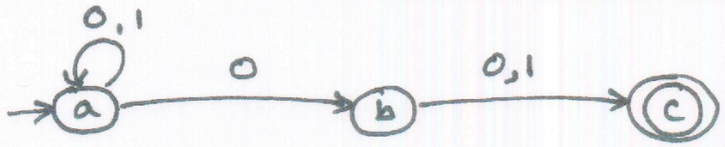
For every Nondeterministic F.S.M. There is an equivalent Deterministic F.S.M.

... But it may be large and hard to find!

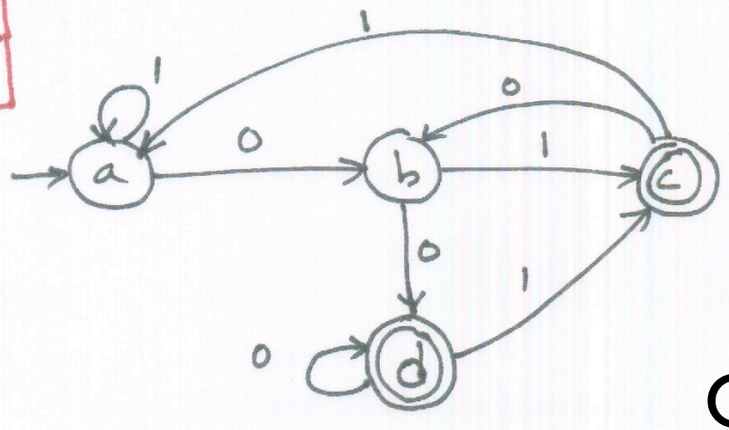
EXAMPLE

All strings over $\{0,1\}^*$ that have a "0" in the second to the last position.

NFA



FSM
DFA

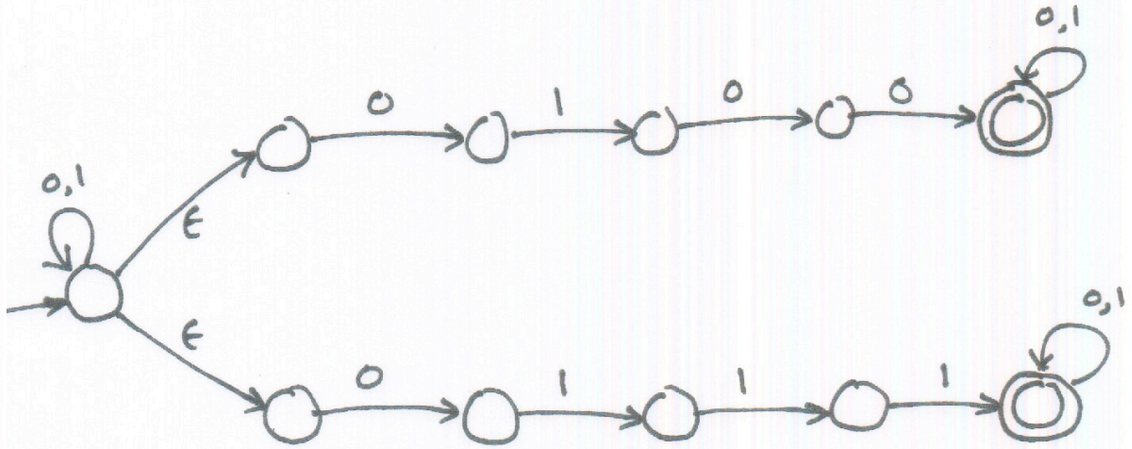


EXAMPLE

String contains either
... 0100...
or ... 0111...

When to start looking?
Which string to look for?

NONDETERMINISM!

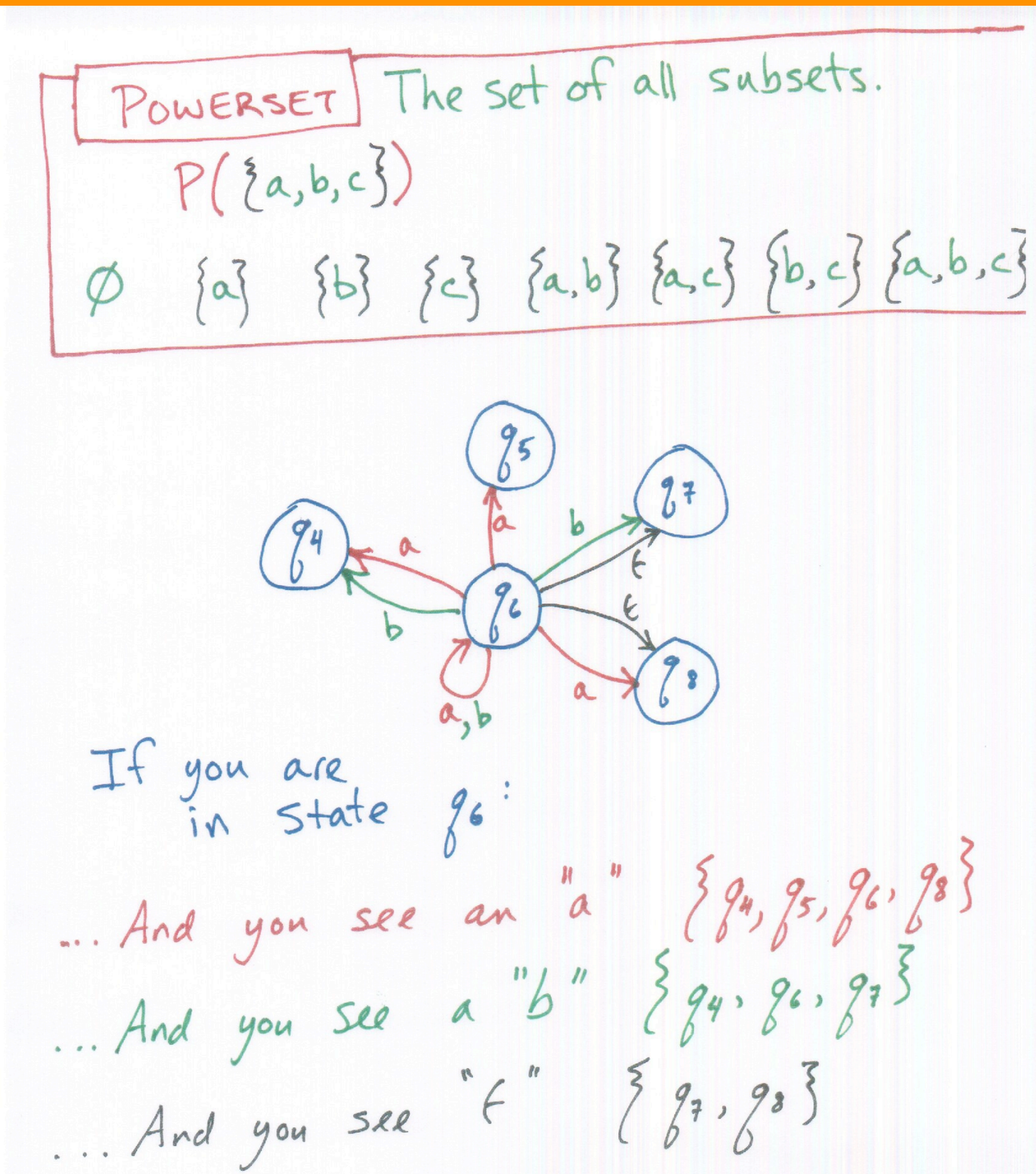


CHALLENGE:

Build/Design a DFA to recognize this language.

Q: how to use NFA for union? for intersection?

Power set (all subsets of a set)



- power set is one of the most important & beautiful mathematical concepts
- P(Q) is often written as 2^Q since $|2^Q| = 2^{|Q|}$

$$2^Q \triangleq \{A \mid A \subseteq Q\}$$

Powerset
NATURAL LANGUAGE SEARCH

Formal Definition of NFA

- two differences on δ
  - $Q \times (\Sigma \cup \{\varepsilon\})$
  - $P(Q)$
- is δ total or partial function?
  - no difference, since $\varnothing \in P(Q)$
  - no need to define trap state

FORMAL DEFINITION OF NONDETERMINISTIC FINITE STATE MACHINE

$M = (Q, \Sigma, \delta, q_0, F)$

Σ = Alphabet

$\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$

Q = States

Σ = Alphabet

q_0 = Start State, $q_0 \in Q$ (member of, not Epsilon.)

F = Accept States, $F \subseteq Q$

δ = Transition Function

$$\delta : Q \times \Sigma_\epsilon \to P(Q)$$

$$\delta : Q \times \Sigma_\epsilon \to P(Q)$$

BEFORE:

States	a	b	c
q_0	q_1	q_2	q_4
q_1	q_2	q_0	q_1
⋮	⋮	⋮	⋮

FOR NFA:

States	a	b	c	ϵ
q_0	$\{q_1, q_2\}$	$\varnothing$	$\{q_4, q_2\}$	$\{q_0\}$
q_1	$\{q_4, q_2, q_1\}$	$\{q_5, q_0\}$	$\varnothing$	$\{q_4, q_3\}$
⋮	⋮	⋮	⋮	⋮

Q: *is every DFA also an NFA??*

A: technically, no!

$\delta_{NFA}(p, a) = \{\delta_{DFA}(p, a)\}$

Computation of NFA w/o epsilon

- string w is accepted iff. $\delta^*(q_0, w) \cap F \neq \varnothing$
- without epsilon transitions, δ^* is easy to define
- with epsilon transitions, you need to define epsilon-closure (very hard; will come back later)

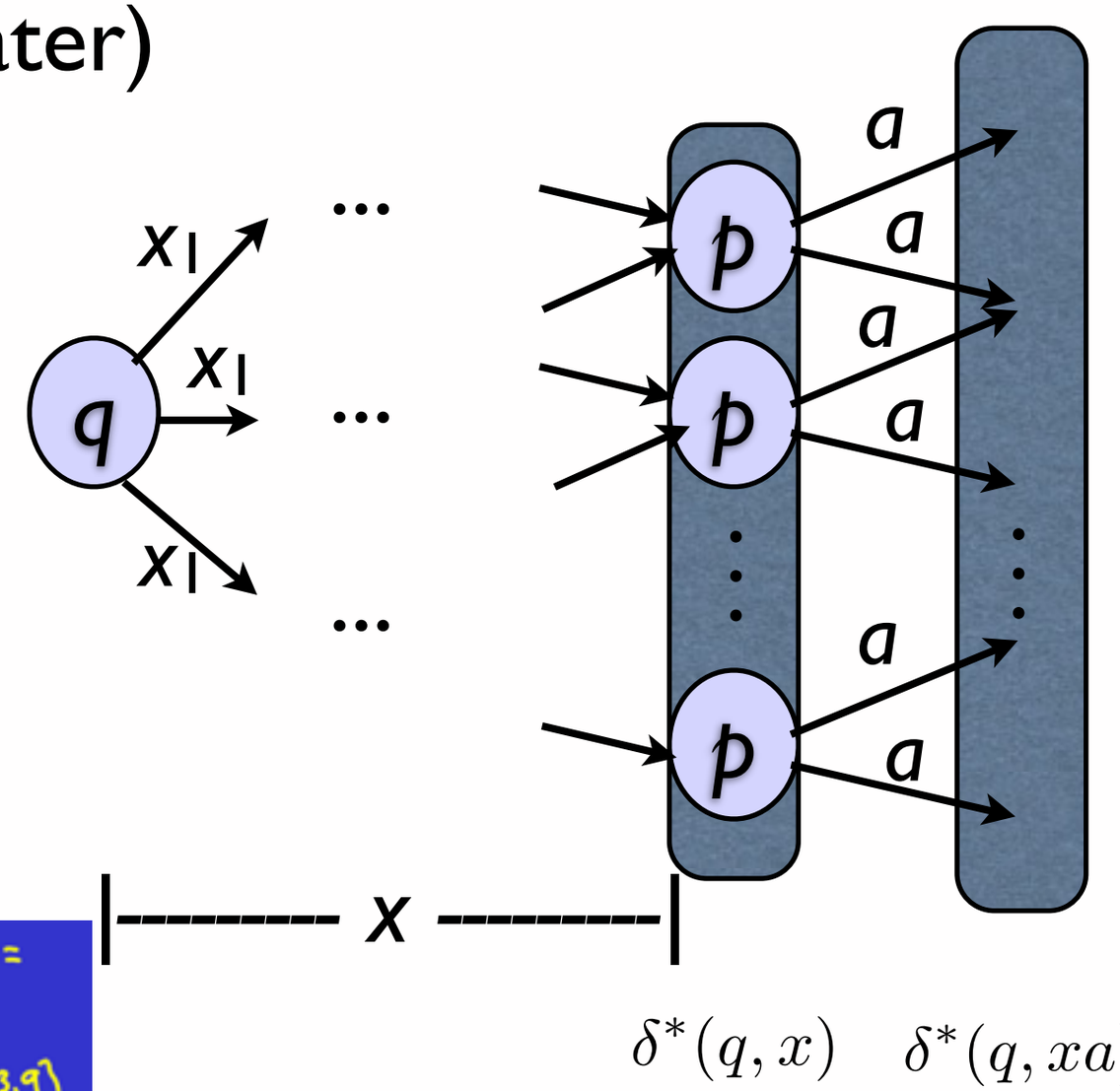
$$\delta^*(q, \epsilon) = \{q\}$$

$$\delta^*(q, xa) = \bigcup_{p \in \delta^*(q,x)} \delta(p, a)$$

alternatively

$$\delta^*(q, ax) = \bigcup_{p \in \delta(q,a)} \delta^*(p, x)$$

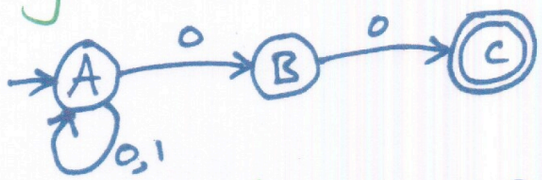
$$\bigcup_{t \in \{1,2,3\}} \{t, t^2\} = \{1,1\} \cup \{2,4\} \cup \{3,9\}$$



NFA => DFA w/o epsilon

EXAMPLE

Accept all strings over $\{0,1\}^*$ ending with "00."

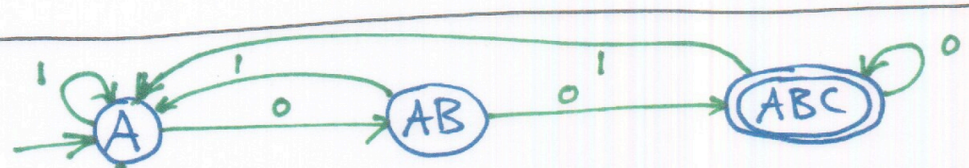


String to check: 00100

Simulate the execution. Put a finger on any state we could be in.

∅ A B C AB BC AC ABC

Let N = Number of states in NFA. What is the (worst case) number of states in the equivalent DFA?

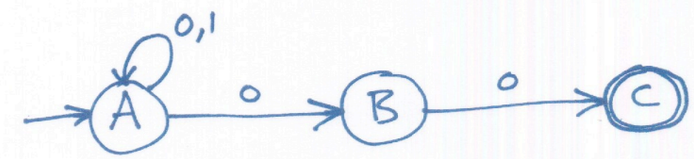


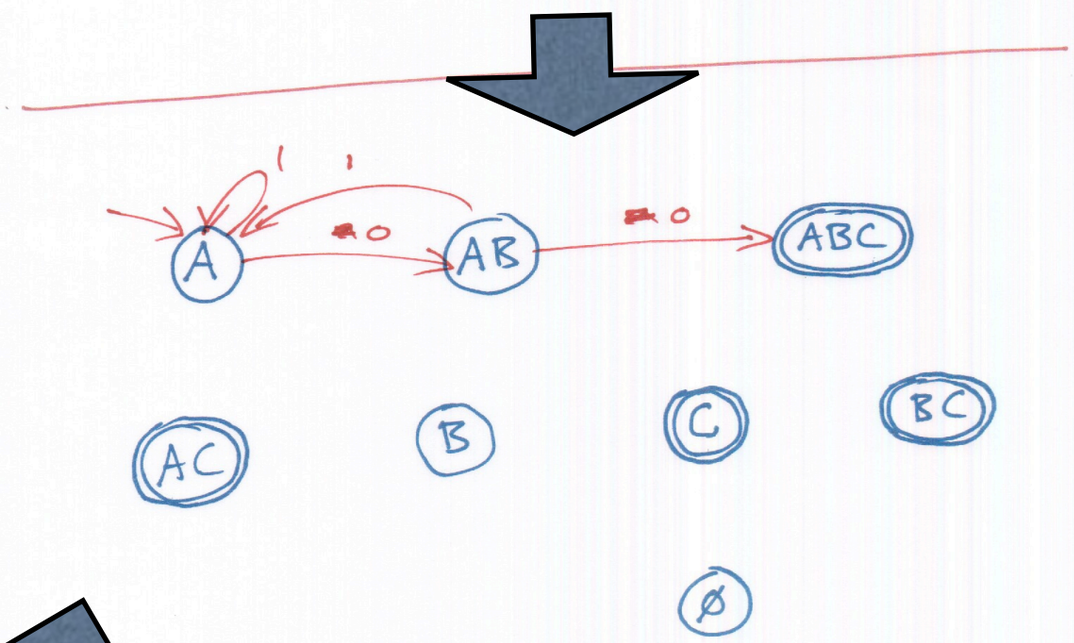
Use Next Slide

unused states

- how to convert NFA to DFA?

NFA





NFA => DFA w/o epsilon

THEOREM

EVERY NONDETERMINISTIC FSM HAS AN EQUIVALENT DETERMINISTIC FSM.

"EQUIVALENT" = Recognizes the same language.

PROOF BY CONSTRUCTION

Given a NFA, let's show how to build an equivalent DFA.

Let $M = (Q, \Sigma, \delta, q_0, F)$

This is the NFA we're given.

Construct

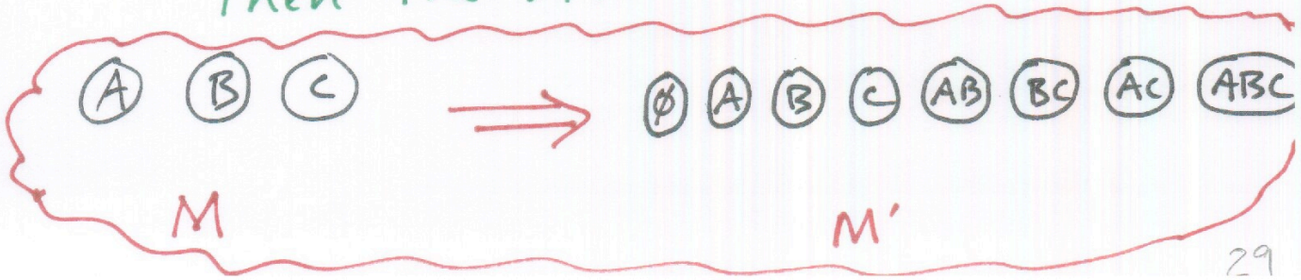
$M' = (Q', \Sigma, \delta', q_0', F')$

This is the DFA we're buiding.

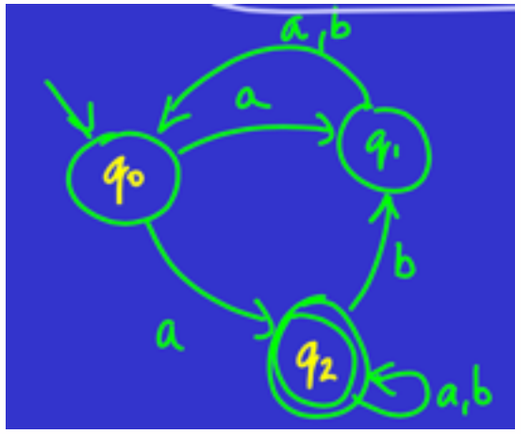
Where...

$Q' = P(Q)$

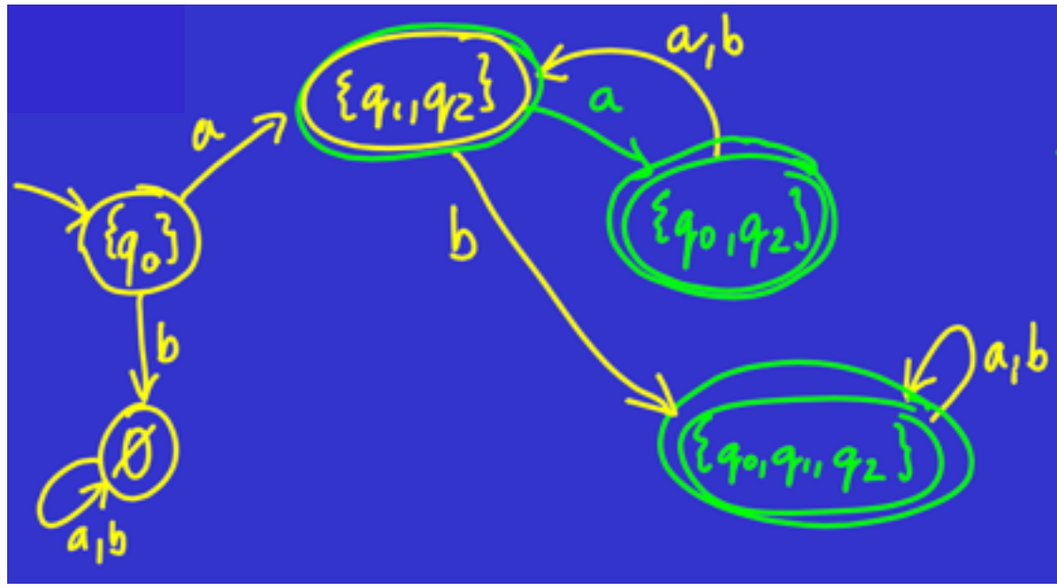
Assume NFA has K states.
Then the DFA will have 2^K states.



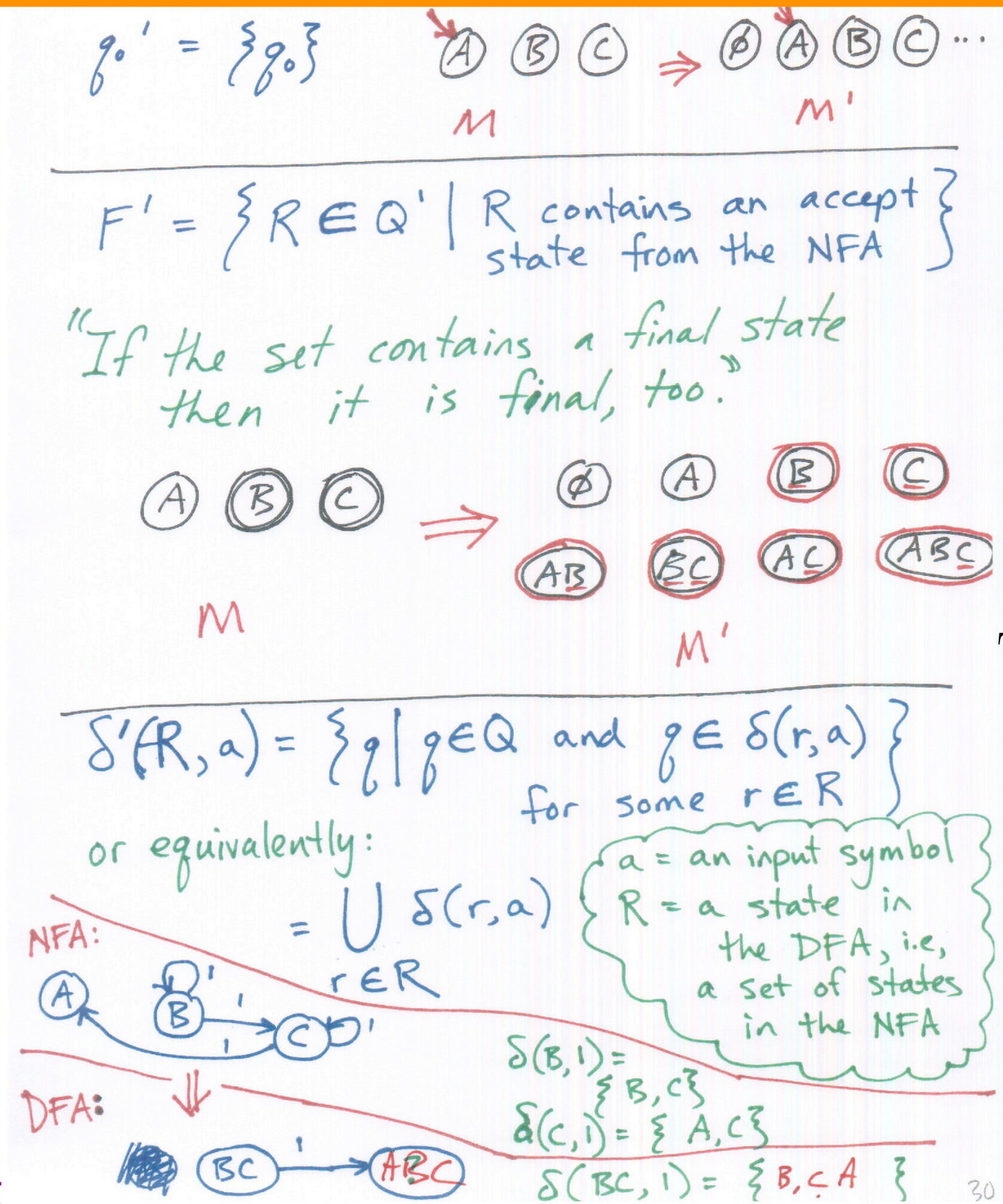
NFA



DFA



Formally: subset construction



NFA $M = (Q, \Sigma, \delta, q_0, F)$

DFA $M' = (Q', \Sigma, \delta', q_0', F')$

where $Q' = P(Q)$

$q_0' = \{q_0\}$

$F' = \{A \in P(Q) \mid A \cap F \neq \emptyset\}$

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$

Thm.: $L(M) = L(M')$

Idea: $\forall w \in L(M), \exists p \in F,$

s.t. $\delta^*(q_0, w) = p.$

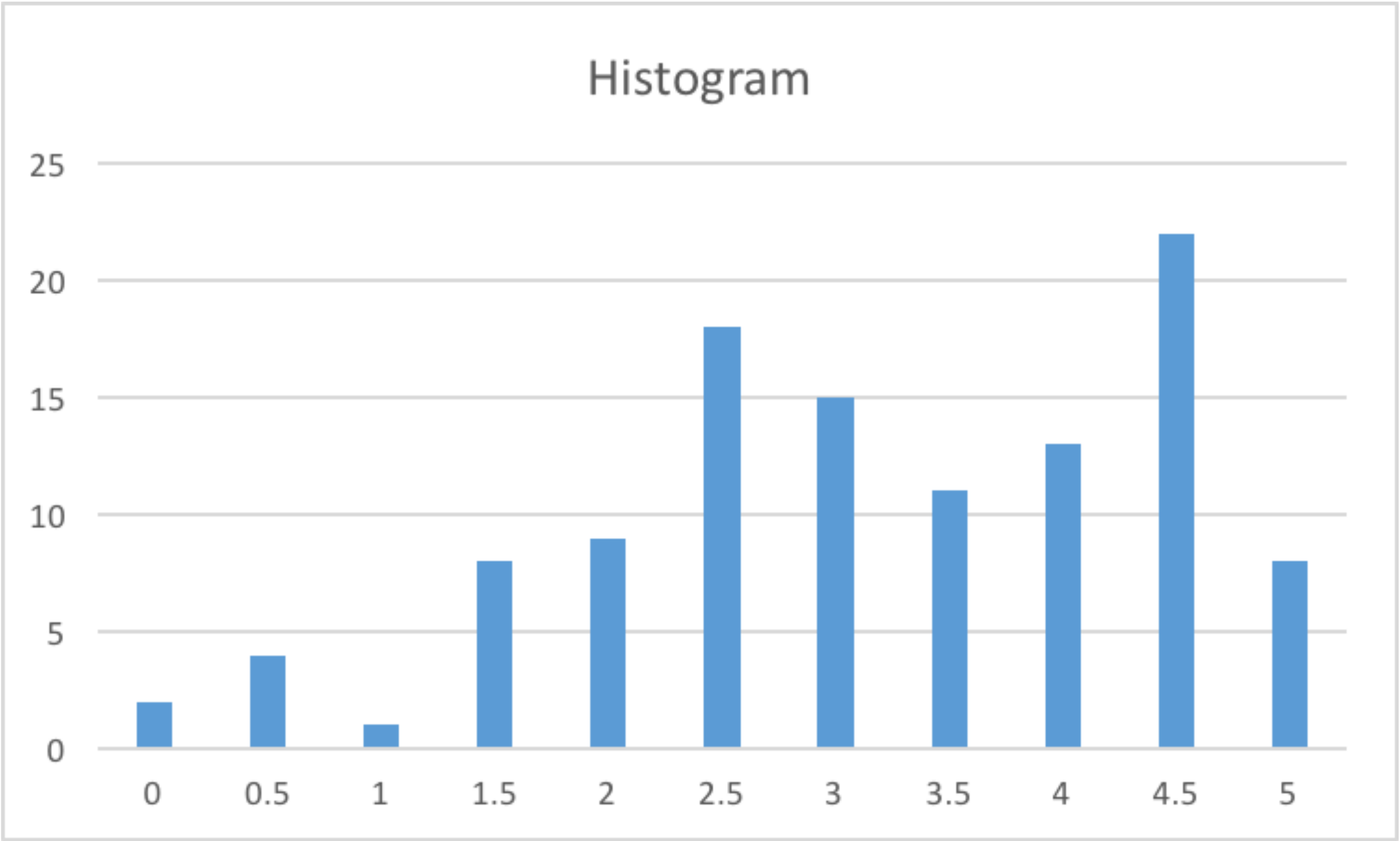
want to show: $p \in \delta'^*(q_0, w)$

also need the other direction

you can still prove that
$\forall q \in Q, w \in \Sigma^*$
$\delta^*(q, w) = \delta'^*(E(\{q\}), w)$

Quiz 3 distribution

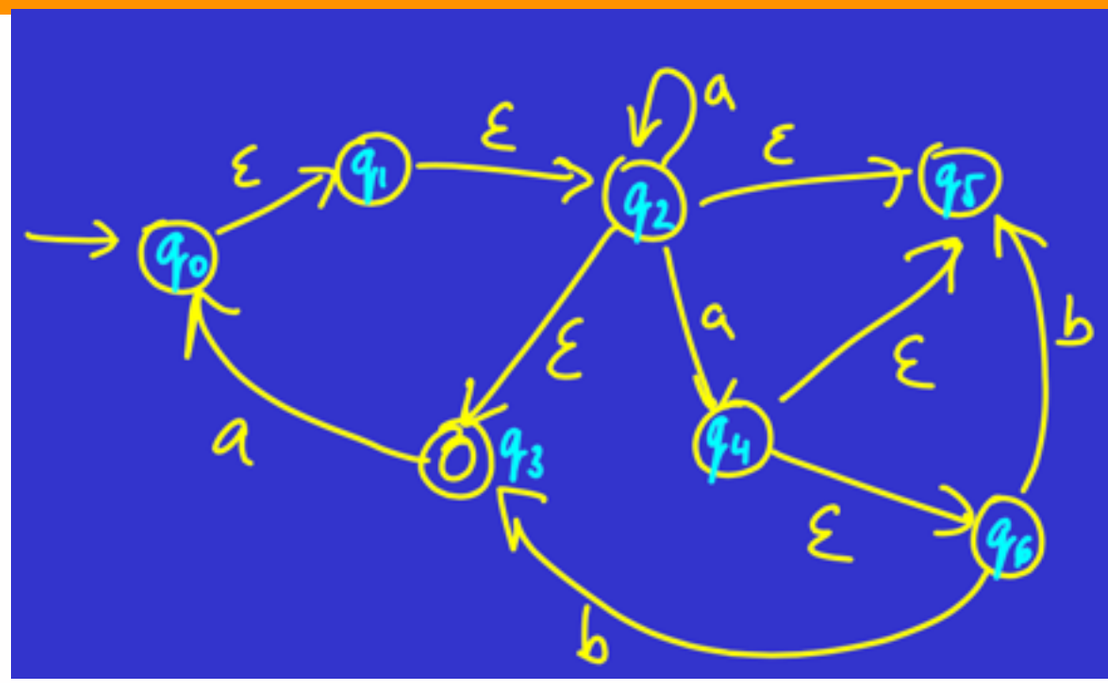
- mean = 3.18, mode = 4.5, stddev = 1.25



F F F C- C C+ B- B B+ A- A

Epsilon-Closure

- def. 0 (informal, Sipser):
  $E(q) = \{p \mid p$ is reachable from q by 0 or more ε edges$\}$
- def. 1 (default): $E(q)$ is the **smallest** set such that:
  - $q \in E(q)$
  - if $p \in E(q)$, then $\delta(p, \varepsilon) \subseteq E(q)$
- def. 2 (inductive): $E(q) = \cup_i E_i(q)$
  - $E_0(q) = \{q\}$
  - $E_{i+1}(q) = \cup_{p \in E_i(q)} \delta(p, \varepsilon)$
- from closure of state to closure of set:
  - $E(R) = \cup_{q \in R} E(q)$

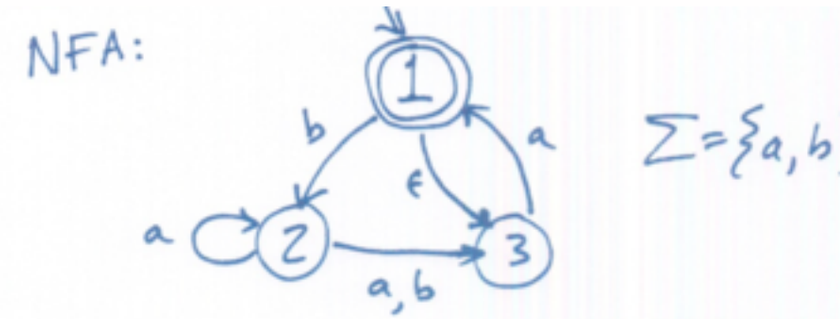


$E(q_0) = \{q_0, q_1, q_2, q_3, q_5\}$
$E(q_2) = \{q_2, q_3, q_5\}$

$E(\{q_2\}) = \{q_2, q_3, q_5\}$
$E(\{q_0\}) = \{q_0, q_1, q_2, q_3, q_5\}$
$E(\{q_1, q_4\}) = \{q_1, q_2, q_3, q_5, q_4, q_6\}$

Computation of NFA w/ epsilon

- string w is accepted iff. $\delta^*(q_0, w) \cap F \neq \varnothing$
- have to refine δ^* using epsilon-closure E(...)



without epsilon

$$\delta : Q \times \Sigma \mapsto P(Q)$$

$$\delta^* : Q \times \Sigma^* \mapsto P(Q)$$

$$\delta^*(q, \epsilon) = \{q\}$$

$$\delta^*(q, xa) = \bigcup_{p \in \delta^*(q,x)} \delta(p, a)$$

alternatively

$$\delta^*(q, ax) = \bigcup_{p \in \delta(q,a)} \delta^*(p, x)$$

with epsilon

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \mapsto P(Q)$$

$$\delta^* : Q \times \Sigma^* \mapsto P(Q)$$

$$\delta^*(q, \epsilon) = E(\{q\})$$

$$\delta^*(q, xa) = E\Big(\bigcup_{p \in \delta^*(q,x)} \delta(p, a) \Big)$$

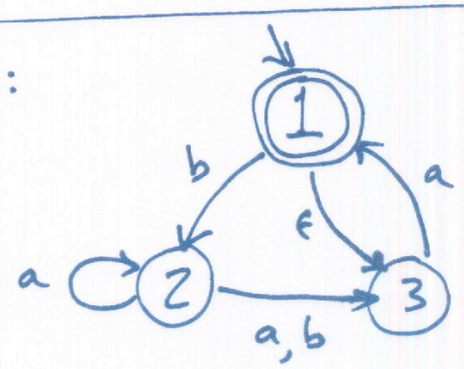
alternatively

$$\delta^*(q, ax) = \bigcup_{p \in E\left(\delta(q,a)\right)} \delta^*(p, x)$$

Example: NFA=>DFA w/ epsilon

EXAMPLE: CONVERT AN NFA TO A DFA.

NFA:



$\Sigma = \{a, b\}$

What states in the DFA?

$Q' = \{ \quad \} = P(Q)$

RED

DFA:

$\emptyset$ 1 2 3 12 13 23 123

Start State?

$q_0' = E(\{1\}) = \{1, 3\}$

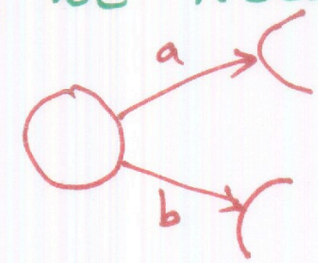
$E(1) = 13$

ACCEPT STATES? ANY THAT CONTAIN 1

$F' =$ 1 12 13 123

TRANSITION FUNCTION?

FOR EACH STATE, WE NEED TWO EDGES:



$\delta'(1, a) = \emptyset$ Since no edges labeled "a" leave 1

$\delta'(1, b) = 2$ From 1 we can get to 2, but no further with ε-edges

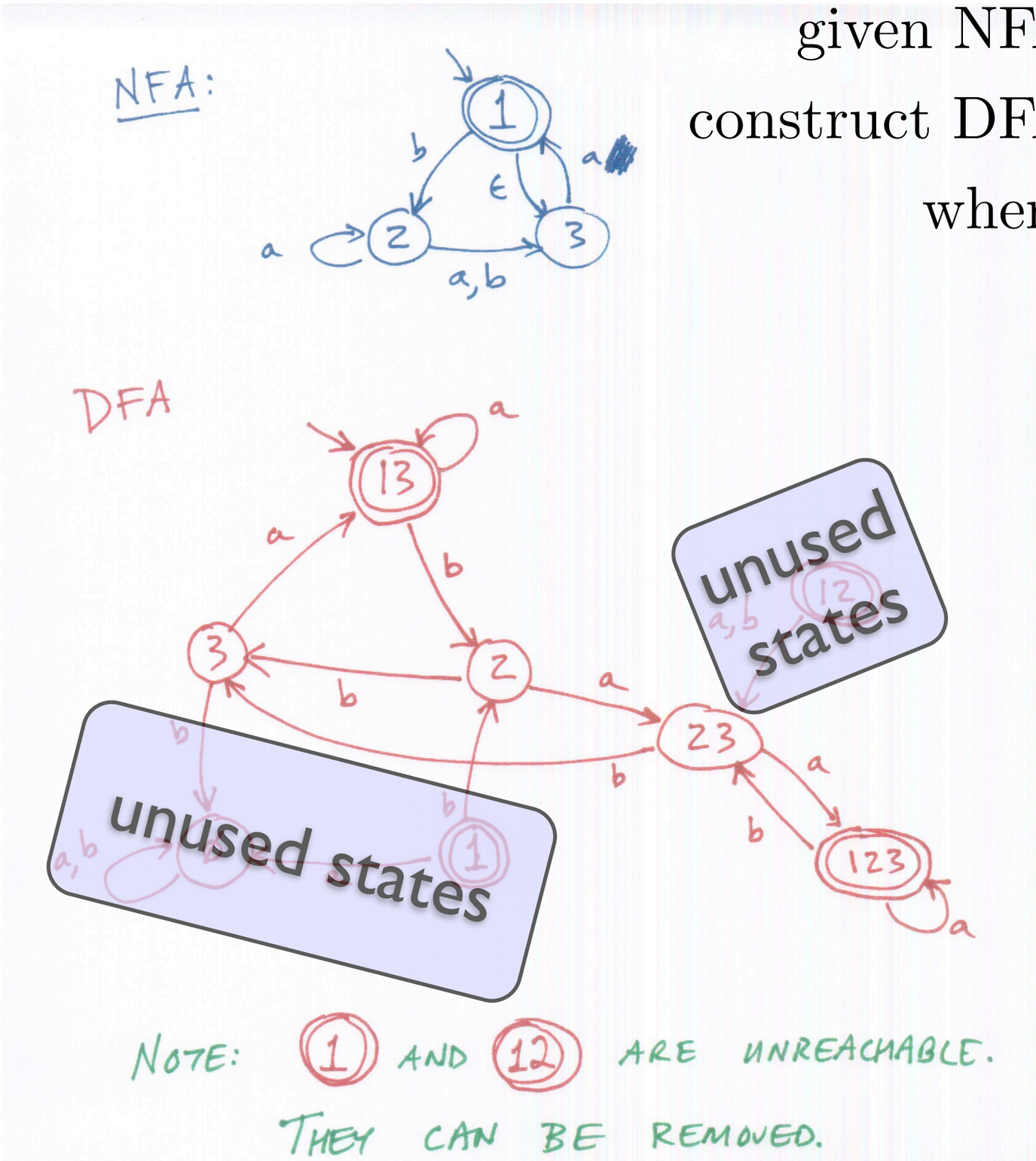
$\delta'(2, a) = 23$

$\delta'(2, b) = 3$

$\delta'(3, a) = 13$ Can get to 1 but can get to 3 by following ε-edge.

$\delta'(3, b) = \emptyset$

Example: NFA=>DFA w/ epsilon



given NFA $M = (Q, \Sigma, \delta, q_0, F)$

construct DFA $M' = (Q', \Sigma, \delta', q_0', F')$

where $Q' = P(Q)$

$q_0' = E(\{q_0\})$

$F' = \{A \in P(Q) \mid A \cap F \neq \emptyset\}$

$$\delta'(R, a) = E\Big(\bigcup_{r \in R} \delta(r, a)\Big)$$

you can still prove that

$\forall q \in Q, w \in \Sigma^*$

$\delta^*(q, w) = \delta'^*(E(\{q\}), w)$

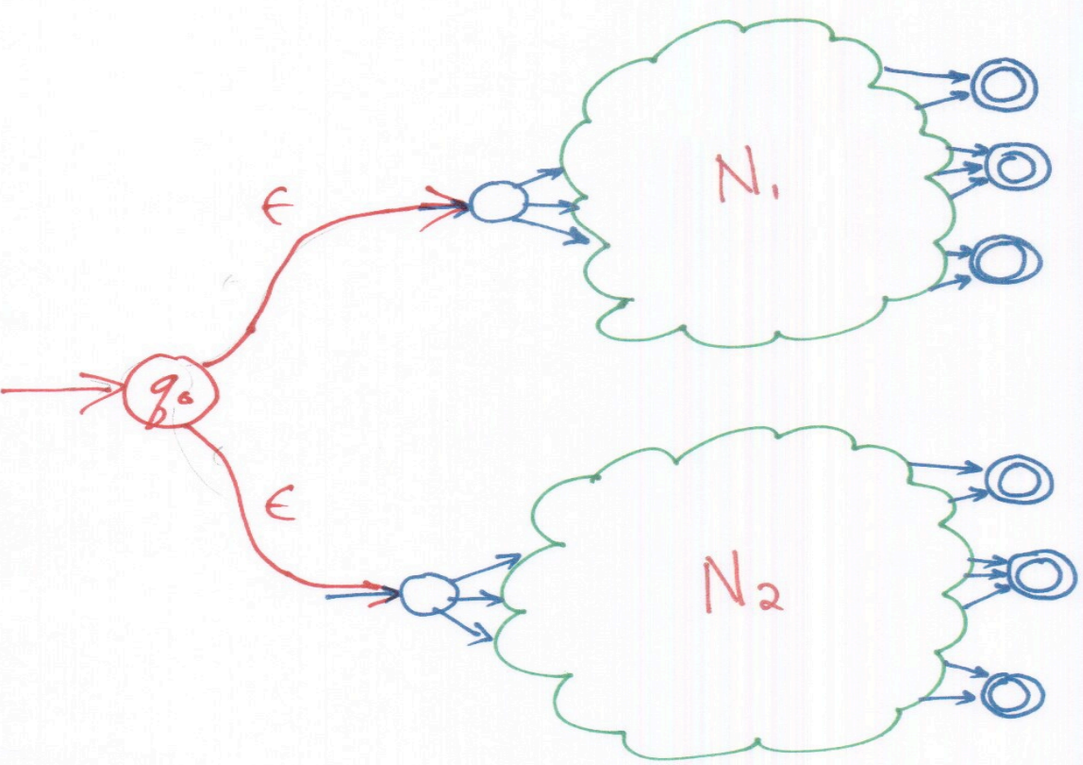
Closed under Union (with NFA)

THEOREM

THE CLASS OF REGULAR LANGUAGES IS "CLOSED" UNDER UNION.

"CLOSURE" OF A LANGUAGE.

PROOF



FORMALLY:

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$

$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$

CONSTRUCT

$N = (Q, \Sigma, \delta, q_0, F)$

$Q = Q_1 \cup Q_2 \cup \{q_0\}$

q_0 is new start state

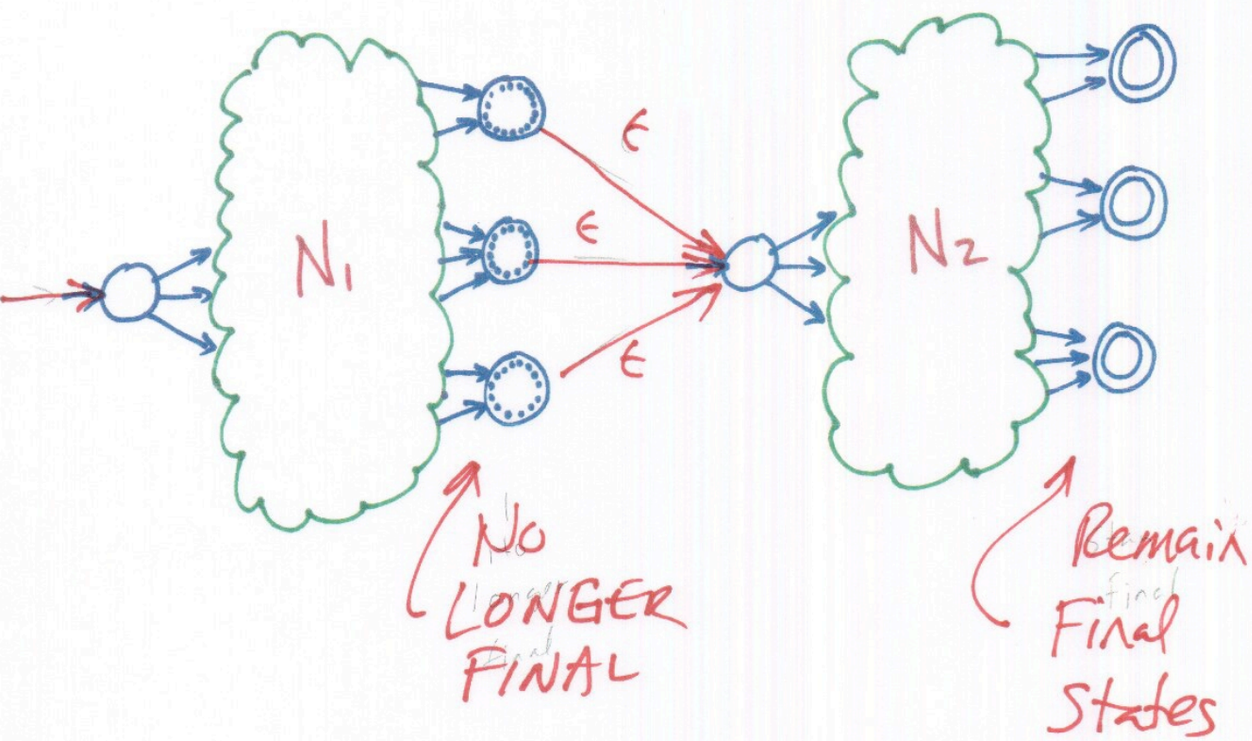
$F = F_1 \cup F_2$

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{if } q \in Q_1 \\ \delta_2(q,a) & \text{if } q \in Q_2 \\ \{q_1, q_2\} & \text{if } q = q_0 \text{ and } a = \epsilon \\ \{\} & \text{if } q = q_0 \text{ and } a \neq \epsilon \end{cases}$$

Closed under Concatenation

THEOREM

THE CLASS OF REGULAR LANGUAGES IS CLOSED UNDER CONCATENATION.



FORMALLY

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$

$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$

Construct

$N = (Q, \Sigma, \delta, q_0, F)$

$Q = Q_1 \cup Q_2$

$q_0 = q_1$ the start state of N_1

$F = F_2$ the final states of N_2

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{if } q \in Q_1 \\ \delta_2(q,a) & \text{if } q \in Q_2 \\ \delta_1(q,a) \cup \{q_2\} & \text{if } q \in F_1 \text{ and } a = \epsilon \\ \delta_1(q,a) & \text{if } q \in F_1 \text{ and } a \neq \epsilon \end{cases}$$

Closed under Star

THEOREM

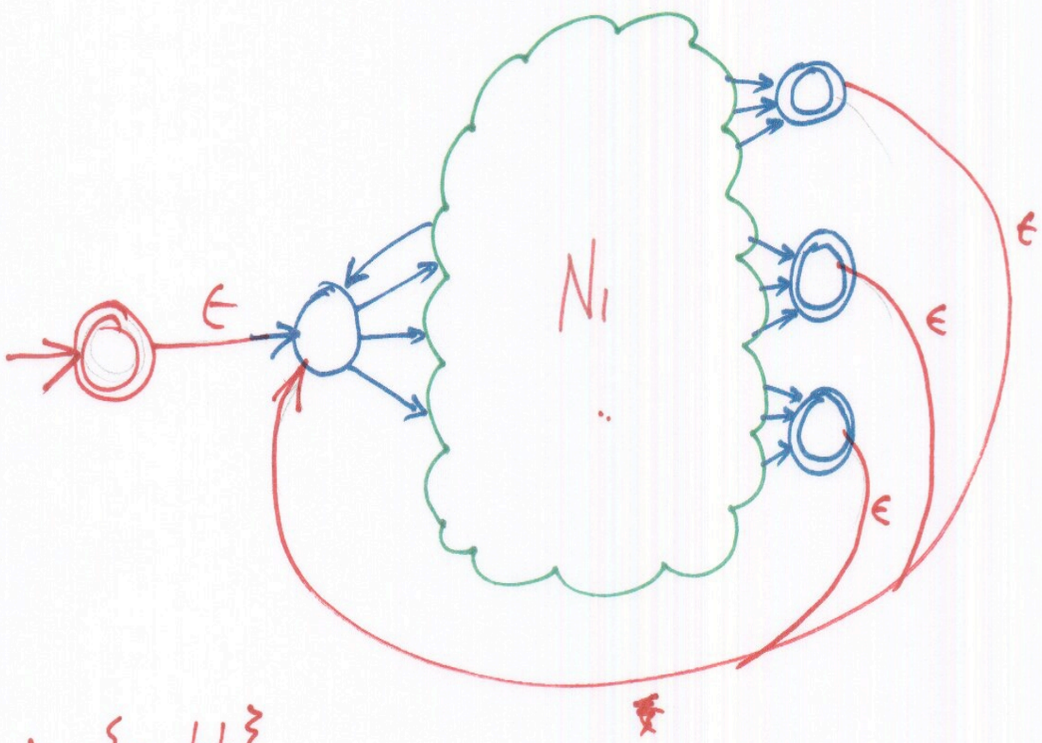
The class of regular languages is closed under "star".

CLOSURE

A^*

PROOF

Same idea.



$A = \{a, bb\}$

$A^* = \{\varepsilon, a, aa, aaa, bb, bbbb, abb, bba, bbaabbabb, \ldots\}$

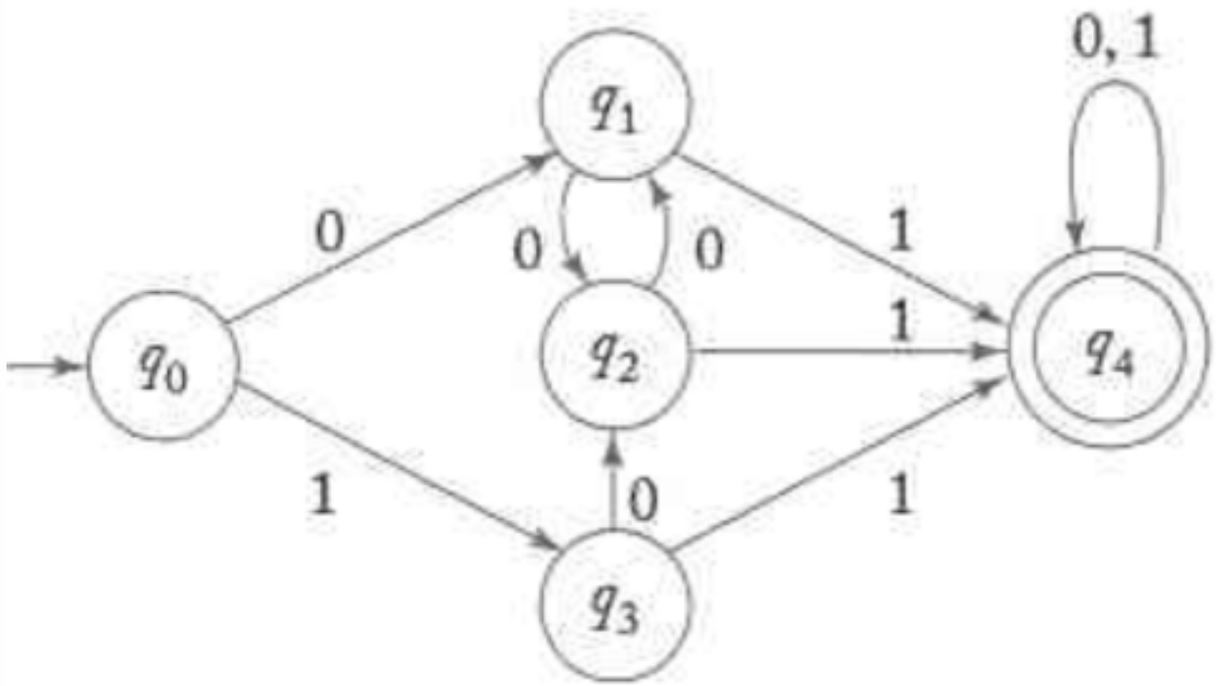
$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1 \text{ and } a = \varepsilon \\ \{q_1\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon. \end{cases}$$

DFA Minimization & Equivalence

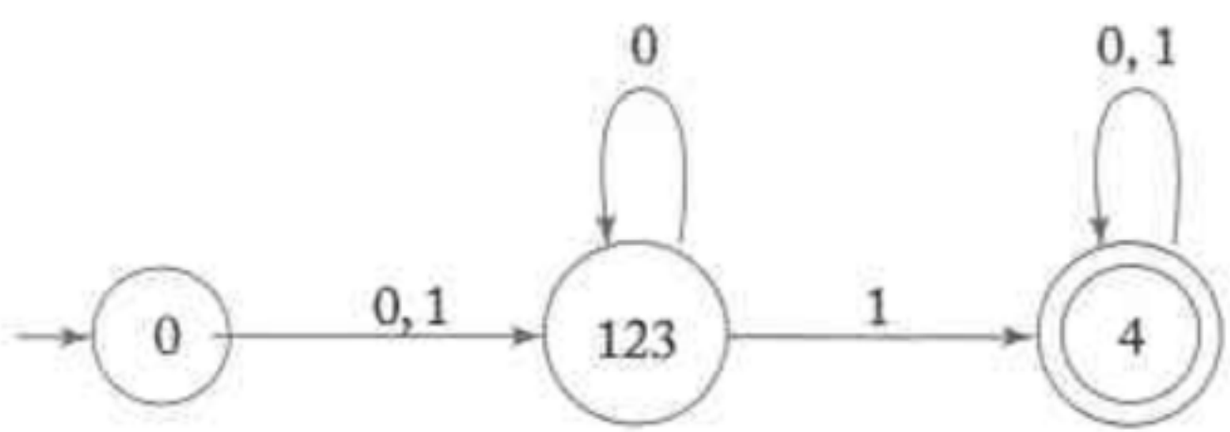
Linz 2.4

DFA minimization (Linz 2.4)

- NFA => DFA=> minimal DFA (b/c some nodes are equivalent or "mergeable")
- first compute equivalence classes of nodes
  - initially, two classes: F and Q - F, and keep splitting:
  - if $\delta(p, a)$ and $\delta(q, a)$ fall into different classes, then p and q aren't equivalent
- merge states in the same class, and form the minimal DFA



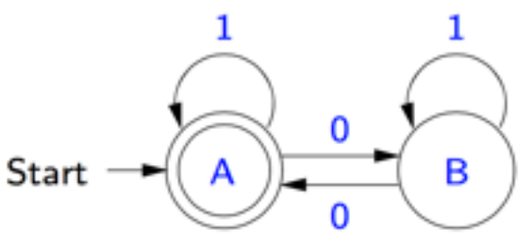
initial: F={q4}, A={q0,q1,q2,q3}
step 1: δ(q0, 1) in A
but δ(q1, 1)= δ(q2, 1)= δ(q3, 1) in F
so split: F= {q4}, A={q0}, B={q1,q2,q3}
no further splitting possible

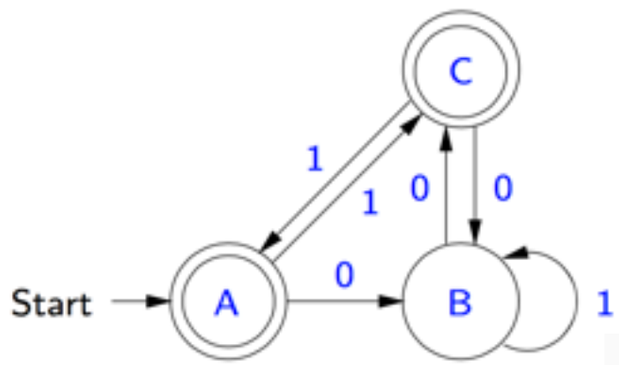


this partition is unique.

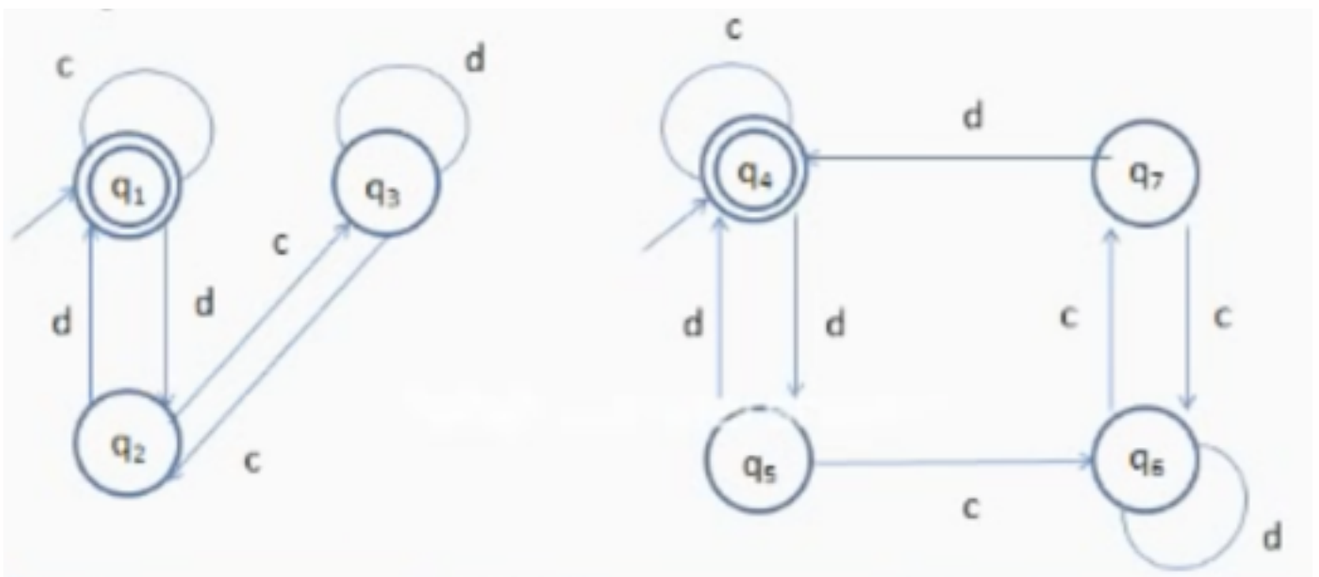
DFA-DFA Equivalence Revisited

- traverse all state-pairs and make sure each pair agrees on "finality" (both accept or both reject)





pair	final?	on 0	on 1
(A, A)	(y, y)	(B, B)	(A, C)
(B, B)	(n, n)	(A, C)	(B, B)
(A, C)	(y, y)	(B, B)	(A, A)
no new pairs found			



Q Q'	Q_c Q_c'	Q_d Q_d'
q_1 , q_4	q_1 , q_4	q_2 , q_5
q_2 , q_5	q_3 , q_6	q_1 , q_4
q_3 , q_6	q_2 , q_7	q_3 , q_6
q_2 , q_7	q_3 , q_6	q_1 , q_4

Regular Expressions

Sipser 1.3, Linz 3.1-3.2

Definition of Regular Expression

DEFINITION 1.52

Say that R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

also notated: $R_1 \mid R_2$ or $R_1 + R_2$

also $R_1 R_2$

let R^+ be shorthand for RR^*. $R^+ \cup \varepsilon = R^*$

$R \cup \emptyset = R$.
Adding the empty language to any other language will not change it.

$R \circ \varepsilon = R$.
Joining the empty string to any string will not change it.

$R \cup \varepsilon$ may not equal R.
For example, if $R = 0$, then $L(R) = \{0\}$ but $L(R \cup \varepsilon) = \{0, \varepsilon\}$.

$R \circ \emptyset$ may not equal R.
For example, if $R = 0$, then $L(R) = \{0\}$ but $L(R \circ \emptyset) = \emptyset$.

EXAMPLE 1.53

In the following instances, we assume that the alphabet Σ is $\{0,1\}$.

1. $0^*10^* = \{w \mid w \text{ contains a single } 1\}$.
2. $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.
3. $\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$.
4. $1^*(01^+)^* = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$.
5. $(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$.[5]
6. $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of } 3\}$.
7. $01 \cup 10 = \{01, 10\}$.
8. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w \mid w \text{ starts and ends with the same symbol}\}$.
9. $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$.
   The expression $0 \cup \varepsilon$ describes the language $\{0, \varepsilon\}$, so the concatenation operation adds either 0 or ε before every string in 1^*.
10. $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$.
11. $1^*\emptyset = \emptyset$.
    Concatenating the empty set to any set yields the empty set.
12. $\emptyset^* = \{\varepsilon\}$.
    The star operation puts together any number of strings from the language to get a string in the result. If the language is empty, the star operation can put together 0 strings, giving only the empty string.

$$(+ \cup - \cup \varepsilon)\,(D^+ \cup D^+ . D^* \cup D^* . D^+)$$

where $D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ is the alphabet of decimal digits. Examples of generated strings are: 72, 3.14159, +7., and -.01.

Examples

Parsing Practice

aa bUc aa bUc aa = ?

(aab) U (caab) U (caa)

---

aab | caab | caa

= (aab) | (caab) | (caa)

---

d U ab* cd* = ?

d | ab* cd* = ?

= (d) U (a(b*)c(d*))

= d | (a(b*)c(d*))

EXAMPLE REGULAR EXPRESSIONS

Assume $\Sigma = \{a, b, c, d\}$

a

$\{a\}$

abccb

$\{abccb\}$

ab U cd = ab/cd

$\{ab, cd\}$

a(bUc)d = a(b/c)d

$\{abd, acd\}$

ab*c

$\{ac, abc, abbc, abbbc, \ldots\}$

a(bUϵ)c = a(b/ϵ)c = a[b]c

$\{abc, ac\}$

$\emptyset$

$\{\}$

a(bUc)$\emptyset$

$\{\}$

$\emptyset^*$

$\{\epsilon\}$

Regular Expression == Regular Language

REGULAR EXPRESSION

Each Regular Expression describes a Language. Which Language?

$L(R) = ?$

$L(a) = \{a\}$

$L(R_1 | R_2) = L(R_1) \cup L(R_2)$

$L(R_1 \circ R_2) = L(R_1) \circ L(R_2)$

$L(R_1^*) = L(R_1)^*$

$L(\epsilon) = \{\epsilon\}$

$L(\emptyset) = \{\}$

$L((R_1)) = L(R_1)$

Regular Languages are closed under Union, Concatenation, and Star.

$\Rightarrow$ Regular Expressions describe Regular Languages.

THEOREM

A LANGUAGE IS REGULAR IFF SOME REGULAR EXPRESSION DESCRIBES IT.

PREVIOUS DEFINITION

A LANGUAGE IS REGULAR IFF IT IS ~~DESCRIBED~~ RECOGNIZED BY SOME FINITE STATE MACHINE. (NFA=DFA).

GIVEN A REGULAR EXPRESSION, E, THE LANGUAGE IT DESCRIBES

$L(E) =$ See previous slide

So we are saying that the class of languages recognized by DFA's, NFAs, and Reg. Expressions is the same! All have equivalent "power"!

Regular Expression == Regular Language

LEMMA 1:

IF A LANGUAGE IS DESCRIBED BY A REGULAR EXPRESSION, THEN IT IS REGULAR.

PROOF #1: USE THE CLOSURE OF $\cup$ $*$ and $\circ$.

PROOF #2: FROM A REGULAR EXPRESSION, BUILD AN NFA TO RECOGNIZE IT.

LEMMA 2:

IF A LANGUAGE IS REGULAR, THEN IT CAN BE DESRIBED BY A REGULAR EXPRESSION.

PROOF APPROACH:

- START WITH A DFA THAT RECOGNIZES IT.
- BUILD A GNFA (GENERALIZED NONDETERMINISTIC FINITE STATE AUTOMATON).
- REDUCE IT (details to follow).
- THIS YIELDS A REGULAR EXPRESSION.

INDUCTIVE PROOF! (STRUCTURAL INDUCTION)

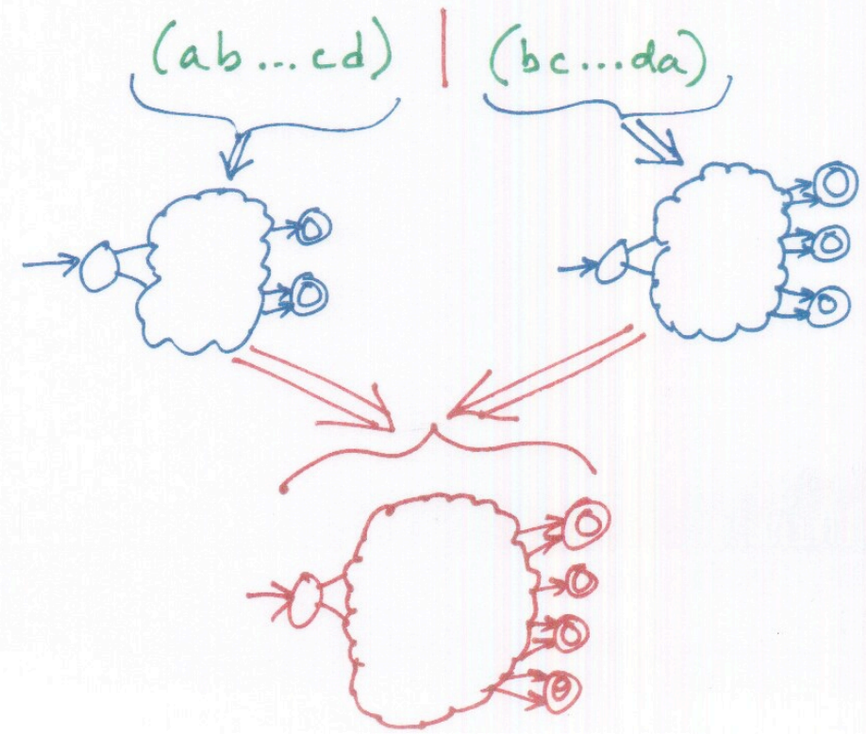
LEMMA 1

- WE HAVE A REGULAR EXPRESSION. CALL IT R.
- CONVERT R TO A NFA.
- CONCLUDE THAT THE LANGUAGE MUST BE REGULAR.

EVERY LARGE REGULAR EXPRESSION IS MADE OF SMALLER REGULAR EXPRESSIONS.

ASSUME WE CAN BUILD THE NFAs FOR SMALLER REGULAR EXPRESSIONS.

SHOW HOW TO BUILD THE NFA FOR LARGER REGULAR EXPRESSIONS.



RE => NFA

DEFINITION OF REGULAR EXPRESSIONS

$R = a$

$R = R_1 | R_2$

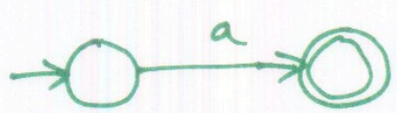
$R = R_1 \circ R_2$

$R = R_1^*$

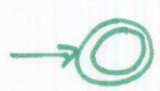
$R = \epsilon$

$R = \emptyset$

$R = a$



$R = \epsilon$



$R = \emptyset$

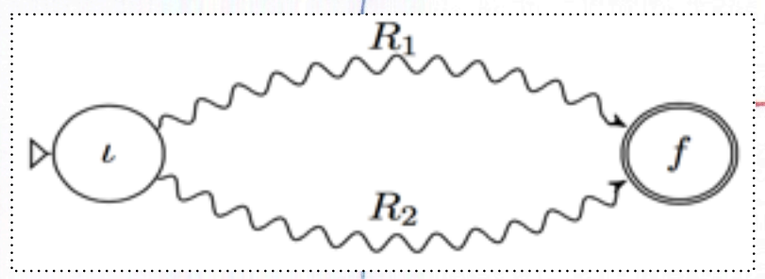
$R = R_1 | R_2$

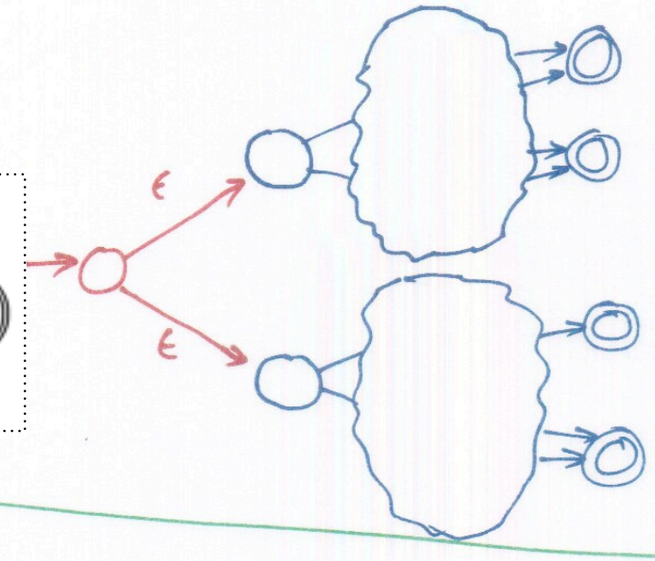
$R = R_1 \circ R_2$

$R = R_1^*$

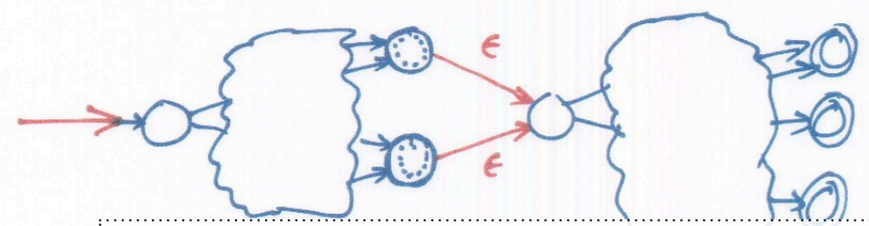
Same construction used in proof of closure.

$R_1 | R_2$

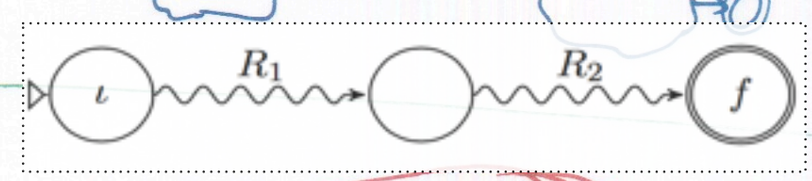


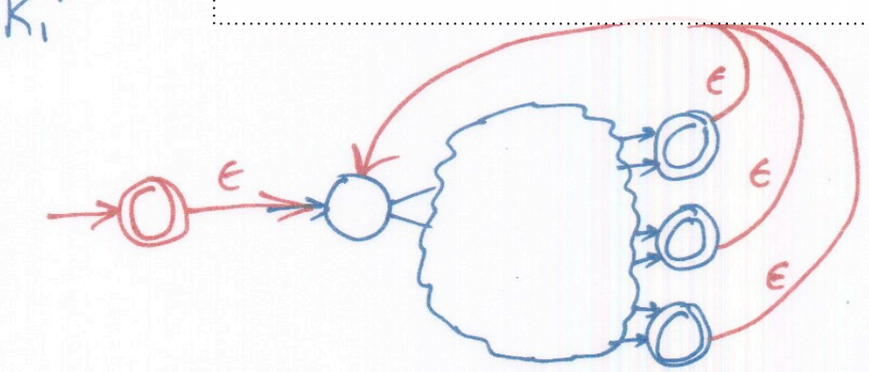


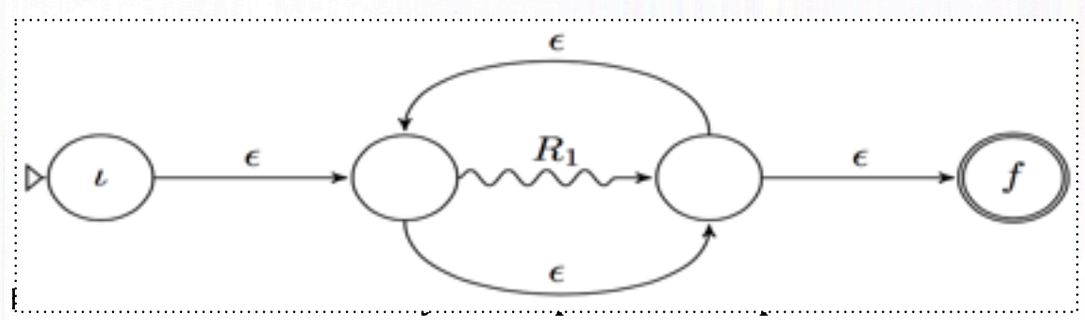
$R_1 \circ R_2$



R_1^*

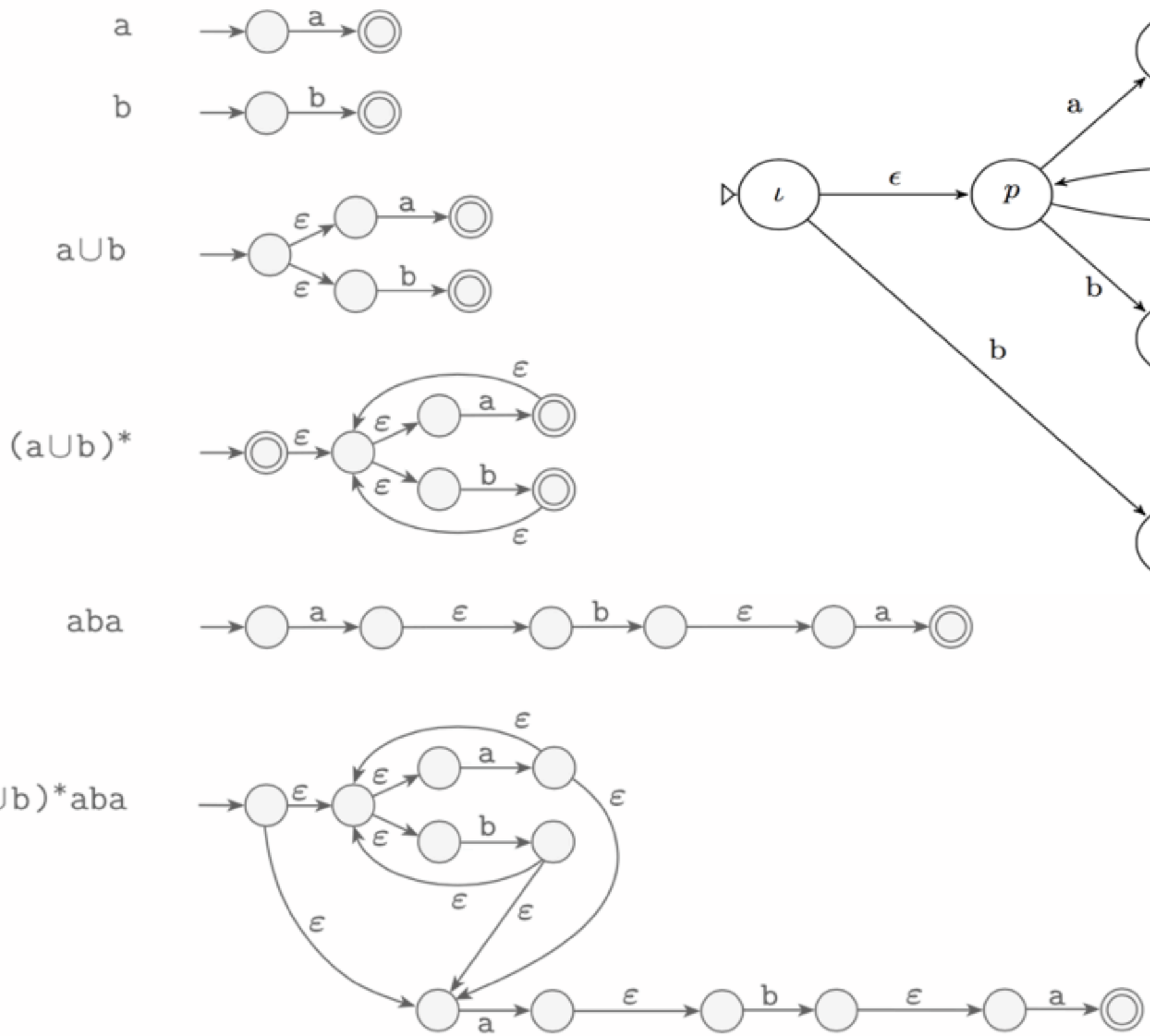


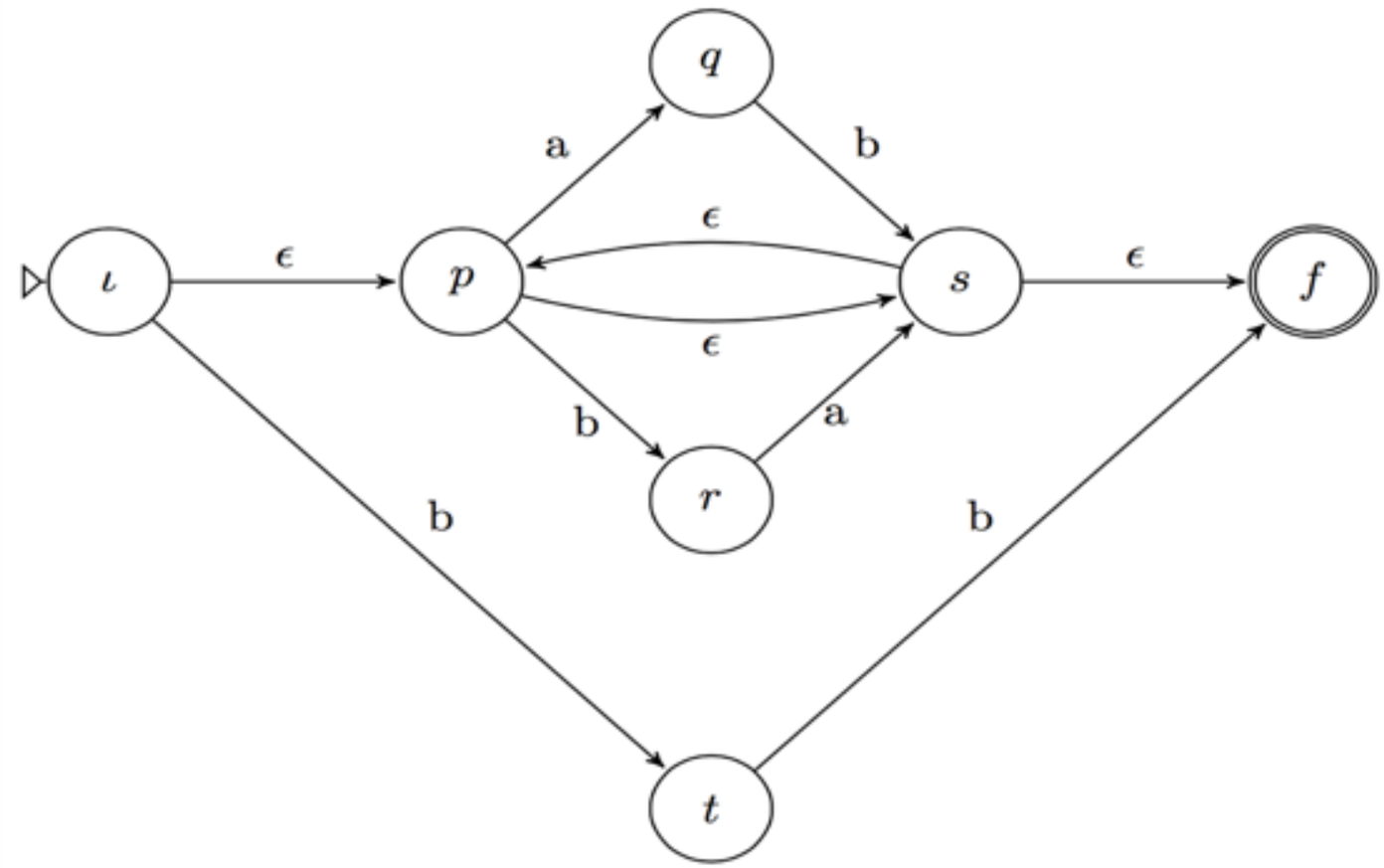




Examples: (a|b)*aba

(ab|ba)*|bb





DFA/NFA => GNFA => RE

LEMMA 2

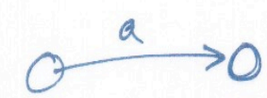
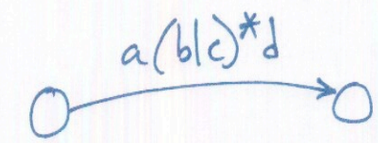
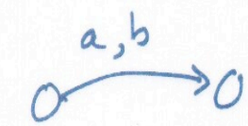
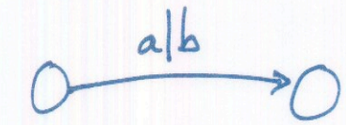
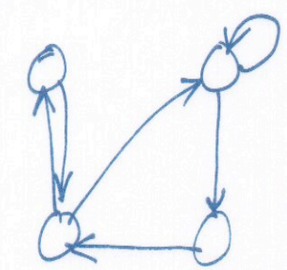
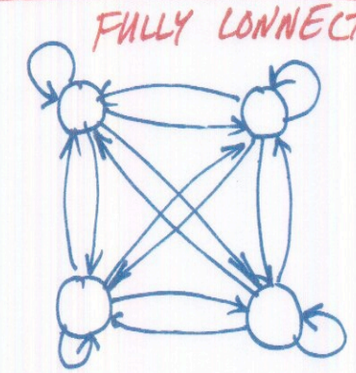
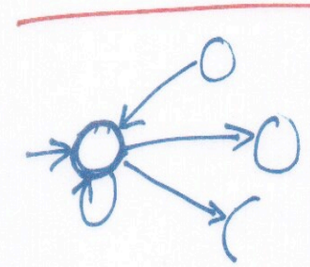
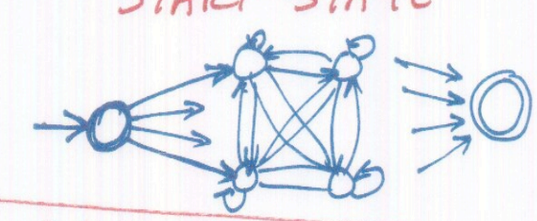
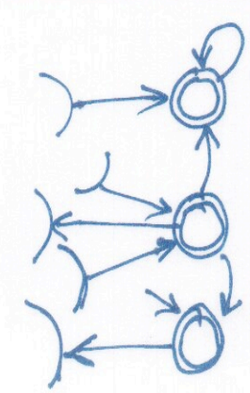
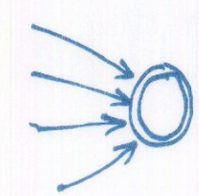
IF A LANGUAGE IS REGULAR, THEN IT CAN BE DESCRIBED BY A REGULAR EXPRESSION

GENERALIZED NONDETERMINISTIC FINITE AUTOMATON (GNFA)

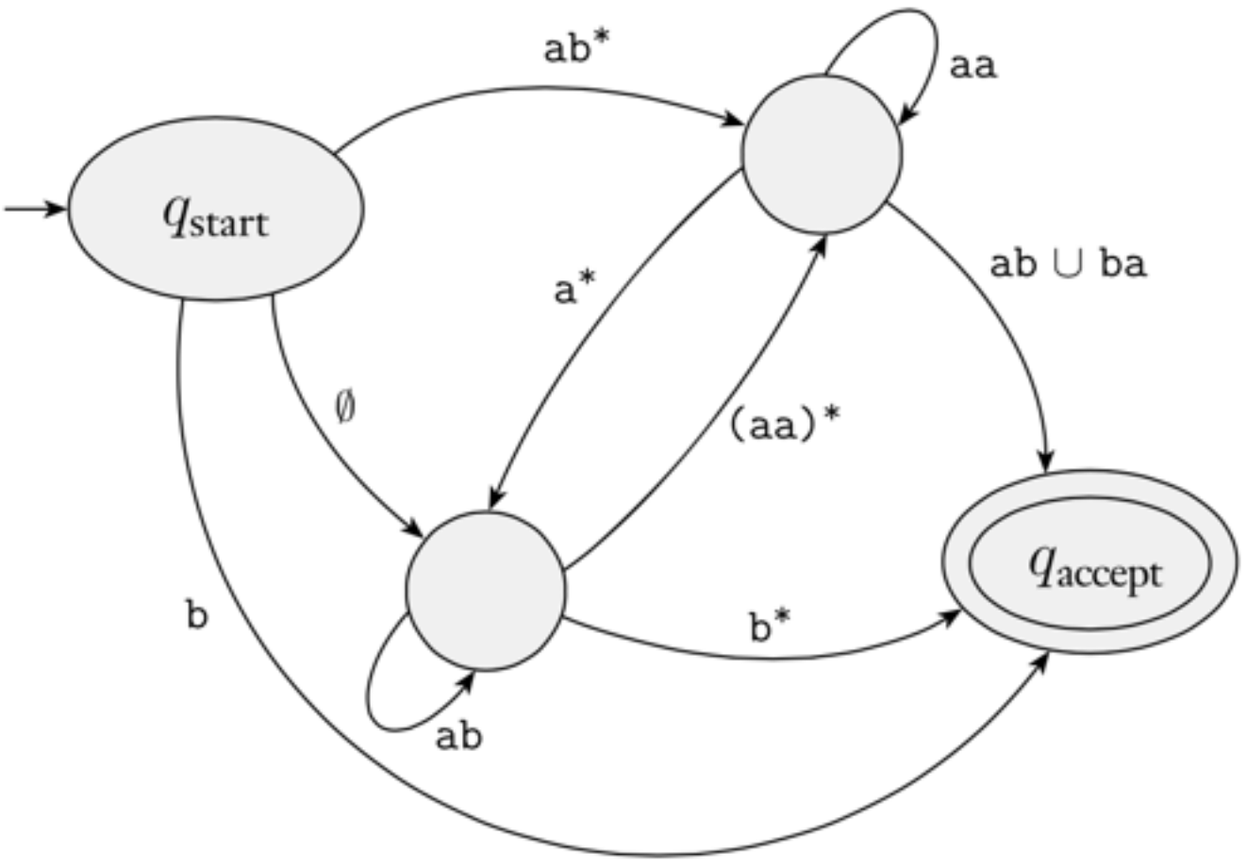
LIKE A N.F.A.

EXCEPT:

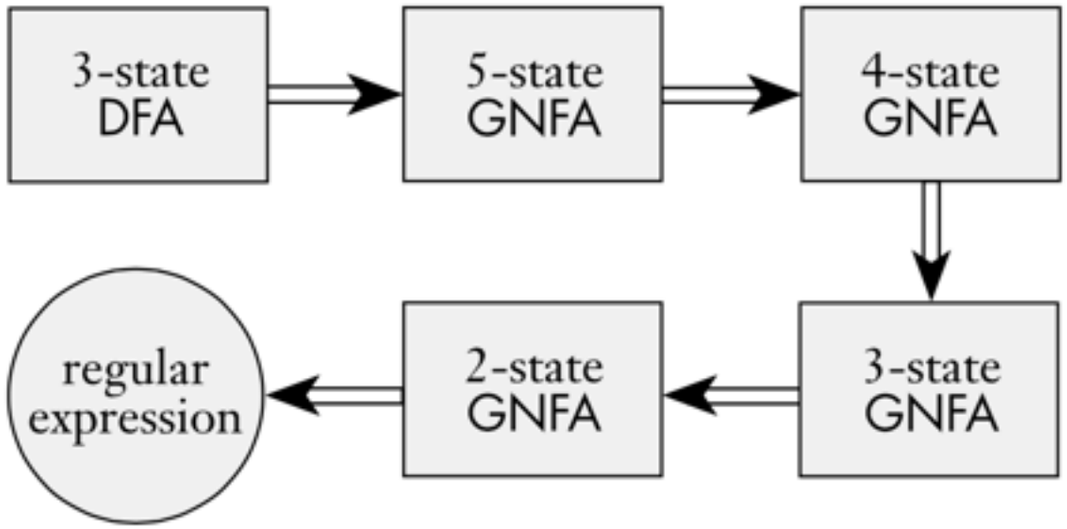
- EDGES ARE LABELED WITH REGULAR EXPRESSIONS!
- ONLY ONE ACCEPT STATE
- THERE IS EXACTLY ONE EDGE FROM EVERY STATE TO EVERY OTHER STATE. (INCLUDING AN EDGE TO SAME STATE)
- EXCEPT: NO EDGES GOING TO THE START STATE.
- EXCEPT: NO EDGES GOING OUT OF THE ACCEPT STATE.

NFA	GNFA
	
	ONLY ONE EDGE BETWEEN STATES 
	FULLY CONNECTED 
	NO EDGES TO START STATE 
	ONLY ONE FINAL STATE; NO EDGES OUT OF IT. 

GNFA example



typical process:



DEFINITION 1.64

A ***generalized nondeterministic finite automaton*** is a 5-tuple, $(Q, \Sigma, \delta, q_{start}, q_{accept})$, where

1. Q is the finite set of states,
2. Σ is the input alphabet,
3. $\delta\colon (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \longrightarrow \mathcal{R}$ is the transition function,
4. q_{start} is the start state, and
5. q_{accept} is the accept state.

GNFA => RE Algorithm

CONVERT(G):

1. Let k be the number of states of G.
2. If $k = 2$, then G must consist of a start state, an accept state, and a single arrow connecting them and labeled with a regular expression R. Return the expression R.
3. If $k > 2$, we select any state $q_{\text{rip}} \in Q$ different from q_{start} and q_{accept} and let G' be the GNFA $(Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$, where

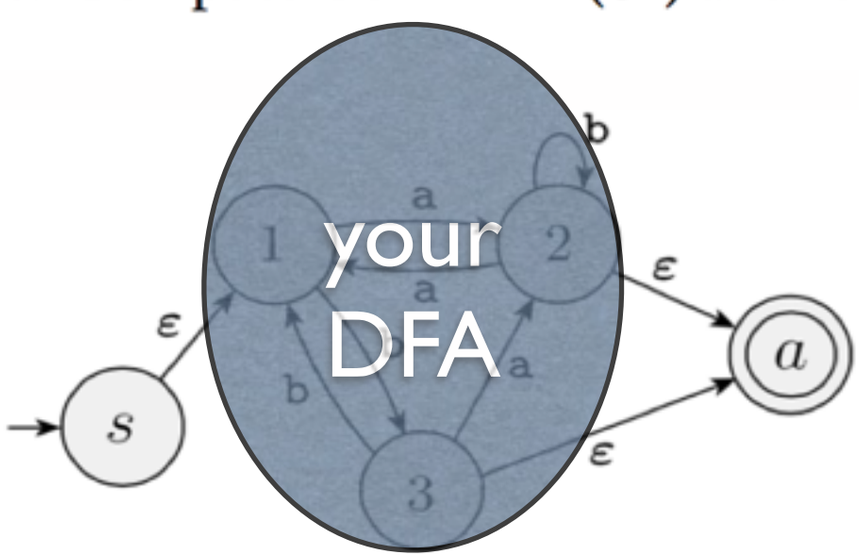
$$Q' = Q - \{q_{\text{rip}}\},$$

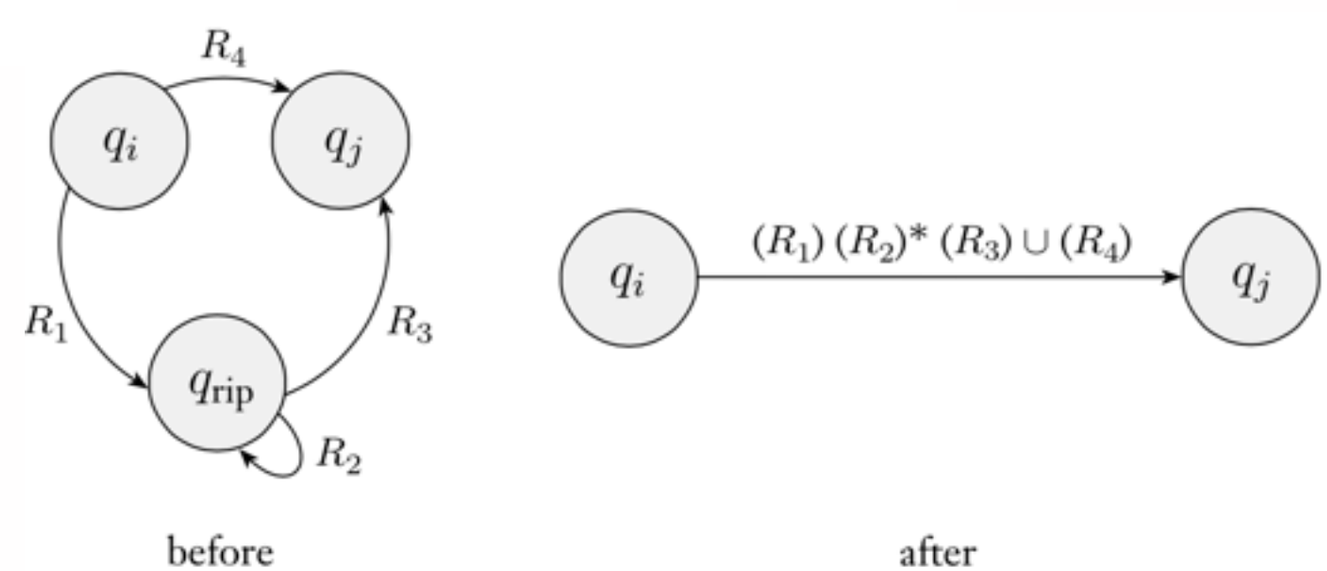
and for any $q_i \in Q' - \{q_{\text{accept}}\}$ and any $q_j \in Q' - \{q_{\text{start}}\}$, let

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$

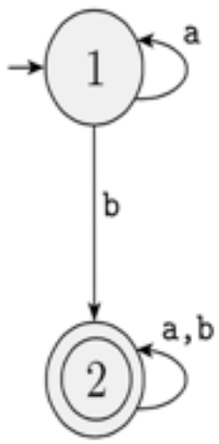
for $R_1 = \delta(q_i, q_{\text{rip}})$, $R_2 = \delta(q_{\text{rip}}, q_{\text{rip}})$, $R_3 = \delta(q_{\text{rip}}, q_j)$, and $R_4 = \delta(q_i, q_j)$.

4. Compute CONVERT(G') and return this value.

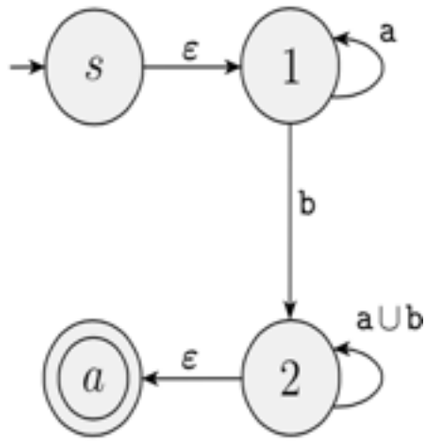




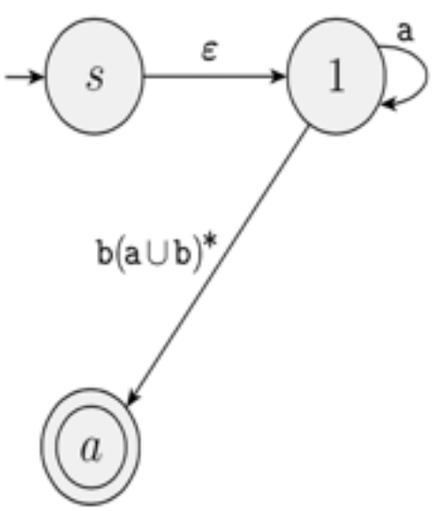
Examples: DFA=>RE



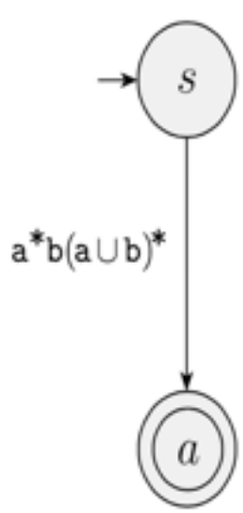
(a)



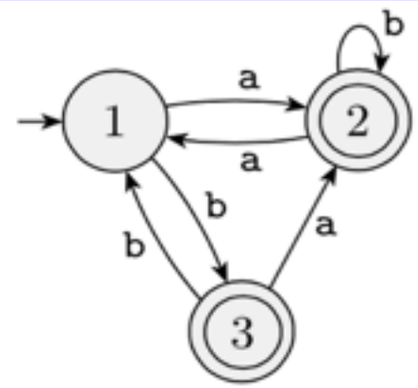
(b)



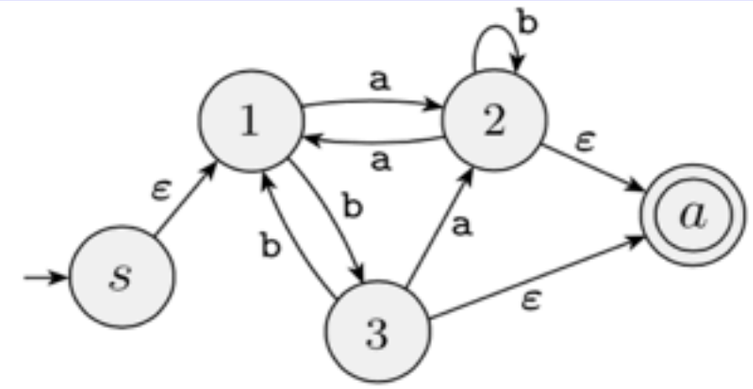
(c)



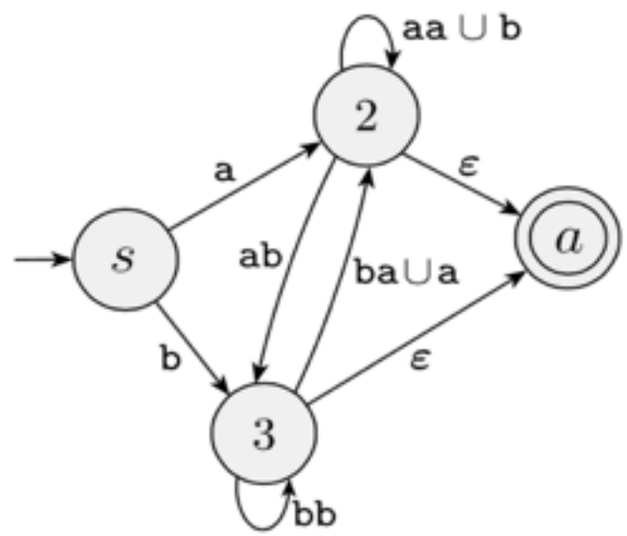
(d)



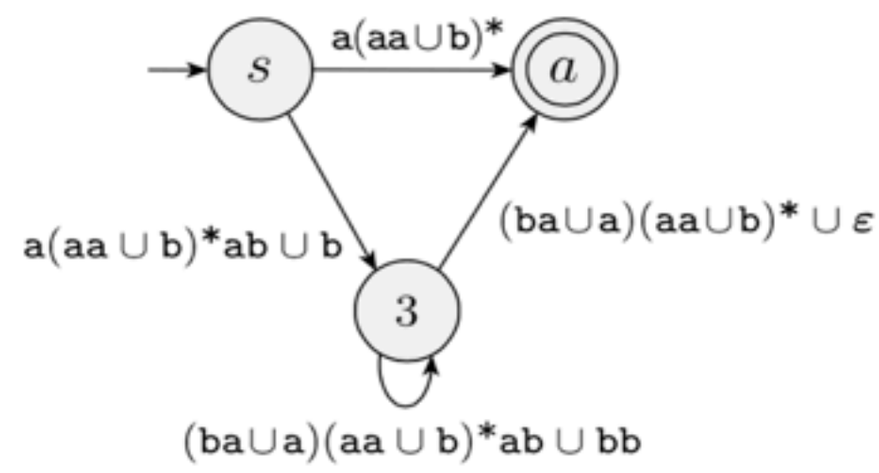
(a)



(b)



(c)



(d)

$(a(aa\cup b)^*ab\cup b)((ba\cup a)(aa\cup b)^*ab\cup bb)^*((ba\cup a)(aa\cup b)^*\cup\varepsilon)\cup a(aa\cup b)^*$

From Regular to Non-Regular Languages

Sipser 1.4, Linz 4.3

Non-Regular: Pumping Lemma

NON-REGULAR LANGUAGES

$B = \{0^N 1^N \mid N \geq 0\}$

$C = \{w \mid w$ has an equal number of 0's and 1's$\}$

$D = \{w \mid w$ has an equal number of "01"s and "10"s$\}$

SURPRISE! This is regular. why?!

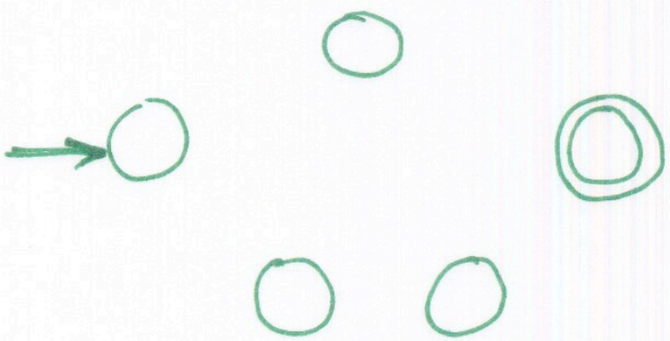
HOW CAN WE PROVE A LANGUAGE IS NOT REGULAR?

THE PUMPING LEMMA!

How to prove non-regularity?

IMAGINE A F.S.M. THAT GENERATES REALLY LONG STRINGS.

- The F.S.M. will have a small number of states.



$|Q| = n$
$= 5$

How can we generate a long string?

TAKE A CYCLIC PATH

How long can the string be without containing a cycle?

$|s| < n$

Any string of length **5** (or longer) must take a cycle.

KEY IDEA:

"If you can go around the cycle once, then you can go around it two times, and that string MUST ALSO BE IN the language."

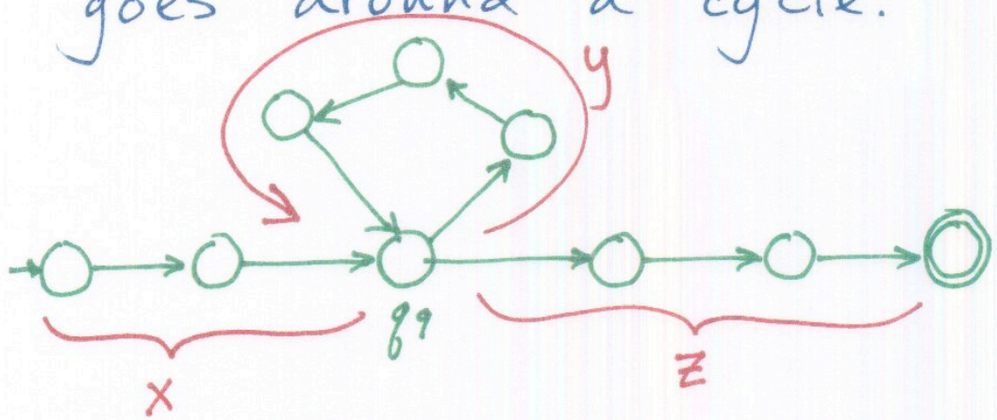
(You can also go around the cycle lots of times. ("i" times))

"You can also skip the cycle, and that string MUST ALSO BE in the language."

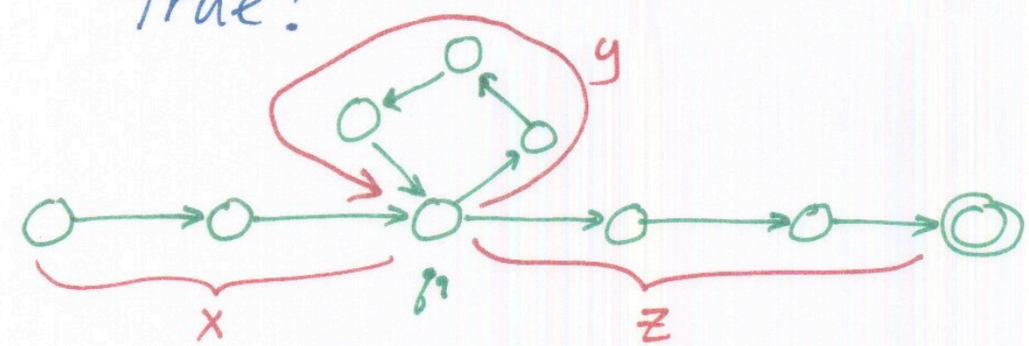
(You can go around the cycle $i = 0$ times.)

Infinite RLs Must Have Cycles

Look at a string "s" that goes around a cycle.



What can we say must be true?



String $s = xyz$

<u>But</u> <u>Note</u>: All strings of the form

$$s = xy^iz$$

are in the language.

$$s = xy^iz \in A \text{ for } i \geq 0$$

This pumping length is a property of the language, and not of any specific Finite State Machine.

If A is a regular language, and if s is a sufficiently long enough string

i.e., $|s| \geq P$

The PUMPING LENGTH. Depends on the language, not any specific F.S.M. Long enough that some FSM will have a cycle.

then s can be divided up

$$s = xyz$$

such that:

xy^iz is in the language, for $i \geq 0$

$|y| > 0$ ← The cycle has at least one edge in it.

$|xy| \leq P$ ← You have to hit the cycle before the string ends

Use Pumping Lemma

PUMPING LEMMA

If A is a REGULAR LANGUAGE, then A has a pumping length p such that <u>any</u> string s may be divided into 3 pieces

$s = x\,y\,z$

As long as: $|s| \geq p$

such that all these conditions hold.

CONDITION 1

$xy^iz \in A$ for every $i \geq 0$

CONDITION 2

$|y| > 0$

CONDITION 3

$|xy| \leq p$

USE THE PUMPING LEMMA" TO PROVE THAT A LANGUAGE "A" IS NOT REGULAR.

BY CONTRADICTION:

- ASSUME A IS REGULAR.
- IT HAS A PUMPING LENGTH. SO CALL IT "p".
- ALL STRINGS LONGER THAN p CAN BE PUMPED. $|s| \geq p$
- FIND A STRING "s" IN A. SUCH THAT $|s| \geq p$

  Just need to find one string.
- DIVIDE s INTO xyz.
- SHOW THAT $xy^iz \notin A$ FOR SOME.
- THEN CONSIDER <u>ALL</u> WAYS THAT s CAN BE DIVIDED INTO xyz.
- SHOW THAT NONE OF THESE CAN SATISFY ALL THE 3 PUMPING CONDITIONS AT THE SAME TIME.
- S CANNOT BE PUMPED $\Rightarrow$ <u>CONTRADICTION</u>

Example 1

EXAMPLE

Let $B = \{0^n 1^n \mid n \geq 0\}$

Prove B is not regular.

Assume that B is regular.

B must have a pumping length.

Let "p" be the pumping length.

The string we will use to get the contradiction is

$$s = 0^p 1^p$$

Let's divide s into pieces xyz.

CASE 1: The "y" is in the zeros part.

CASE 2: The "y" is in the ONES part.

CASE 3: The "y" has zeros and ones

$$0^7 1^7 = 0\,0\underbrace{0\,0\,0\,0}_{y}\,0\,1\,1\underbrace{1\,1\,1\,1}_{y}\,1\,1\,1$$

CASE 1: $0\,0\underbrace{0\,0\,0}_{y}\,0\,1\,1\,1\,1\,1$

So xy^iz must be in B.

$$xy^2z = 0\,0\underbrace{0\,0\,0}_{y}\,\underbrace{0\,0\,0}_{y}\,0\,1\,1\,1\,1\,1$$

CASE 2: $0\,0\,0\,0\,0\,1\underbrace{1\,1\,1}_{y}\,1$

$$xy^2z = 0\,0\,0\,0\,0\,1\underbrace{1\,1\,1}_{y}\,\underbrace{1\,1\,1}_{y}\,1$$

CASE 3: $0\,0\,0\underbrace{0\,0\,1\,1}_{y}\,1\,1\,1$

$$xy^2z = 0\,0\,0\,\underbrace{0\,0\,1\,1}_{y}\,\underbrace{0\,0\,1\,1}_{y}\,1\,1\,1$$

Not of the form $0^n 1^n$

ALSO RECALL CONDITION 3

$$|xy| \leq p$$

$$s = 0^p 1^p = \overbrace{\underbrace{0\,0}_{x}\,\underbrace{0\,0\,0}_{y}}^{p} \,\vdots\, 1\,1\,1\,1\,1$$

So cases 2 and 3 could be ruled out that way too!

Examples 2 & 3

EXAMPLE

Let $F = \{ww \mid w \in \{0,1\}^*\}$

Show F is not regular.

PROOF

- Assume it is regular.
- Let p be the pumping length of F.
- Let's use $s = 0^p 1 0^p 1$
- Now split s into 3 pieces

  $s = xyz$

  Condition 2: $|y| > 0$

  condition 3: $|xy| \leq p$

$p=7$: $0^7 1 0^7 1 = 0\underbrace{000000}_{y}0 1 0000000 1$

- So y must be all zeros.
- So xy^2z is not in F.
- Contradiction!

F is <u>not</u> regular.

EXAMPLE 1.76

Here we demonstrate a nonregular unary language. Let $D = \{1^{n^2} \mid n \geq 0\}$. In other words, D contains all strings of 1s whose length is a perfect square. We use the pumping lemma to prove that D is not regular. The proof is by contradiction.

Assume to the contrary that D is regular. Let p be the pumping length given by the pumping lemma. Let s be the string 1^{p^2}. Because s is a member of D and s has length at least p, the pumping lemma guarantees that s can be split into three pieces, $s = xyz$, where for any $i \geq 0$ the string xy^iz is in D. As in the preceding examples, we show that this outcome is impossible. Doing so in this case requires a little thought about the sequence of perfect squares:

$$0, 1, 4, 9, 16, 25, 36, 49, \ldots$$

Note the growing gap between successive members of this sequence. Large members of this sequence cannot be near each other.

Now consider the two strings xyz and xy^2z. These strings differ from each other by a single repetition of y, and consequently their lengths differ by the length of y. By condition 3 of the pumping lemma, $|xy| \leq p$ and thus $|y| \leq p$. We have $|xyz| = p^2$ and so $|xy^2z| \leq p^2 + p$. But $p^2 + p < p^2 + 2p + 1 = (p+1)^2$. Moreover, condition 2 implies that y is not the empty string and so $|xy^2z| > p^2$. Therefore, the length of xy^2z lies strictly between the consecutive perfect squares p^2 and $(p+1)^2$. Hence this length cannot be a perfect square itself. So we arrive at the contradiction $xy^2z \notin D$ and conclude that D is not regular. ■

Summary

REGULAR LANGUAGES
AND
FINITE STATE MACHINES

SOME QUESTIONS THAT WE CAN ANSWER...

"DECIDABLE QUESTIONS"
CAN WRITE A PROGRAM.
THE PROGRAM WILL ALWAYS TERMINATE.

GIVEN A F.S.M., WHAT IS THE MINIMAL EQUIVALENT F.S.M?

DO TWO FSMs ACCEPT THE SAME LANGUAGE?
(MINIMIZE EACH OF THEM & COMPARE THE GRAPHS.)

IS THE LANGUAGE EMPTY? OR INFINITE?

ARE TWO REGULAR EXPRESSIONS EQUIVALENT?

ALMOST EVERY PROBLEM ABOUT REGULAR LANGUAGES IS DECIDABLE!

... BUT MAY BE NP-COMPLETE, i.e., EXPONENTIAL IN THE NUMBER OF STATES.

SUMMARY

Regular Expressions

$01(11011)^*00(11|00)11$

$\text{Letter}\ (\text{Letter} \cup \text{Digit})^*$

Regular Expressions
||
Regular Languages
||
Finite State Machines
// \\
DFA = NFA

Summary: Machines, Languages, Grammars

- we model computation by three concepts
  - machine (automaton) -- accepts or rejects a string
    - the set of strings accepted by machine M is language L(M)
  - language -- a (possibly infinite) set of strings
  - grammar -- a (recursive) way of generating strings
    - the set of strings generated by grammar G is language L(G)
- we'll see a lot about converting between M, L, G

machine	language	grammar
finite-state	regular (RegExps)	regular
pushdown	context-free	context-free
Turing	recursively enumerable	unrestricted