

Theory of Computation

Introduction

From high level

- From a high level what this course is about?
 - Given a set, say S .
 - Let $A \subseteq S$.
 - Both S and A are well defined.
 - We are given an element $x \in S$, and asked to find whether this x is in A or not.
 - That's all !

Surprises !!

- Surprising things
 - This is related to decision problems.
 - S is set all face images. A is set of images of a particular person. {Face verification}
 - S is set of all graphs. A is set of graphs with Hamiltonian cycle.
 - Sometimes this is an **unsolvable** problem. ??
 - Sometimes this is an **easy** task, sometimes quite a **difficult** one.
 - Recall, $O(n^2)$ is time consuming than $O(n)$ algorithm.

We want general enough set

- Set of strings over some alphabet like {0,1}.
- For example set of strings that end with a 0,
 $\{0, 00, 10, 000, 010, 100, 110, \dots\}$
- Eg2: Each string in the set can be seen as a positive binary number and let the set be the set of prime numbers.
 - Given a number (binary string of 0s and 1s) you want to find whether this is in the set (prime) or not (not a prime).

Why strings are chosen?

- Any data element like number or image or anything can be represented as a string.
 - Can we say DNA code represents a human being?
- Even a method which solves a problem can be represented as a string.
- A proof can be represented as a string.
- So strings over an alphabet gives us power to represent the things... that is we have *languages of strings to represent the things*.

What will you learn from this course?

- How to define a computer? **Automata theory**
- Are there problems that a computer cannot solve? If so, can we find one such problem? **Computability theory**
- For problems that a computer can solve, some problems are easy (e.g., sorting) and some are difficult (e.g., time-table scheduling). Any systematic way to classify problems? **Complexity theory**

Syllabus

- **Syllabus:**
- **Unit – 1 [8 Hours]:** Introduction - Alphabets, Strings and Languages, Automata and Grammars; Deterministic finite Automata (DFA) - Formal Definition, Simplified notation, State transition graph, Transition table, Language of DFA; Nondeterministic finite Automata (NFA) - NFA with epsilon transition, Language of NFA, Equivalence of NFA and DFA, Minimization of Finite Automata, Distinguishing one string from other
- **Unit – 2 [8 Hours]:** Regular Expression (RE) - Definition, Operators of regular expression and their precedence, Algebraic laws for Regular expressions; Relation with FA - Regular expression to FA, DFA to Regular expression; Non Regular Languages - Pumping Lemma for regular Languages, Application of Pumping Lemma; Properties - Closure properties of Regular Languages, Decision properties of Regular Languages, Applications and Limitation of FA
- **Unit – 3 [8 Hours]:** Context Free Grammar (CFG) - Definition, Examples, Derivation, Derivation trees; Ambiguity in Grammar - Inherent ambiguity, Ambiguous to Unambiguous CFG; Normal forms for CFGs - Useless symbols, Simplification of CFGs, CNF and GNF; Context Free Languages (CFL) - Closure properties of CFLs, Decision Properties of CFLs, Emptiness, Finiteness and Membership, Pumping lemma for CFLs
- **Unit – 4 [8 Hours]:** Push Down Automata (PDA) - Description and definition, Instantaneous Description, Language of PDA; Variations of PDA - Acceptance by Final state, Acceptance by empty stack, Deterministic PDA; Equivalence of PDA and CFG - CFG to PDA and PDA to CFG
- **Unit – 5 [8 Hours]:** Turing machines (TM) - Basic model, definition and representation, Instantaneous Description; Variants of Turing Machine - TM as Computer of Integer functions, Universal TM; Church's Thesis; Language acceptance by TM - Recursive and recursively enumerable languages;
- **Unit – 6 [8 Hours]:** Decidability - Halting problem, Introduction to Undecidability, Undecidable problems about TMs; Complexity - Time Complexity, Problem classes - P, NP, NP-Hard, NP-Complete.

Text Books

Text Books:

- John E. Hopcroft, Rajeev Motwani and Jeffrey D. Ullman, Introduction to Automata Theory, Languages and Computation, Pearson Education, 3rd edition, 2014, ISBN: 978-0321455369
- Michael Sipser, Introduction to the Theory of Computation, Cengage Learning, 3rd Edition, 2014, ISBN: 978-8131525296

Reference Books:

- John C. Martin, Introduction to Languages and the Theory of Computation, McGraw-Hill Education, 4th edition, 2010, ISBN: 978-0073191461
- Bernard M. Moret, The Theory of Computation, Pearson Education, 2002, ISBN: 978-8131708705

Evaluation

- Quiz: Best (n-1) 20 marks
- Mid1: 20 marks
- Mid2: 25 marks
- Endsem: 35 marks
- Can be updated (and will be informed).

Mathematics Review I

(Basic Terminology)

- Unlike other CS courses, this course is a MATH course...
- We will look at a lot of definitions, theorems and proofs
- This lecture: reviews basic math notation and terminology
 - Set, Sequence, Function, Graph, String...
- Also, common proof techniques
 - By construction, induction, contradiction

Set

- A **set** is a group of items
- One way to describe a set: list every item in the group inside { }
 - E.g., { 12, 24, 5 } is a set with three items
- When the items in the set has trend: use ...
 - E.g., { 1, 2, 3, 4, ... } means the set of natural numbers
- Or, state the rule
 - E.g., { $n \mid n = m^2$ for some positive integer m } means the set { 1, 4, 9, 16, 25, ... }
- A set with no items is an **empty set** denoted by { } or \emptyset

Set

- The order of describing a set does not matter
 - $\{ 12, 24, 5 \} = \{ 5, 24, 12 \}$
- Repetition of items does not matter too
 - $\{ 5, 5, 5, 1 \} = \{ 1, 5 \}$
- Membership symbol \in
 - $5 \in \{ 12, 24, 5 \}$ $7 \notin \{ 12, 24, 5 \}$

- How many items are in each of the following set?
 - { 3, 4, 5, ..., 10 }
 - { 2, 3, 3, 4, 4, 2, 1 }
 - { 2, {2}, {{1,2,3,4,5,6}} }
 - \emptyset
 - { \emptyset }

Set

Given two sets A and B

- we say $A \subseteq B$ (read as A is a **subset** of B) if every item in A also appears in B
 - E.g., A = the set of primes, B = the set of integers
- we say $A \subsetneq B$ (read as A is a **proper subset** of B) if $A \subseteq B$ but $A \neq B$

Warning: Don't be confused with \in and \subseteq

- Let $A = \{1, 2, 3\}$. Is $\emptyset \in A$? Is $\emptyset \subseteq A$?

Union, Intersection, Complement

Given two sets A and B

- $A \cup B$ (read as the **union** of A and B) is the set obtained by combining all elements of A and B in a single set
 - E.g., $A = \{1, 2, 4\}$ $B = \{2, 5\}$
 $A \cup B = \{1, 2, 4, 5\}$
- $A \cap B$ (read as the **intersection** of A and B) is the set of common items of A and B
 - In the above example, $A \cap B = \{2\}$
- \bar{A} (read as the **complement** of A) is the set of items under consideration not in A

Set

- The **power set** of A is the set of all subsets of A , denoted by 2^A
 - E.g., $A = \{0, 1\}$
 $2^A = \{\{\}, \{0\}, \{1\}, \{0, 1\}\}$
 - How many items in the above power set of A ?
- If A has n items, how many items does its power set contain? Why?

Sequence

- A **sequence** of items is a list of these items in some order
- One way to describe a sequence: list the items inside ()
 - (5, 12, 24)
- Order of items inside () matters
 - (5, 12, 24) \neq (12, 5, 24)
- Repetition also matters
 - (5, 12, 24) \neq (5, 12, 12, 24)
- Finite sequences are also called **tuples**
 - (5, 12, 24) is a 3-tuple
 - (5, 12, 12, 24) is a 4-tuple

Sequence

Given two sets A and B

- The **Cartesian product** of A and B, denoted by $A \times B$, is the set of all possible 2-tuples with the first item from A and the second item from B
 - E.g., $A = \{1, 2\}$ and $B = \{x, y, z\}$
 $A \times B = \{(1,x), (1,y), (1,z), (2,x), (2,y), (2,z)\}$
- The Cartesian product of k sets, A_1, A_2, \dots, A_k , denoted by $A_1 \times A_2 \times \dots \times A_k$, is the set of all possible k-tuples with the i^{th} item from A_i

Functions

- A function takes an input and produces an output
- If f is a function, which gives an output b when input is a , we write
$$f(a) = b$$
- For a particular function f , the set of all possible input is called f 's domain
- The outputs of a function come from a set called f 's range

Functions

- To describe the property of a function that it has domain D and range R, we write

$$f : D \rightarrow R$$

- E.g., The function add (to add two numbers) will have an input of two integers, and output of an integer
 - We write: $\text{add} : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$

Strings

- An **alphabet** = a set of characters
 - E.g., The English Alphabet = {A,B,C,...,Z}
- A **string** = a sequence of characters
- A string over an alphabet Σ
 - A sequence of characters, with each character coming from Σ
- The **length** of a string w , denoted by $|w|$, is the number of characters in w
- The **empty string** (written as ε) is a string of length 0

Strings

Let $w = w_1w_2\dots w_n$ be a string of length n

- A **substring** of w is a consecutive subsequence of w (that is, $w_iw_{i+1}\dots w_j$ for some $i \leq j$)
- The **reverse** of w , denoted by w^R , is the string $w_n\dots w_2 w_1$
- A set of strings is called a **language**

PROOF TECHNIQUES

We look at

- Proof by contradiction
- Proof by construction
- Proof by induction
-
- But we may use some other also..
- For eg., Proof by counter example to disprove a statement...

By Contradiction

- One common way to prove a theorem is to assume that the theorem is false, and then show that this assumption leads to an obviously false consequence (also called a contradiction)
- This type of reasoning is used frequently in everyday life, as shown in the following example

By Contradiction

- Jack sees Jill, who just comes in from outdoor
- Jill looks completely dry
- Jack knows that it is not raining
- Jack's proof:
 - If it *were* raining (the assumption that the statement is false), Jill will be wet.
 - The consequence is: "Jill is wet" AND "Jill is dry", which is obviously false
 - Therefore, it must not be raining

By Contradiction [Example 1]

- Let us define a number is **rational** if it can be expressed as p/q where p and q are integers; if it cannot, then the number is called **irrational**
- E.g.,
 - 0.5 is rational because $0.5 = 1/2$
 - 2.375 is rational because $2.375 = 2375 / 1000$

By Contradiction

- Theorem: $\sqrt{2}$ (the square-root of 2) is irrational.
- How to prove?
- First thing is ...

Assume that $\sqrt{2}$ is rational

By Contradiction

- Proof: Assume that $\sqrt{2}$ is rational. Then, it can be written as p/q for some positive integers p and q .
- In fact, we can further restrict that p and q does not have common factor.
 - If D is a common factor of p and q , we use $p' = p/D$ and $q' = q/D$ so that $p'/q' = p/q = \sqrt{2}$ and there is no common factor between p' and q'
- Then, we have $p^2/q^2 = 2$, or $2q^2 = p^2$.

By Contradiction

- Since $2q^2$ is an even number, p^2 is also an even number
 - This implies that p is an even number (why?)
- So, $p = 2r$ for some integer r
- $2q^2 = p^2 = (2r)^2 = 4r^2$
 - This implies $2r^2 = q^2$
- So, q is an even number
- Something wrong happens... (what is it?)

By Contradiction

- We now have: “ p and q does not have common factor” AND “ p and q have common factor”
 - This is a contradiction
- Thus, the assumption is wrong, so that $\sqrt{2}$ is irrational

By Contradiction [Example 2]

- Theorem (Pigeonhole principle): A total of $n+1$ balls are put into n boxes. At least one box containing 2 or more balls.
 - Proof: Assume "at least one box containing 2 or more balls" is false
 - That is, each has at most 1 or fewer ball
- Consequence: total number of balls $\leq n$
- Thus, there is a contradiction (what is that?)

Proof By Construction

- Many theorem states that a particular type of object exists
- One way to prove is to find a way to construct one such object
- This technique is called **proof by construction**

- Theorem: There exists a rational number p which can be expressed as q^r , with q and r both irrational.
- How to prove?
 - Find p, q, r satisfying the above condition
- What is the irrational number we just learnt? Can we make use of it?

By Construction

- What is the following value?
 $(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}}$
- If $\sqrt{2}^{\sqrt{2}}$ is rational, then $q = r = \sqrt{2}$ gives the desired answer
- Otherwise, $q = \sqrt{2}^{\sqrt{2}}$ and $r = \sqrt{2}$ gives the desired answer

By Induction

- Normally used to show that all elements in an infinite set have a specified property
- The proof consists of proving two things: The **basis**, and the **inductive step**

- Mathematical induction proves that we can climb as high as we like on a ladder, by proving that we can climb onto the bottom rung (**the basis**) and that from each rung we can climb up to the next one (**the inductive step**).

We consider only enumerable or countable sets
with a least element [well ordered sets]

1. The **base case**: prove that the statement holds for the first natural number n . Usually, $n = 0$ or $n = 1$;
 - rarely, but sometimes conveniently, the base value of n may be taken as a larger number, or even as a negative number (the statement only holds at and above that threshold).
2. The **step case or inductive step**: assume the statement holds for some natural number n , and prove that then the statement holds for $n + 1$.

By Induction [Example 1]

- Let $F(k)$ be a sequence defined as follows:
- $F(1) = 1$
- $F(2) = 1$
- for all $k \geq 3$, $F(k) = F(k-1) + F(k-2)$
- Theorem: For all $n \geq 1$,
$$F(1) + F(2) + \dots + F(n) = F(n+2) - 1$$

By Induction

- Let $P(k)$ means "the theorem is true when $n = k$ "
- Basis: To show $P(1)$ is true.
 - $F(1) = 1$, $F(3) = F(1) + F(2) = 2$
 - Thus, $F(1) = F(3) - 1$
 - Thus, $P(1)$ is true
- Inductive Step: To show for $k \geq 1$, $P(k) \rightarrow P(k+1)$
 - $P(k)$ is true means: $F(1) + F(2) + \dots + F(k) = F(k+2) - 1$
 - Then, we have
$$\begin{aligned} & F(1) + F(2) + \dots + F(k+1) \\ &= (F(k+2) - 1) + F(k+1) \\ &= F(k+3) - 1 \end{aligned}$$
 - Thus, $P(k+1)$ is true if $P(k)$ is true

Variants

- There can be many other types of basis and inductive step, as long as by proving both of them, they can cover all the cases
- For example, to show P is true for all $k > 1$, we can show
 - Basis: $P(1)$ is true, $P(2)$ is true
 - Inductive step: $P(k) \rightarrow P(k+2)$

Variants

- **Complete (strong) induction:** (in contrast to which the basic form of induction is sometimes known as **weak induction**)
makes the inductive step easier to prove by using a stronger hypothesis: one proves the statement $P(m + 1)$ under the assumption that $P(n)$ holds for all $n, n \leq m$.

Example: forming dollar amounts by coins

- Assume an infinite supply of 4 and 5 dollar coins.
- Prove that any whole amount of dollars greater than 12 can be formed by a combination of such coins.
- In more precise terms, we wish to show that for any amount $n \geq 12$ there exist natural numbers a and b such that $n = 4a + 5b$, where 0 is included as a natural number.
- The statement to be shown true is thus:

$$S(n) : n \geq 12 \Rightarrow \exists a, b \in \mathbb{N}. n = 4a + 5b$$

Base case: Show that $S(k)$ holds for $k = 12, 13, 14, 15$.

$$4 \cdot 3 + 5 \cdot 0 = 12$$

$$4 \cdot 2 + 5 \cdot 1 = 13$$

$$4 \cdot 1 + 5 \cdot 2 = 14$$

$$4 \cdot 0 + 5 \cdot 3 = 15$$

The base case holds.

Induction step:

For $j = 12, 13, \dots, 15, \dots, k$ we assume that the theorem is true.

For $j = k + 1$, we show that the theorem is true.

Since for $j = k, k - 1, k - 2, k - 3$ the theorem is true (why?).

So, $k - 3 = 4a + 5b$, for some nonnegative integers a and b .

Since $k + 1 = (k - 3) + 4$,

we have, $k + 1 = 4a + 5b + 4 = 4(a + 1) + 5b$. Q.E.D.

- The following is not a valid proof by induction!

By Induction?

- CLAIM: In any set of h horses, all horses are of the same color.
- PROOF: By induction. Let $P(k)$ means "the claim is true when $h = k$ "
- Basis: $P(1)$ is true, because in any set of 1 horse, all horses clearly are the same color.

By Induction?

- Inductive step:
 - Assume $P(k)$ is true.
 - Then we take any set of $k+1$ horses.
 - Remove one of them. Then, the remaining horses are of the same color (because $P(k)$ is true).
 - Put back the removed horse into the set, and remove another horse
 - In this new set, all horses are of same color (because $P(k)$ is true).
 - Therefore, all horses are of the same color!
- What's wrong?

More on Pigeonhole Principle

- Theorem: For any graph with more than two vertices, there exists two vertices whose degree are the same.
- How to prove?

For connected graphs

First, suppose that G is a connected finite simple graph with n vertices. Then every vertex in G has degree between 1 and $n - 1$ (the degree of a given vertex cannot be zero since G is connected, and is at most $n - 1$ since G is simple). Since there are n vertices in G with degree between 1 and $n - 1$, the pigeon hole principle lets us conclude that there is some integer k between 1 and $n - 1$ such that two or more vertices have degree k .

For arbitrary graphs

Now, suppose G is an arbitrary finite simple graph (not necessarily connected). If G has any connected component consisting of two or more vertices, the above argument shows that that component contains two vertices with the same degree, and therefore G does as well. On the other hand, if G has no connected components with more than one vertex, then every vertex in G has degree zero, and so there are multiple vertices in G with the same degree. \square

- As we go, we see various proofs.
- Proofs has to be formal.
- You can not scribble something and expect the examiner to interpret the answer !!

- As we go, we will
 - Proofs have
 - You can not expect the examiner to
- and expect the !!





Your TA should not become like this !

General Advise

- Use standard vocabulary, simple notation, and techniques described in the class.
- Most numeric questions are toy problems, whose answer can be obtained by common sense (often simple trial and error method is enough).
- You **have to** use the method(s) described in the **classwork** or in the **text books**. Otherwise you may not get marks.

Properties of Regular Languages

DEFINITION 1.16

A language is called a *regular language* if some finite automaton recognizes it.

- DFA and NFA are finite automaton
- So, a language recognized by DFA or NFA is a regular language.

Closure Properties

- **THEOREM 1.45** -----
 - The class of regular languages is closed under the union operation.
 - Product DFA construction proof, we have seen.
 - Now, we attempt using NFAs.

- Let $L(N_1) = A_1$, and $L(N_2) = A_2$

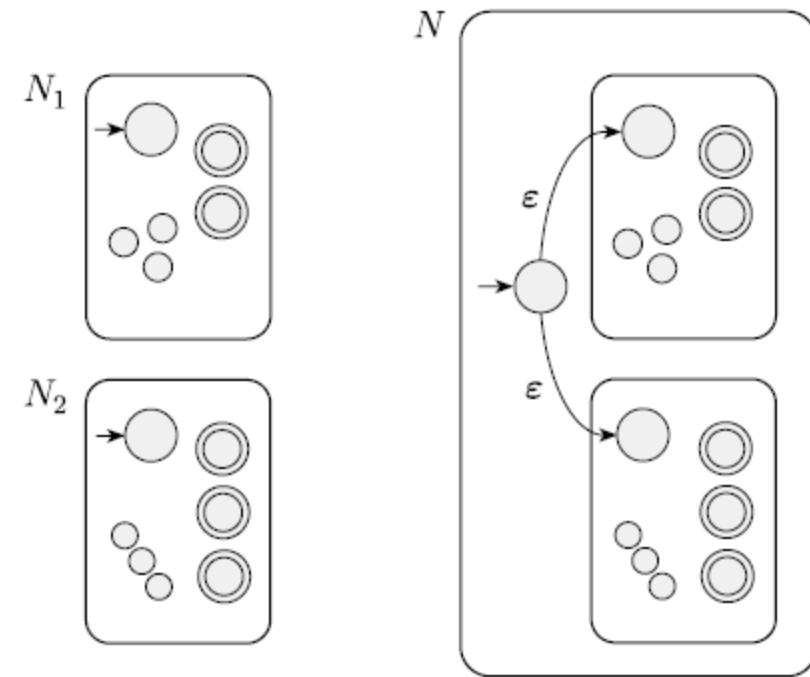


FIGURE 1.46

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

- Mathematical description of this construction is left as an exercise. {can refer to Sipser book}*

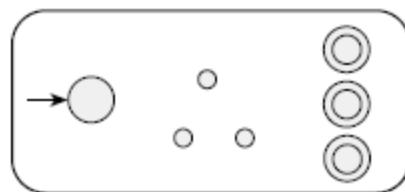
- But, for intersection, still product machine is needed. You cannot do like this for intersection.

- But, for intersection, still product machine is needed. You cannot do like this for intersection.
 - Trying to do product construction as we did for DFAs will not work.
 - Similarly complementation principle as we did with DFAs will not work for NFAs.

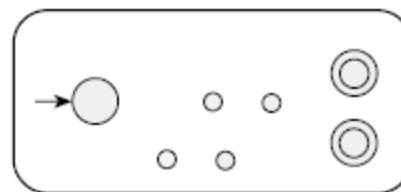
THEOREM 1.47

The class of regular languages is closed under the concatenation operation.

• N_1



N_2



N

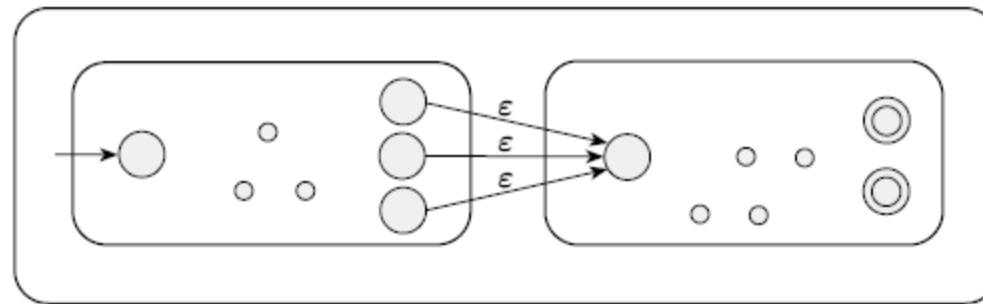


FIGURE 1.48

Construction of N to recognize $A_1 \circ A_2$

Mathematically,



PROOF

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1 , and
 $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2 .

Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$.

1. $Q = Q_1 \cup Q_2$.
The states of N are all the states of N_1 and N_2 .
2. The state q_1 is the same as the start state of N_1 .
3. The accept states F_2 are the same as the accept states of N_2 .
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

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

THEOREM 1.49

The class of regular languages is closed under the star operation.

•

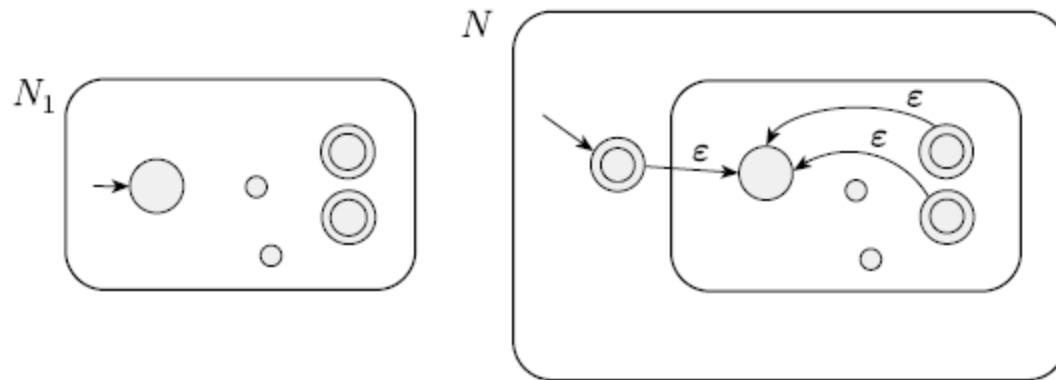


FIGURE 1.50

Construction of N to recognize A^*

PROOF Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1 .

Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize A_1^* .

1. $Q = \{q_0\} \cup Q_1$.

The states of N are the states of N_1 plus a new start state.

2. The state q_0 is the new start state.

3. $F = \{q_0\} \cup F_1$.

The accept states are the old accept states plus the new start state.

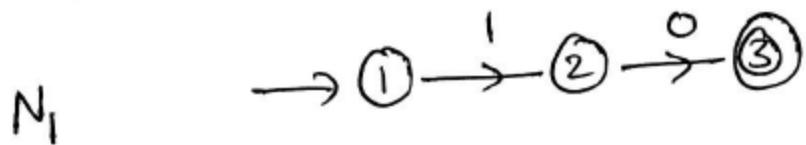
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

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

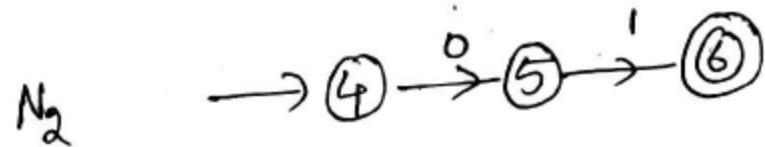
Exercise

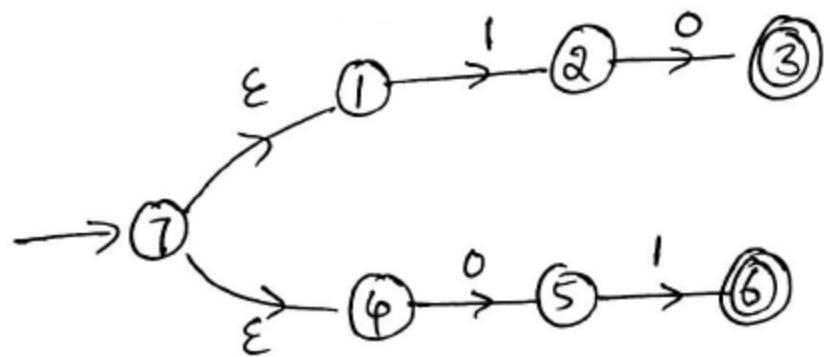
- **design a DFA to accept A^* where $A= \{10, 01\}$.**
 - Construct NFA
 - Then convert this to DFA

NFA for $\{10\}$

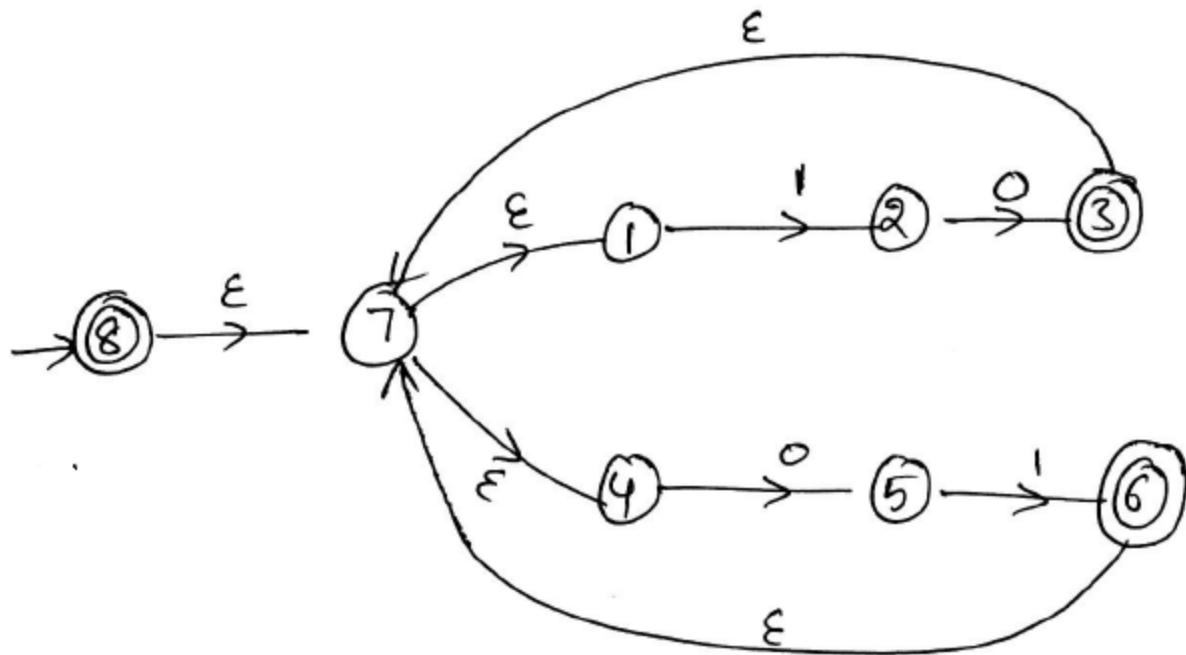


NFA for $\{01\}$





NFA N for $A = \{01, 10\}$

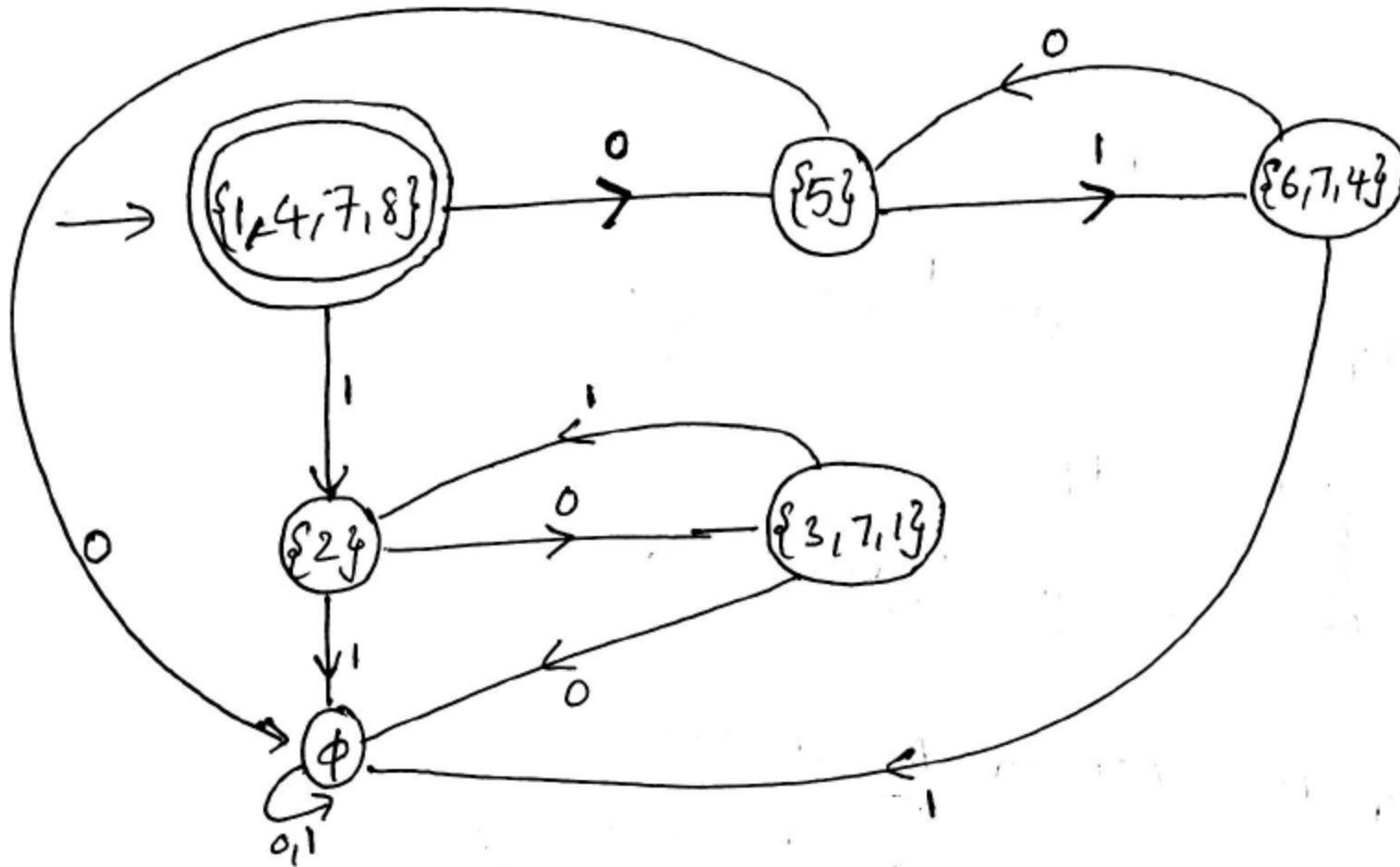


NFA for !^*

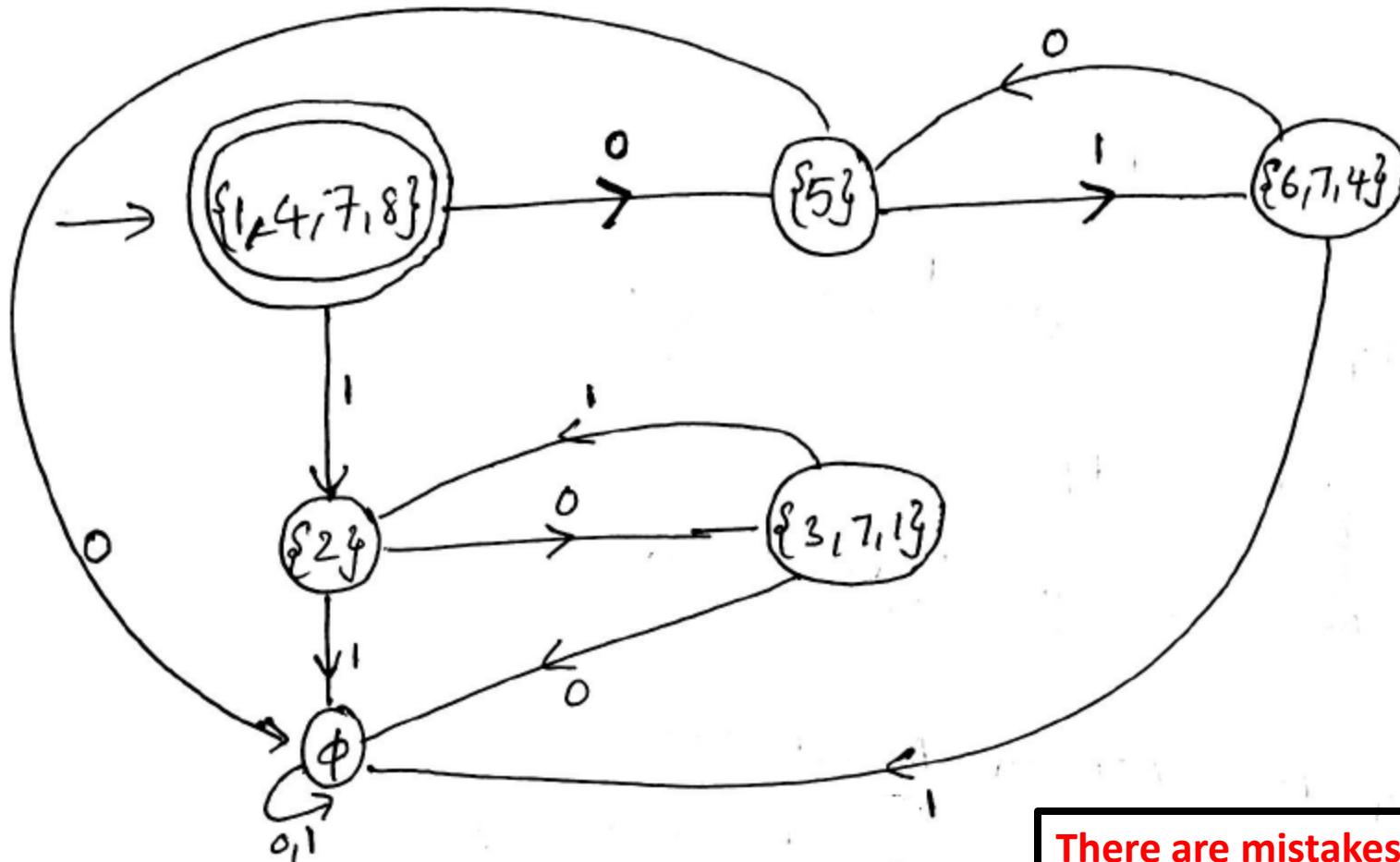
Now we should convert this to DFA.

Note that, $E(\{8\}) = \{8, 7, 1, 4\}$.

Now, can you convert this NFA in to an equivalent DFA ?



DFA for A^*



There are mistakes in this slide, try to correct them !!!

DFA for A^*

Some non-regular languages we already know are:

- Palindromes = $\{w = w^R \mid w \in \Sigma^*\}$
- Copy Language = $\{ww \mid w \in \Sigma^*\}$
- $\{a^n b^n \mid n \geq 0\}$.

Some non-regular languages we already know are:

- Palindromes = $\{w = w^R \mid w \in \Sigma^*\}$
 - Copy Language = $\{ww \mid w \in \Sigma^*\}$
 - $\{a^n b^n \mid n \geq 0\}$.
-
- But, recall $\{a^n b^n \mid 0 \leq n \leq 5\}$ is regular.
 - Every finite language is regular. {Can you prove this?}.
 - Σ^* is regular, ϕ is regular, ...

Regular Expressions

1.3

REGULAR EXPRESSIONS

In arithmetic, we can use the operations $+$ and \times to build up expressions such as

$$(5 + 3) \times 4.$$

Similarly, we can use the regular operations to build up expressions describing languages, which are called *regular expressions*. An example is:

$$(0 \cup 1)0^*.$$

- What are values of these expressions?

Where used

- Very useful to describe a set of strings having certain patterns.
 - In UNIX, `rm *.c` → removes all files ending with .c
 - Lex, a tool used in compiler generators
 - grep, awk available utilities in UNIX use regular expressions.

Meaning ...

0 means the language $\{0\}$

1 means $\{1\}$

$(0 \cup 1)$ means $\{0\} \cup \{1\}$

0^* means $\{0\}^*$

$(0 \cup 1)0^*$ actually is shorthand for $(0 \cup 1) \circ 0^*$

$$(0 \cup 1)0^* = \{0, 1\}\{0\}^*$$

Inductive Definition

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.

In items 1 and 2, the regular expressions a and ϵ represent the languages $\{a\}$ and $\{\epsilon\}$, respectively. In item 3, the regular expression \emptyset represents the empty language. In items 4, 5, and 6, the expressions represent the languages obtained by taking the union or concatenation of the languages R_1 and R_2 , or the star of the language R_1 , respectively.

Don't confuse between null string and null set

Don't confuse the regular expressions ϵ and \emptyset . The expression ϵ represents the language containing a single string—namely, the empty string—whereas \emptyset represents the language that doesn't contain any strings.

Parentheses in an expression may be omitted. If they are, evaluation is done in the precedence order: star, then concatenation, then union.

- Precedence is $*$, \circ , U
 - So, aUb^* is different from $(aUb)^*$
 $aUb^* = (a \cup (b))^*$
 - aUb^*c is same as $aU(b^*c)$
-
- Many authors use $+$ for U
So, $aUb = a+b$ But $+$ is overloaded.
Sipser reserved the $+$ to mean only one thing.

+ (positive closure)

$$R^* = R^0 \cup R^1 \cup R^2 \cup \dots \cup R^i \cup \dots$$

$$R^+ = R^1 \cup R^2 \cup \dots \cup R^i \cup \dots$$

$$R^+ = RR^*$$

$$R^* = R^+ \cup \epsilon$$

- The value of a regular expression R is nothing but the language represented by R
- When we want to distinguish between r.e. and the language represented by it. We use $L(R)$ to mean language represented by R .

We can write Σ as shorthand for regular expression $(0 \cup 1)^*$

From the context it should be clear for us by saying Σ do we mean alphabet or the language consisting of all possible strings of length 1.

Σ^* is a regular expression which is the language of all strings (including ϵ) over the alphabet Σ .

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. $\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}.$

Can you describe the language?

$$1^*(01^+)^* =$$

$$(\Sigma\Sigma)^* =$$

$$(\Sigma\Sigma\Sigma)^* =$$

$$01 \cup 10 =$$

$$0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 =$$

$$(0 \cup \varepsilon)(1 \cup \varepsilon) =$$

$$1^*\emptyset =$$

$$\emptyset^* =$$

Can you describe the language?

$1^*(01^+)^* = \{w \mid$ every 0 in w is followed by at least one 1 $\}.$

$(\Sigma\Sigma)^* =$

$(\Sigma\Sigma\Sigma)^* =$

$01 \cup 10 =$

$0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 =$

$(0 \cup \varepsilon)(1 \cup \varepsilon) =$

$1^*\emptyset =$

$\emptyset^* =$

Can you describe the language?

$1^*(01^+)^* = \{w \mid$ every 0 in w is followed by at least one 1 $\}.$

$(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}.$

$(\Sigma\Sigma\Sigma)^* =$

$01 \cup 10 =$

$0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 =$

$(0 \cup \varepsilon)(1 \cup \varepsilon) =$

$1^*\emptyset =$

$\emptyset^* =$

$1^*(01^+)^* = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}.$

$(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}.$

$(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of 3}\}.$

$01 \cup 10 = \{01, 10\}.$

$0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w \mid w \text{ starts and ends with the same symbol}\}.$

$(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}.$

$1^*\emptyset = \emptyset.$

$\emptyset^* = \{\varepsilon\}.$

Understood?

If we let R be any regular expression, we have the following identities. They are good tests of whether you understand the definition.

$$R \cup \emptyset = R.$$

Adding the empty language to any other language will not change it.

$$R \circ \epsilon = R.$$

Joining the empty string to any string will not change it.

Understood?

However, exchanging \emptyset and ε in the preceding identities may cause the equalities to fail.

$R \cup \varepsilon$ 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 $L(R \circ \emptyset) = \emptyset$.

Compilers -- Tokens

- Lexical Analysis
 - Automatic tools can be used (like Lex)
 - But , you need to describe what you want.
- For Decimal Numbers:

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

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.

Equivalence of RE with DFA/NFA

- It is somewhat surprising to note that, RE can be used to describe any regular language.
 - This is not true for other higher level languages, like CFL {We can describe a CFL by a CFG, not by an expression}.
- Now, how is that we prove this?

Proof has two directions

- Given RE, show that we can build a DFA/NFA recognizing the language given by the RE.
- Given DFA/NFA, show that we can convert this in to a RE.

To Show

- Given RE, show that we can build a NFA recognizing the language given by the RE.
- We use inductive definition of RE.

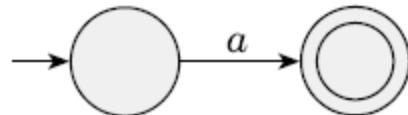
Inductive Definition of RE

DEFINITION 1.52

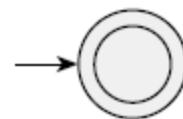
Say that R is a *regular expression* if R is

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.

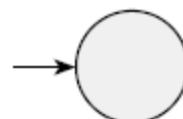
1. $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$, and the following NFA recognizes $L(R)$.



2. $R = \varepsilon$. Then $L(R) = \{\varepsilon\}$, and the following NFA recognizes $L(R)$.



3. $R = \emptyset$. Then $L(R) = \emptyset$, and the following NFA recognizes $L(R)$.

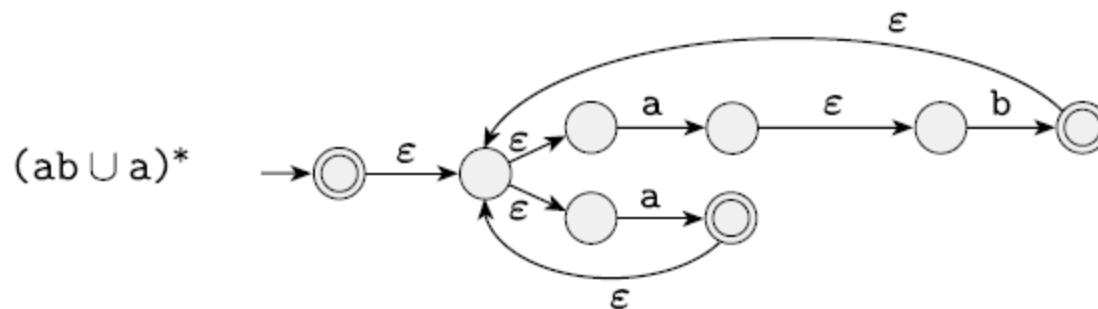
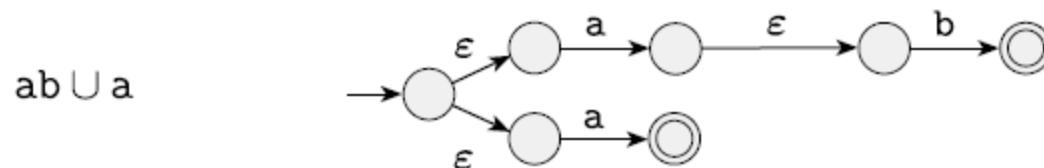
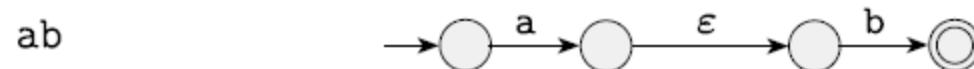
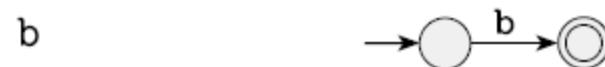


- One should be able to do these formally !!

- 4. $R = R_1 \cup R_2.$
- 5. $R = R_1 \circ R_2.$
- 6. $R = R_1^*.$

- **Use the construction proofs.**

We convert the regular expression $(ab \cup a)^*$ to an NFA in a sequence of stages.



Equivalence between RE and DFA/NFA

- The second part is,
- Given DFA/NFA convert this to equivalent RE.
- There are various ways for this.
 - The Ullman's book gives a rigorous algorithm
 - Same thing in essence is achieved by the Sipser's book in a different way.
 - We follow Sipser's book.

DFA/NFA → RE

- Go for GNFA (Generalized Nondeterministic Finite Automata)
- RE can be on arrows.

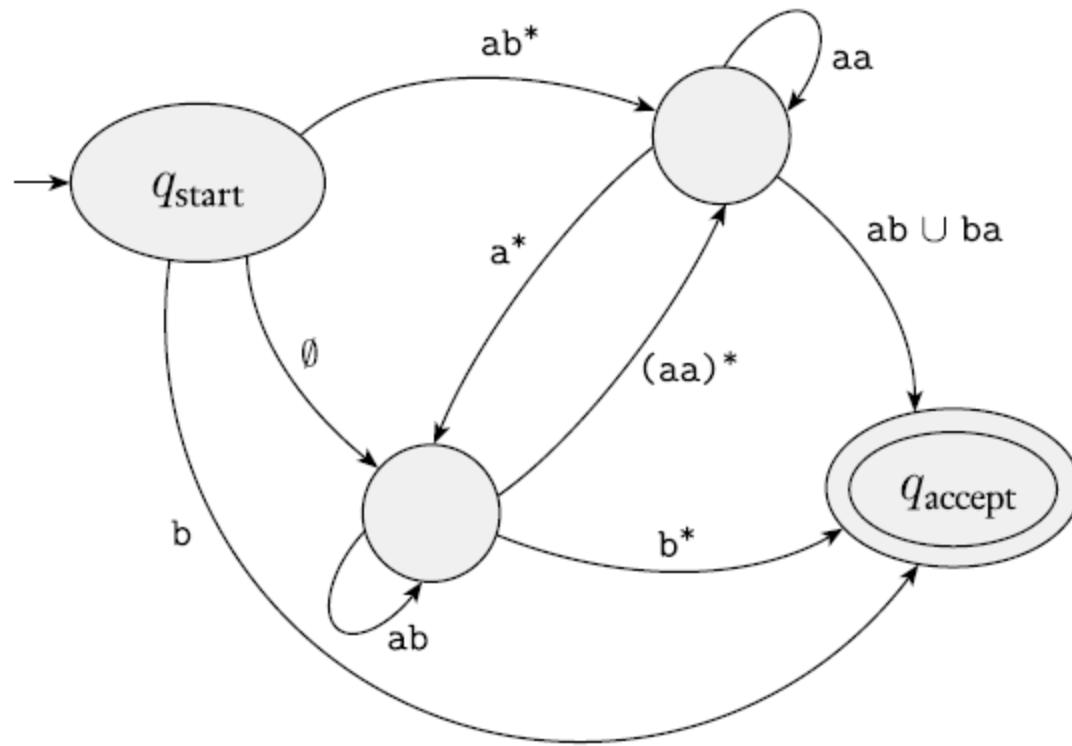


FIGURE 1.61

A generalized nondeterministic finite automaton

GNFA should -

- The start state has transition arrows going to every other state but no arrows coming in from any other state.
- There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
- Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.
- For more details, consult the Sipser's book.

What we do?

- We convert the given DFA/NFA into a GNFA.
- Then, progressively we remove all states, one by one, except for the start and accept states.

What we do?

- For example we are given a 3 state DFA

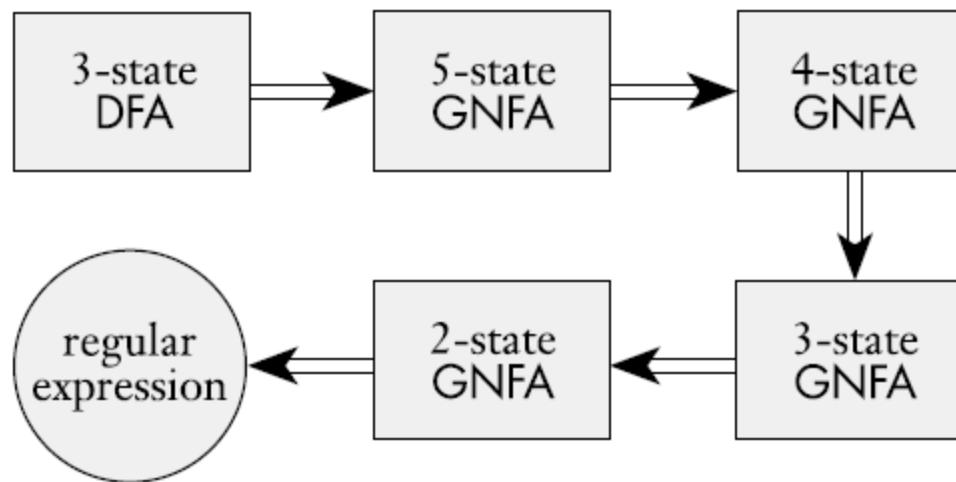


FIGURE 1.62

Typical stages in converting a DFA to a regular expression

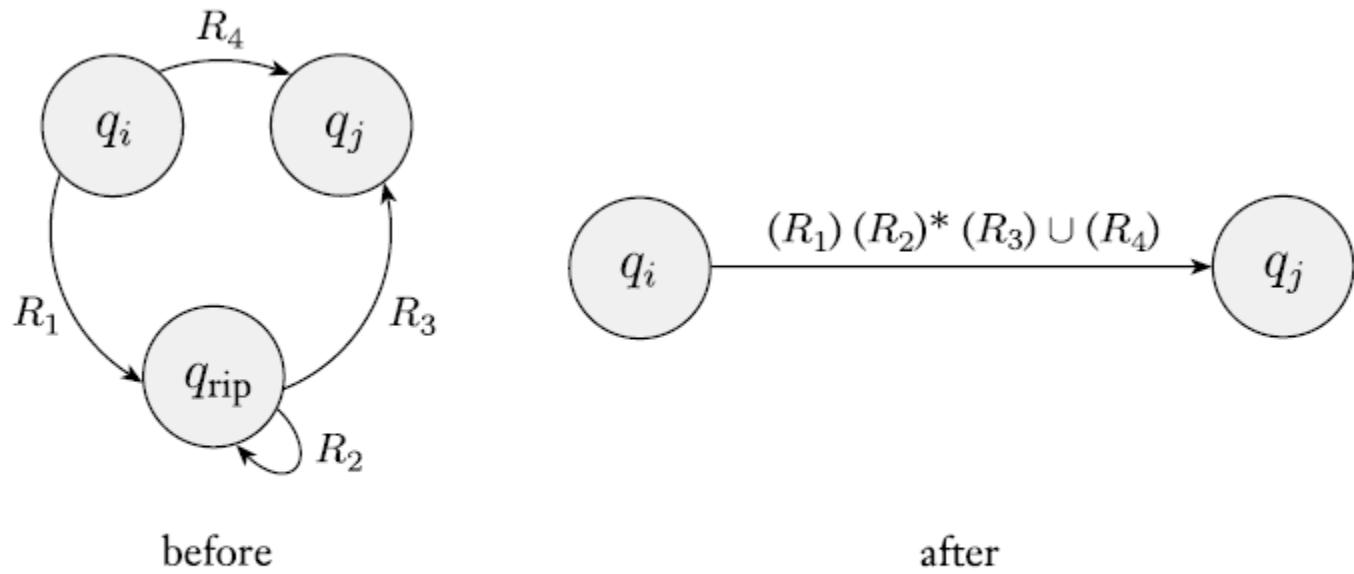
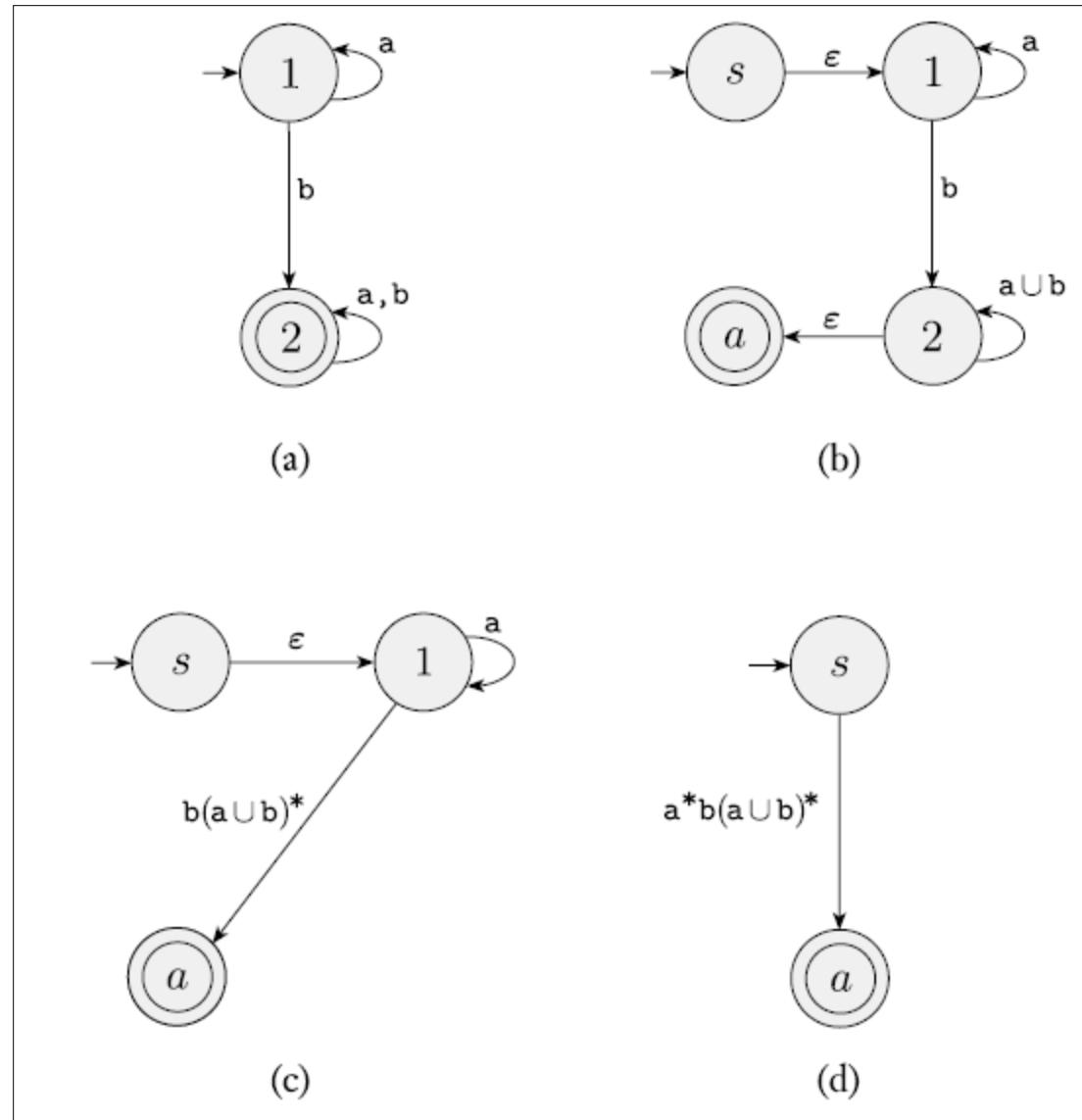


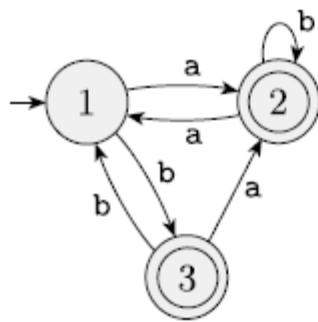
FIGURE 1.63
Constructing an equivalent GNFA with one fewer state

Example 1

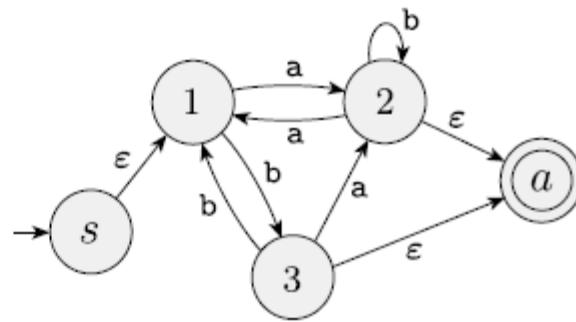


To avoid cluttering up the figure, we do not draw the arrows labeled \emptyset , even though they are present.

Example 2

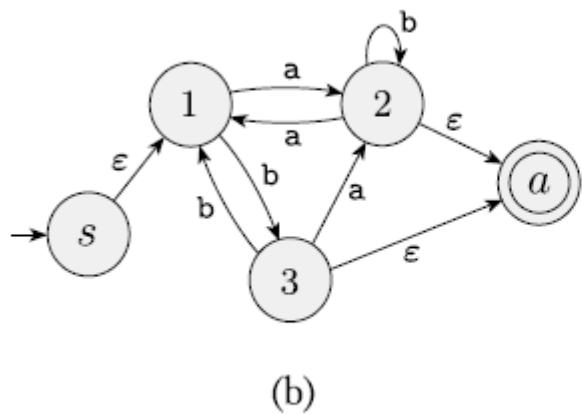


(a)

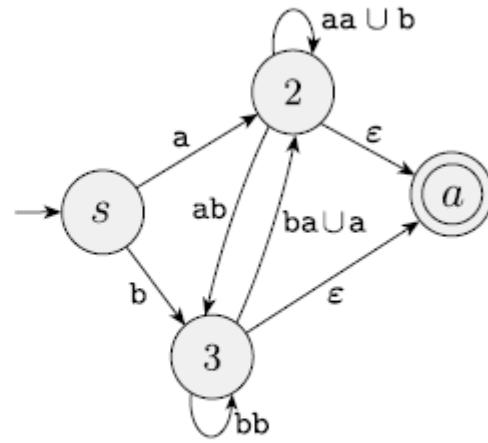


(b)

Example 2...

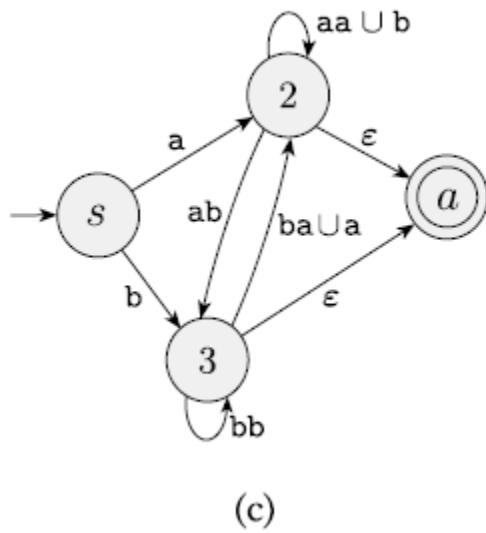


(b)

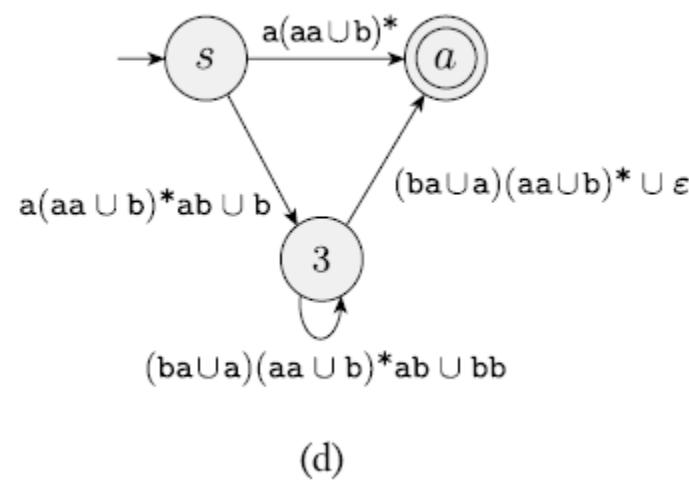


(c)

Example 2...

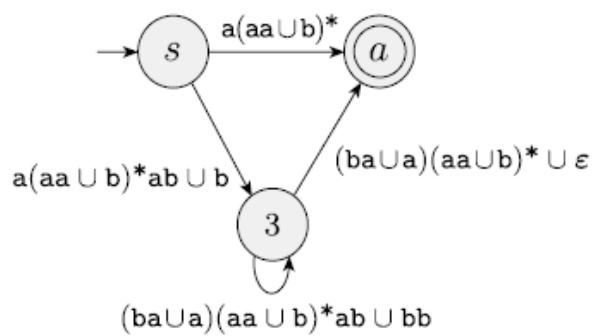


(c)

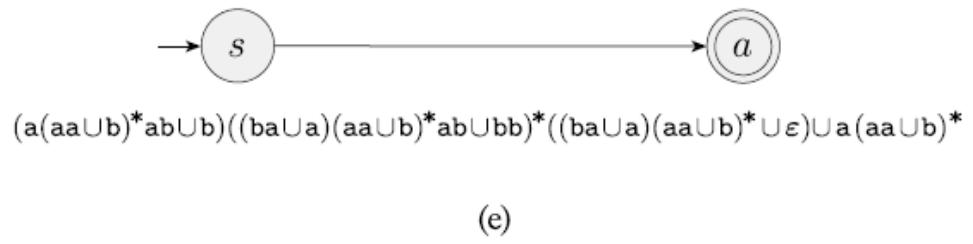


(d)

Example 2...



(d)



(e)

Laws concerning R.E.

- We overload + to mean \cup also.
 - You have to live with this notational abuse between Sipser and Ullman.
- $a+b = a \cup b$

3.4 Algebraic Laws for Regular Expressions

➤ Commutativity and Associativity

- $L + M = M + L$ (Remember, $L \cup M = M \cup L$)
- $(L + M) + R = L + (M + R) = L + M + R$
- $(LM)R = L(MR) = LMR$

➤ Left distribution and right distribution

- $L(M + N) = LM + LN$
- $(M + N)L = ML + NL$

Identities and Annihilors

- $\emptyset + L = L + \emptyset = L$. This law asserts that \emptyset is the identity for union.
- $\epsilon L = L\epsilon = L$. This law asserts that ϵ is the identity for concatenation.
- $\emptyset L = L\emptyset = \emptyset$. This law asserts that \emptyset is the annihilator for concatenation.

Idempotent Laws

- $L+L = L$
- $(L^*)^* = L^*$

How to prove?

- Inequality can be easily proved by a counter example.
- But, equality, to be proved is cumbersome.
 - You can follow your set theory knowledge to deduct that from LHS, RHS is deductible.
 - These is an established simple way of doing this.

Assuming only three regular operators,
viz., $,$ $+$, \cdot $*$ are only used.

- To test whether $E = F$ is true or false.
 1. Convert E and F to concrete regular expressions C and D , respectively, by replacing each variable by a concrete symbol.
 2. Test whether $L(C) = L(D)$. If so, then $E = F$ is a true law, and if not, then the “law” is false.
- What do you mean by **concrete r.e.** ?

Concretizing a r.e.

Let $E = P + Q(RS^*)$ be a regular expression where P, Q, R, S are some regular expressions.

Here, we say E has variables P, Q, R, S .

Concretizing E means replacing each variable in E by a distinct symbol.

In this example, we can concretize $P + Q(RS^*)$ to $a + b(cd^*)$

To prove, $P(M + N) = PM + PN$.

Concretizing L.H.S will give us $a(b + c)$

Concretizing R.H.S will give us $ab + ac$

Now, one has to show $L(a(b + c)) = L(ab + ac)$, which can be done easily.

Now, verify whether $PR = RP$ is true or false.

Concretize L.H.S in to ab

Concretize R.H.S in to ba

Now, it is clear that $L(ab) = \{ab\}$ is not equal to $L(ba) = \{ba\}$.

! Exercise 3.4.2: Prove or disprove each of the following statements about regular expressions.

- * a) $(R + S)^* = R^* + S^*$.
- b) $(RS + R)^*R = R(SR + R)^*$.
- * c) $(RS + R)^*RS = (RR^*S)^*$.
- d) $(R + S)^*S = (R^*S)^*$.
- e) $S(RS + S)^*R = RR^*S(RR^*S)^*$.

Don't use operators beyond

+ · *

- That is, stick to, regular operators only.

Extensions of the Test Beyond Regular Expressions May Fail

Consider the "law" $L \cap M \cap N = L \cap M$;

Concretizing, we get, $\{a\} \cap \{b\} \cap \{c\} = \{a\} \cap \{b\}$. This is true.

But clearly the above "law" is false.

Counter example to disprove the "law".

For example, let $L = M = \{a\}$ and $N = \emptyset$.

Clearly L.H.S and R.H.S are distinct languages.

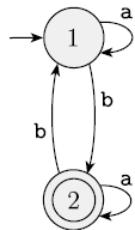
Tutorial problems on Regular Expressions

From Sipser's book –

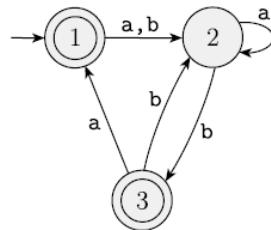
- 1.20 For each of the following languages, give two strings that are members and two strings that are *not* members—a total of four strings for each part. Assume the alphabet $\Sigma = \{a, b\}$ in all parts.

- | | |
|-------------------|--|
| a. a^*b^* | e. $\Sigma^* a \Sigma^* b \Sigma^* a \Sigma^*$ |
| b. $a(ba)^*b$ | f. $aba \cup bab$ |
| c. $a^* \cup b^*$ | g. $(\epsilon \cup a)b$ |
| d. $(aaa)^*$ | h. $(a \cup ba \cup bb)\Sigma^*$ |

- 1.21 Use the procedure described in Lemma 1.60 to convert the following finite automata to regular expressions.



(a)



(b)

- 1.28 Convert the following regular expressions to NFAs .

In all parts, $\Sigma = \{a, b\}$.

- $a(ab\bar{b})^* \cup b$
- $a^+ \cup (ab)^+$
- $(a \cup b^+)a^+b^+$

From Ullman's book –

3.4.8 Exercises for Section 3.4

Exercise 3.4.1: Verify the following identities involving regular expressions.

- * a) $R + S = S + R$.
- b) $(R + S) + T = R + (S + T)$.
- c) $(RS)T = R(ST)$.
- d) $R(S + T) = RS + RT$.
- e) $(R + S)T = RT + ST$.
- * f) $(R^*)^* = R^*$.
- g) $(\epsilon + R)^* = R^*$.
- h) $(R^*S^*)^* = (R + S)^*$.

! Exercise 3.4.2: Prove or disprove each of the following statements about regular expressions.

- * a) $(R + S)^* = R^* + S^*$.
- b) $(RS + R)^*R = R(SR + R)^*$.
- * c) $(RS + R)^*RS = (RR^*S)^*$.
- d) $(R + S)^*S = (R^*S)^*$.
- e) $S(RS + S)^*R = RR^*S(RR^*S)^*$.

Nonregular Languages

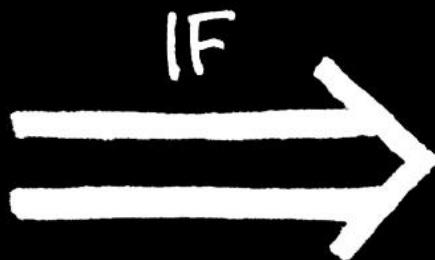
- How to show that a given language is nonregular.
- In some sense, we need to prove that No DFA is possible to recognize the language.
- How we do this?

Some properties can help us

- L is regular $\Rightarrow L$ obeys “Pumping Lemma”
- DFA must have finite number of states.
 - For the given L , we need infinite number of states in the DFA.
 - Myhill-Nerode Theorem (Gives a **necessary and sufficient** condition for regular languages).
- There are other ways ...

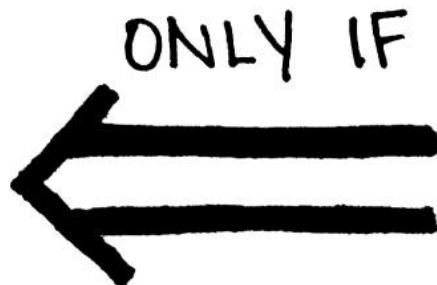
- Pumping Lemma is useful to show that L is nonregular.
- It cannot be used to show that L is regular.
- Why?

the sufficient condition



(If you assume this, you'll get what you want.)

the necessary condition



(You can't get what you want without assuming this.)

- $A \Rightarrow B$ (A being true is a sufficient condition for B to be true)
 - If A is true, we know B is true.
 - If A is false, what about B?
- $A \leq B$ (A being true is a necessary condition for B to be true)
 - If A is false, we know B is false.
 - If A is true, what about B?

- $A \Rightarrow B$ (A being true is a sufficient condition for B to be true)
 - If A is true, we know B is true.
 - If A is false, what about B?
 - If B is false, then what about A?

- L is regular $\Rightarrow L$ obeys “Pumping Lemma”
 - If L fails to obey “Pumping Lemma” then L is nonregular.
-
- Moral of the story: Never use Pumping Lemma to prove that L is regular.

- Nonregular examples

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

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

- But, the following is regular

$$D = \{w \mid w \text{ has an equal number of occurrences of 01 and 10 as substrings}\}$$

- Nonregular examples

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

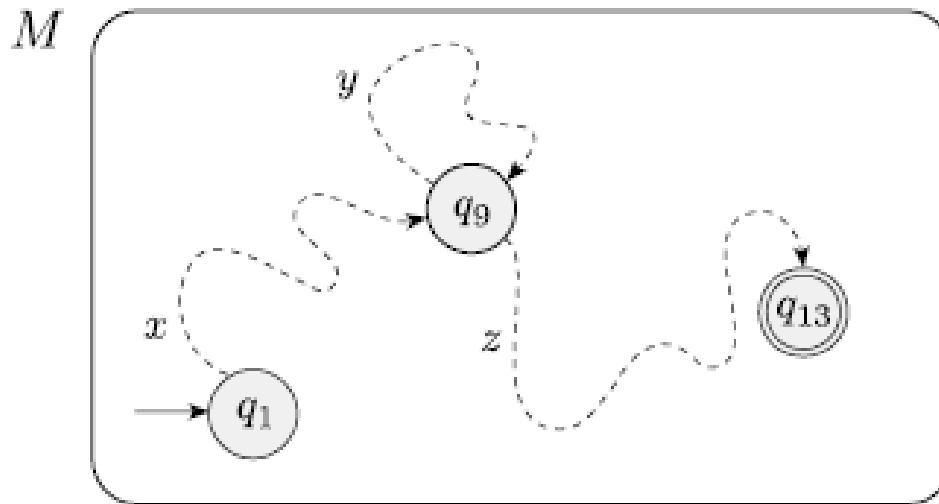
$$C = \{w \mid w \text{ has an equal number of 0s and 1s}\}$$

- But, the following is regular

$$D = \{w \mid w \text{ has an equal number of occurrences of } 01 \text{ and } 10 \text{ as substrings}\}$$

- http://www.cs.gordon.edu/courses/cps220/Notes/nonregular_languages
 - Follow the above URL for an answer to show D is regular.

Pumping Lemma



- In any DFA, if $w = xyz$ is “long enough”, then such a loop must occur. Why?

Pigeonhole Principle



The following figure shows the string s and the sequence of states that M goes through when processing s . State q_9 is the one that repeats.

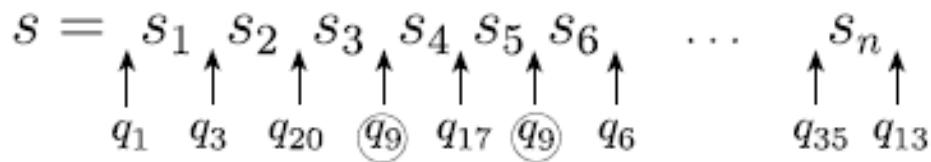


FIGURE 1.71
Example showing state q_9 repeating when M reads s

Pumping Lemma

THEOREM 1.70

Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p , then s can be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

THEOREM 1.70

Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p , then s can be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

When s is divided into xyz , either x or z may be ϵ , but condition 2 says that $y \neq \epsilon$.

Observe that without condition 2 the theorem would be trivially true.

Negation of Pumping Lemma

- There is **a** string w in L which of length atleast of the pumping length (p), where **every** division of w into xyz fails to satisfy at-least **one** of the following –
 - $|y| \neq 0$
 - $|xy| \leq p$
 - $x y^i z$ is in L for all i in $\{0,1,2,\dots\}$.

Negation of Pumping Lemma (we simplify)

- There is **a** string w in L which of length atleast of the pumping length (p), where **every** division of w into xyz that obeys
 - $|y| \neq 0$
 - $|xy| \leq p$Fails to satisfy the following for at-least one i .
 - $x y^i z$ is in L for all i in $\{0,1,2,\dots\}$.

Show that, $L = \{0^{n^2} | n \geq 1\}$ is nonregular.

Show that, $L = \{0^{n^2} | n \geq 1\}$ is nonregular.

Proof [by Pumping Lemma]:

Let p be the pumping length.

Let us choose $w = 0^{p^2}$ and we know, $|w| \geq p$.

Show that, $L = \{0^{n^2} | n \geq 1\}$ is nonregular.

Proof [by Pumping Lemma]:

Let p be the pumping length.

Let us choose $w = 0^{p^2}$ and we know, $|w| \geq p$.

Let us choose $x = 0^m$, $y = 0^n$, $z = 0^r$ and $m + n + r = p^2$.

Show that, $L = \{0^{n^2} | n \geq 1\}$ is nonregular.

Proof [by Pumping Lemma]:

Let p be the pumping length.

Let us choose $w = 0^{p^2}$ and we know, $|w| \geq p$.

Let us choose $x = 0^m$, $y = 0^n$, $z = 0^r$ and $m + n + r = p^2$.

Along with this, we have $1 \leq n \leq p$.

Show that, $L = \{0^{n^2} | n \geq 1\}$ is nonregular.

Proof [by Pumping Lemma]:

Let p be the pumping length.

Let us choose $w = 0^{p^2}$ and we know, $|w| \geq p$.

Let us choose $x = 0^m$, $y = 0^n$, $z = 0^r$ and $m + n + r = p^2$.

Along with this, we have $1 \leq n \leq p$.

Now, consider the string xy^2z . Note, $|xy^2z| = p^2 + n$.

We have, $p^2 < p^2 + n < (p + 1)^2$.

Show that, $L = \{0^{n^2} | n \geq 1\}$ is nonregular.

Proof [by Pumping Lemma]:

Let p be the pumping length.

Let us choose $w = 0^{p^2}$ and we know, $|w| \geq p$.

Let us choose $x = 0^m$, $y = 0^n$, $z = 0^r$ and $m + n + r = p^2$.

Along with this, we have $1 \leq n \leq p$.

Now, consider the string xy^2z . Note, $|xy^2z| = p^2 + n$.

We have, $p^2 < p^2 + n < (p + 1)^2$.

So, $|xy^2z|$ is not a perfect square, hence xy^2z is not in L .

Thus, Pumping Lemma failed for L .

EXAMPLE

Let B be the language $\{0^n 1^n \mid n \geq 0\}$. We use the pumping lemma to prove that B is not regular.

Let p be the pumping length.

Let $s = 0^{\lceil p/2 \rceil} 1^{\lceil p/2 \rceil}$

Because s is a member of B and s has length more than p , the pumping lemma guarantees that s can be split into three pieces, $s = xyz$, where for any $i \geq 0$ the string $xy^i z$ is in B with conditions $y \neq \epsilon$ and $|xy| \leq p$.

We consider three cases to show that this result is impossible.

1. The string y consists only of 0s. In this case, the string $xyyz$ has more 0s than 1s and so is not a member of B , violating condition 1 of the pumping lemma. This case is a contradiction.
2. The string y consists only of 1s. This case also gives a contradiction.
3. The string y consists of both 0s and 1s. In this case, the string $xyyz$ may have the same number of 0s and 1s, but they will be out of order with some 1s before 0s. Hence it is not a member of B , which is a contradiction.

A good choice for the string

EXAMPLE

Let B be the language $\{0^n 1^n \mid n \geq 0\}$. We use the pumping lemma to prove that B is not regular.

Let p be the pumping length.

Choose s to be the string $0^p 1^p$.

Because s is a member of B and s has length more than p ,
the pumping lemma guarantees that s can be split into three
pieces, $s = xyz$, where for any $i \geq 0$ the string $xy^i z$ is in B .

With conditions $y \neq \epsilon$ and $|xy| \leq p$,

y can be of only 0s,

and the string $xy^2 z$ will clearly have more 0s than 1s,

hence is not in the language.

Following are all nonregular.

1. Strings having equal number of 0s and 1s.

2. Dyck language.

$\Sigma = \{(), ()\}$. Dyck language is the set of all balanced strings like $\{(), (), (), ((())), \dots\}$

3. Palindromes (over any alphabet, other than unary alphabet).

4. Copy language, i.e., $L = \{ww \mid w \in \Sigma^*\}$.

5. $L = \{0^n 1 0^n \mid n \geq 0\}$.

6. $L = \{ww^R \mid w \in \Sigma^*\}$.

- Can you prove for each of these that PL fails.

What is wrong?

In order to show that the set of palindromes over $\Sigma = \{0,1\}$ regular,

I have chosen $s = 0^{\lceil p/2 \rceil} 1 0^{\lceil p/2 \rceil}$.

Now, I split $s = xyz$, with $y = 1$.

I can pump y as many times as I want and the resulting string is in the language.

So, the language is regular.

- There are two mistakes.

Mistakes.

- The argument is not for a particular division of the chosen s in to xyz .
But, for every division, satisfying the conditions $y \neq \epsilon$ and $|xy| \leq p$.

Mistakes.

- If you want to show that Pumping Lemma is true, then you have to show for all strings s , such that $|s| \geq p$, s can be divided in to xyz , satisfying the three conditions. Just showing it for one s is not enough.
- Note, on the otherhand, to show that Pumping Lemma is false, you can choose just one string s whose length is at-least p , but, now, for every division of s in to xyz , at-least one of the three conditions is not satisfied.
- But, the serious mistake is, you have not learnt the moral.
- Never use PL to show that a language is regular.

Set of primes – a nonregular language

- Let p be the pumping length.
- Consider $s = 0^n$ where n is prime and let $n \geq p$, represents a prime number.

Set of primes – a nonregular language

- Let p be the pumping length.
- Consider $s = 0^n$ where n is prime and let $n \geq p$, represents a prime number.
- Now, let us divide $s = 0^x 0^y 0^{(n-x-y)}$ and $y > 0, x + y \leq p$.

Set of primes – a nonregular language

- Let p be the pumping length.
- Consider $s = 0^n$ where n is prime and let $n \geq p$, represents a prime number.
- Now, let us divide $s = 0^x 0^y 0^{(n-x-y)}$ and $y > 0, x + y \leq p$.
- Consider $i = n + 1$.
- We show $0^x (0^y)^i 0^{(n-x-y)}$ does not represent a prime number

- $0^x (0^y)^i 0^{(n-x-y)} = 0^{x+y(n+1)+n-x-y}$
 $= 0^{n(y+1)}$

Show that $L = \{a^i b^j \mid i \neq j\}$ is non-regular.

Direct proof using pumping lemma is somewhat an involved one. All the trouble is in choosing an appropriate $s \in L$ for which the lemma is going to fail. After some investigation the following s is found which will ease out the proof.

Show that $L = \{a^i b^j \mid i \neq j\}$ is non-regular.

Direct proof using pumping lemma is somewhat an involved one. All the trouble is in choosing an appropriate $s \in L$ for which the lemma is going to fail. After some investigation the following s is found which will ease out the proof.

Proof:

Let the pumping length be p .

Choose $s = a^p b^{p! + p}$. Here $p!$ is factorial of p .

Let $s = xyz$ where $x = a^{p-n}$, $y = a^n$, $z = b^{p! + p}$ such that $1 \leq |n| \leq p$.

This division of s into xyz satisfies the constraints, viz., (i) $|y| \neq 0$, and (ii) $|xy| \leq p$.

We show that for some i , $xy^i z \notin L$.

Show that $L = \{a^i b^j \mid i \neq j\}$ is non-regular.

Direct proof using pumping lemma is somewhat an involved one. All the trouble is in choosing an appropriate $s \in L$ for which the lemma is going to fail. After some investigation the following s is found which will ease out the proof.

Proof:

Let the pumping length be p .

Choose $s = a^p b^{p! + p}$. Here $p!$ is factorial of p .

Let $s = xyz$ where $x = a^{p-n}$, $y = a^n$, $z = b^{p! + p}$ such that $1 \leq |n| \leq p$.

This division of s into xyz satisfies the constraints, viz., (i) $|y| \neq 0$, and (ii) $|xy| \leq p$.

We show that for some i , $xy^i z \notin L$.

Choose $i = \frac{p!}{n} + 1$. Note this i is a non-negative integer. Then $xy^i z =$

$$a^{p-n}(a^n)^{\frac{p!}{n}+1}b^{p!+p} = a^{p-n+p!+n}b^{p!+p} = a^{p!+p}b^{p!+p} \notin L.$$

An easy way to show $\{a^i b^j \mid i \neq j\}$ is nonregular

We know $a^* b^*$ is regular (why?)

We know $\{a^n b^n \mid n \geq 0\}$ is nonregular. Since PL fails for this.

Now assume $\{a^i b^j \mid i \neq j\}$ is regular.

Now this leads to a contradiction.

Can you prove these

1. $\{a^m b^n \mid m < n\}$ is nonregular.
2. $\{a^m b^n \mid m \leq n\}$ is nonregular.
3. $\{a^m b^n \mid m > n\}$ is nonregular.
4. $\{a^m b^n \mid m \geq n\}$ is nonregular.

- Prove or disprove: “every finite language is regular”.
- Prove or disprove: “every infinite language is nonregular”.

- Prove or disprove: “every finite language is regular”.
- True. We can build a NFA.
- Prove or disprove: “every infinite language is nonregular”.
- False. Counter example is: a^*b^*

- Prove or disprove : “nonregular languages are closed under union”.

- Prove or disprove : “nonregular languages are closed under union”.
- False.
- Counter example:

$\{a^n b^n | n \geq 0\} \cup \{a^i b^j | i \neq j\}$ is equal to $a^* b^*$, which is regular.

- Prove or disprove : “nonregular languages are closed under intersection”.

- Prove or disprove : “nonregular languages are closed under intersection”.
- False.
- Counter example:

$\{a^n b^n | n \geq 0\} \cap \{a^i b^j | i \neq j\}$ is empty language, which is regular.

- Prove or disprove : “nonregular languages are closed under complementation”.

- Prove or disprove : “nonregular languages are closed under complementation”.
- True.
- Proof: [by contradiction] using the fact that regular languages are closed under complementation.

Example of nonregular language that satisfies the pumping lemma

Let $\Sigma = \{\$\text{, } a\text{, } b\}$.

Consider the language $L = \{\$a^n b^n | n \geq 1\} \cup \{ \$^k w | k \neq 1, w \in \{a, b\}^*\}$.

Let p be the pumping length.

For every string s such that $|s| \geq p$, we show that $s = xyz$ satisfying,

1. $|y| \neq 0$,
2. $|xy| \leq p$, and
3. For all i , $xy^i z \in L$.

Example of nonregular language that satisfies the pumping lemma

Let $\Sigma = \{\$\text{, } a, b\}$.

Consider the language $L = \{\$a^n b^n | n \geq 1\} \cup \{\$^k w | k \neq 1, w \in \{a, b\}^*\}$.

Let p be the pumping length.

For every string s such that $|s| \geq p$, we show that $s = xyz$ satisfying,

1. $|y| \neq 0$,
2. $|xy| \leq p$, and
3. For all i , $xy^i z \in L$.

There are two cases. The string s might be of the form $\$a^n b^n$ or of the form $\$^k w$ where $k \neq 1$.

For all cases, except when $k = 0$, consider $y = \$$. This satisfies all three conditions.

For the case when $s \in \{\$^k w | k = 0, w \in \Sigma^*\}$. Then s can be any string from $\{a, b\}^*$.

And, y can be any nonempty substring of s . This satisfies all three conditions.

Example of nonregular language that satisfies the pumping lemma

- $L = \{\$\overline{a^n}b^n | n \geq 1\} \cup \{\$^k w | k \neq 1, w \in \{a, b\}^*\}$.

Now, how is that we show L is nonregular.

This is through proof by contradiction.

Assume that L is regular.

We know $\$a^*b^*$ is regular. (why?)

Since regular languages are closed under intersection. Intersection of L and $\$a^*b^*$ must be regular.

But, Intersection of L and $\$a^*b^*$ is $\{\$a^n b^n | n \geq 0\}$, and this is nonregular (why?).

Hence, the contradiction.

Yet another language that is nonregular,
but for which PL is satisfied.

- This language is very similar to the previous one.
- $L = \{a^i b^j c^k | (i = 1) \Rightarrow (j = k)\}$.

An Important Other way of showing that a language is nonregular

- By using Myhill-Nerode Theorem
 - DFA or NFA for a regular language must have finite number of states.
 - If you show that infinite number of states are needed, then it is equivalent to showing that the language is nonregular.
 - Apart from this, Myhill-Nerode theorem has one important application, viz., minimization of a DFA.

Myhill-Nerode Theorem is much more than the Pumping Lemma

- Myhill-Nerode theorem can be used to show that a language is regular also. Of course it can be used to show that a language is nonregular.
 - This gives a necessary and sufficient condition for a language being regular.
- Note, the pumping lemma, on the otherhand can be used only to show that a language is nonregular.
 - Pumping lemma should not be used to show that a language is regular.

1.29 Use the pumping lemma to show that the following languages are not regular.

a. $A_1 = \{0^n 1^n 2^n \mid n \geq 0\}$

b. $A_2 = \{www \mid w \in \{a, b\}^*\}$

c. $A_3 = \{a^{2^n} \mid n \geq 0\}$ (Here, a^{2^n} means a string of 2^n a's.)

- **Reading Assignment – From Sipser’s book**

1.30 Describe the error in the following “proof” that $0^* 1^*$ is not a regular language. (An error must exist because $0^* 1^*$ *is* regular.) The proof is by contradiction. Assume that $0^* 1^*$ is regular. Let p be the pumping length for $0^* 1^*$ given by the pumping lemma. Choose s to be the string $0^p 1^p$. You know that s is a member of $0^* 1^*$, but Example 1.73 shows that s cannot be pumped. Thus you have a contradiction. So $0^* 1^*$ is not regular.

CPS 220 – Theory of Computation

Non-regular Languages

Warm up Problem

Problem #1.48 (p.90)

Let $\Sigma = \{0,1\}$ and let

$D = \{w \mid w \text{ contains an equal number of occurrences of the substrings } 01 \text{ and } 10\}$.

Thus $101 \in D$ because 101 contains a single 01 and a single 10 , but $1010 \notin D$ because 1010 contains two 10 s and only one 01 . Show that D is a regular language.

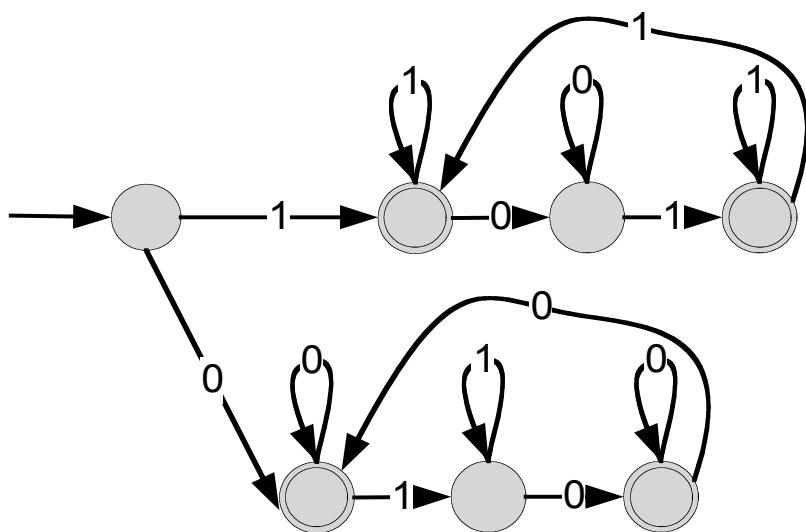
Solution:

This language is regular because it can be described by a regular expression and a FA (NFA):

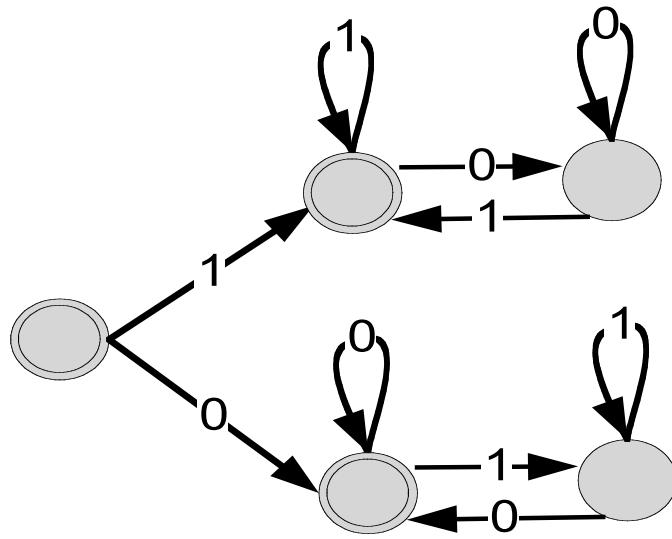
Regular Expression

$$(1^+ 0^* 1^*)^* + (0^+ 1^* 0^*)^*$$

NFA



DFA



Proving that a language is not regular—the Pumping Lemma

Consider the language $B = \{ 0^n 1^n \mid n \geq 0 \}$. Is B regular?

If B is regular, then there is a DFA M recognizing B . That means, M accepts the string $0^{2003} 1^{2003}$,

but rejects the string $0^{2003} 1^{1999}$. How can M achieve that? As it reads the input, it has to remember

how many 0s it encountered so far. Then, when it starts reading 1s, it has to count the 1s and match

them with the number of 0s. However, a DFA by definition is finite, i.e., it has limited memory, which

is just the current state in which it is. To count it would require enough bits of memory to store a number...

How can we prove that a language is not regular? We will now demonstrate a method of doing that.

+++++
+++++

Theorem. The Pumping lemma. If A is a regular language, then there is a number p (the pumping length)

where, if s is any string in A of length at least p , then s may be divided into three pieces, $s = xyz$, satisfying

the following conditions:

1. for each $i \geq 0$, $xy^i z \in A$
2. $|y| > 0$
3. $|xy| \leq p$

Note: either x or z may be ϵ

+++++
+++++

So if s is long enough, there is a nonempty string y within s , which can be “pumped”.

=====

In other words: If a language A is accepted by a DFA M with q states, then every string s in A

with $|s| \geq q$ can be written as $s = xyz$ such that $y \neq \epsilon$ and $xy^* z \subseteq A$

=====

Proof. If A is regular, then there is a DFA M that accepts A .

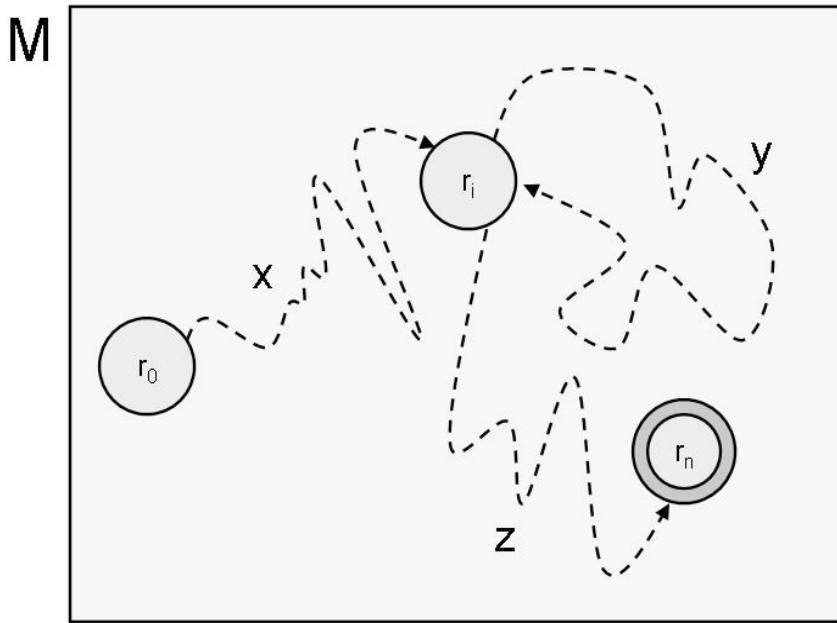
Say that M has p states.

Let s be a string of length $n \geq p$. Let $r_1 \dots r_{n+1}$ be the sequence of states of M , when processing s .

Since $n+1 > p$, at least one of M 's states appears twice: $r_i = r_j$ for $j \neq i$.

Let $x = s_1 \dots s_i$, $y = s_{i+1} \dots s_j$, $z = s_{j+1} \dots s_n$. Then, $xyz = s_1 \dots s_n = s$, and $|y| > 0$.

Here is why $xy^i z$ is accepted by M :



How can we maintain this condition - $|xy| \leq p$? We can make sure that this is true, by selecting the *smallest* i and j such that $r_i = r_j$.

The first $p+1$ states in the sequence must contain a repetition - therefore $|xy| \leq p$.

Pigeonhole principle - if p pigeons are placed into fewer than p holes, some hole has to have more than one pigeon in it.

Example of using the Pumping Lemma

Theorem. $B = \{ 0^n 1^n \mid n \geq 0 \}$ is not regular.

Proof. Proof by contradiction.

Assume that language B is regular. [If B is regular, then there exists a constant p (the pumping length) such that the conditions of the pumping lemma hold.] Let p be the pumping length and let's choose a string s that fits the conditions for our pumping lemma. Let $s = 0^p 1^p$. Because $s \in B$ and $|s| \geq p$ - then s can be broken into xyz where for any $i \geq 0$ the string $xy^i z \in B$.

Must consider 3 cases:

Case 1:

The substring y consists of only 0s. In this case the string $xyyz$ has more 0s than 1s and therefore

$xxyz \notin B$. This case is a contradiction.

Case 2:

The string y consists of only 1s. For the same basic reason, this case is a contradiction.

Case 3:

The string y consists of both 0s and 1s. In this case the string xyyz may have the same number of 1s and 0s - however it violates the basic structure of the language - where all 0 come before all 1s. This case is a contradiction

Thus a contradiction is unavoidable.

Definition. The complement of a language A , $\overline{A} = \Sigma^* - A$, is all strings except those in A .

Theorem. Regular languages are closed under complementation.

Proof. Let A be a regular language. We will show that \overline{A} is regular.

Since A is regular, a DFA M accepts it.

By turning all the accept states of M into non-accept, and all non-accept states into accept, every input in A ends up in a non-accept state, and every input not in A ends up in an accept state. Therefore, the modified M machine accepts \overline{A} .

Theorem. Regular languages are closed under intersection, \cap .

Proof. Recall that if A and B are two sets, $A \cap B$ is the set of all the common elements.

A simple logic fact that you can prove as an exercise is that:

$$A \cap B = \overline{\overline{A} \cup \overline{B}}$$

Then, the result follows by closure of regular languages under complement and union.

Theorem. $A = \{ w \mid w \text{ has an equal number of 0s and 1s} \}$ is not regular.

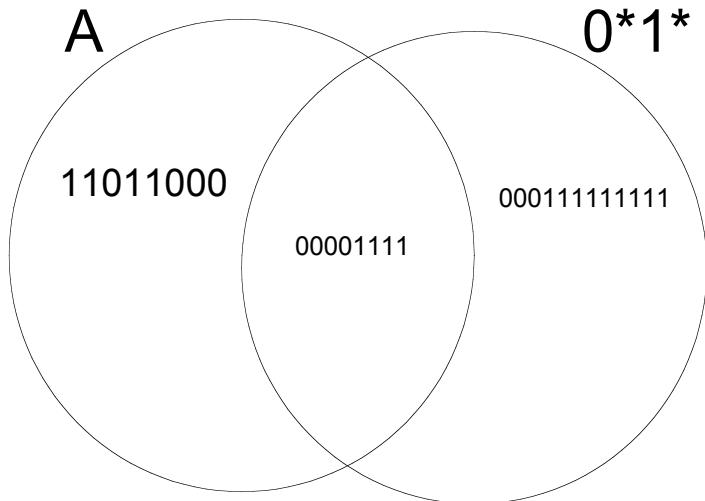
Proof. Assume that A is regular, in order to arrive at a contradiction. (Very cool proof)

We know that 0^*1^* is regular.

Therefore $A \cap (0^*1^*)$ is regular, by closure of regular languages under \cap .

However, $A \cap (0^*1^*) = \{ 0^n1^n \mid n \geq 0 \}$, which we proved not to be regular.

Therefore A cannot be regular.



Another pumping lemma proof:

Theorem. $A = \{ 0^p \mid p \text{ is a prime} \}$ is not a regular language.

Proof. Assume that A is regular, in order to arrive at a contradiction.

Then A is accepted by a DFA M . Let s be the number of states in M . Consider a prime number $p > s$.

Note that $0^p \in A$ and $|0^p|=p > s$. Therefore, by the pumping lemma, 0^p can be written as $0^p = xyz$ such that $|y| > 0$ and $xy^i z \in A$

Let $i = |x|+|z|$ and $j = |y|$. Then, the condition $xy^*z \in A$ means that, for any $k \geq 0$, $i+kj$ is a prime.

In particular, when $k = 0$ it means that i is a prime. So $i \geq 2$. When $k = i$, this means that $i(1+j)$ is a prime.

However, since $|y| > 0$ (not the empty string), we have $j = |y| \geq 1$ and so $i(1+j)$ is not prime. This is a contradiction.

References:

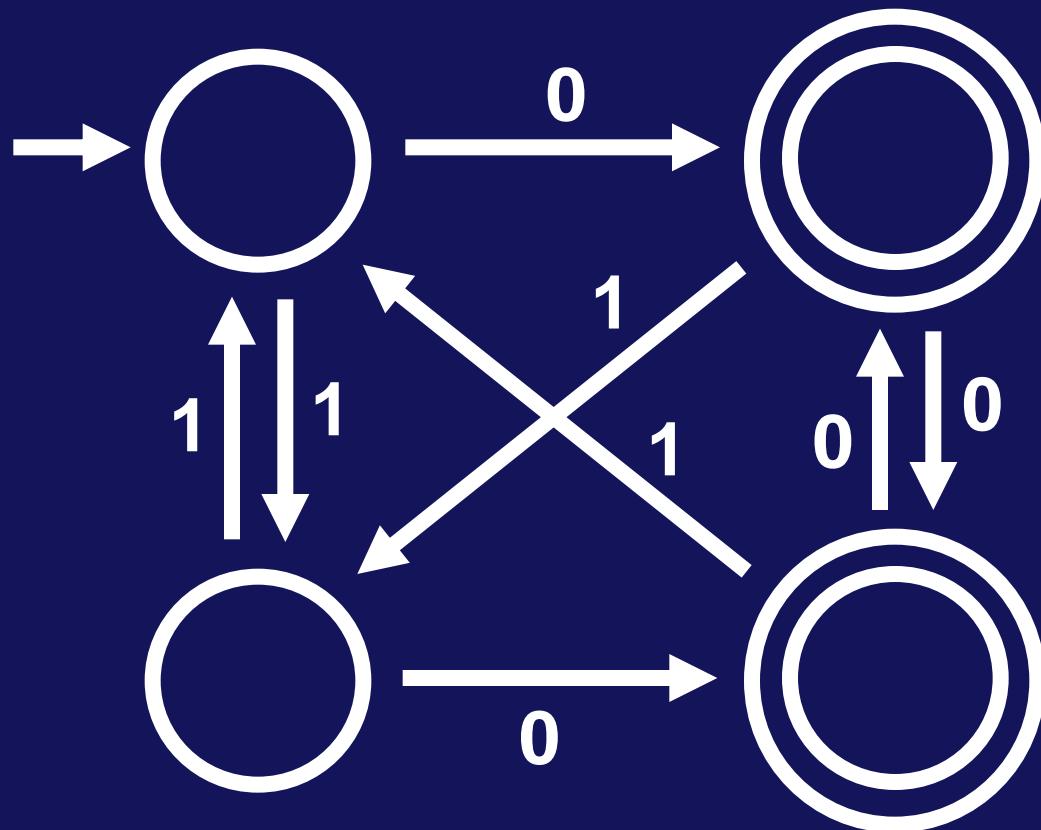
Introduction to the Theory of Computation (2nd ed.) Michael Sipser

Problem Solving in Automata, Languages, and Complexity Ding-Zhu Du and Ker-I Ko

MINIMIZING DFAs

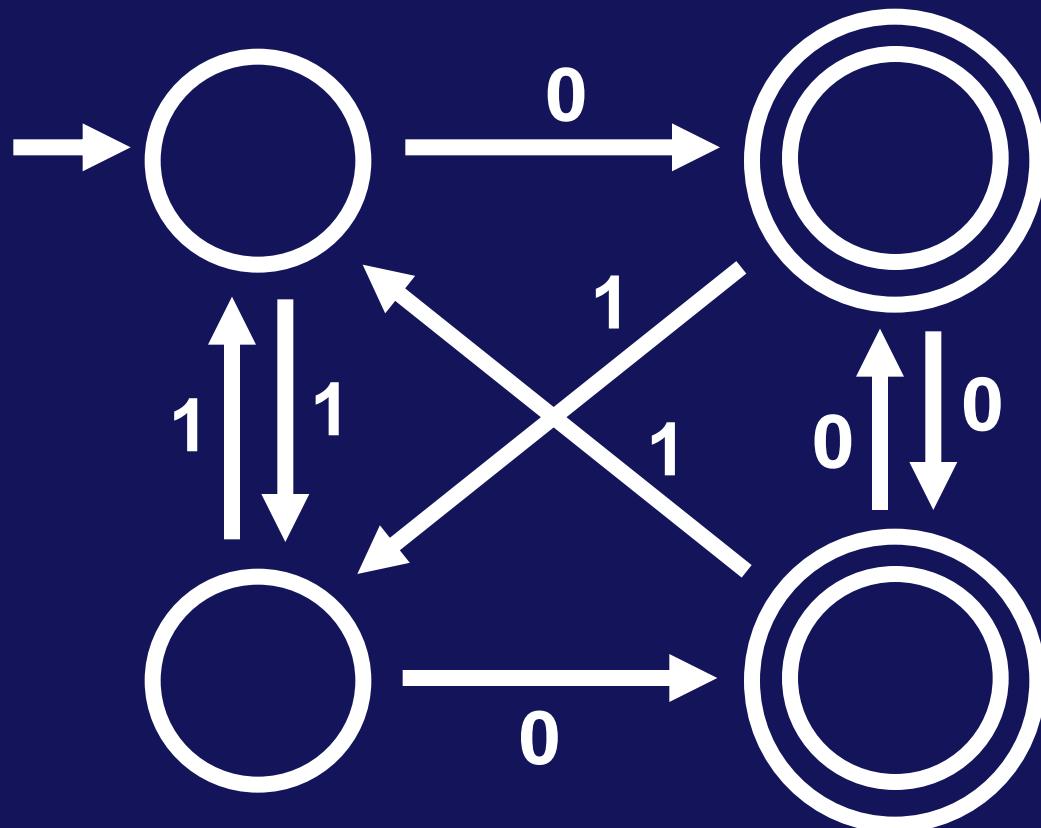
To have minimum number of states

IS THIS MINIMAL?

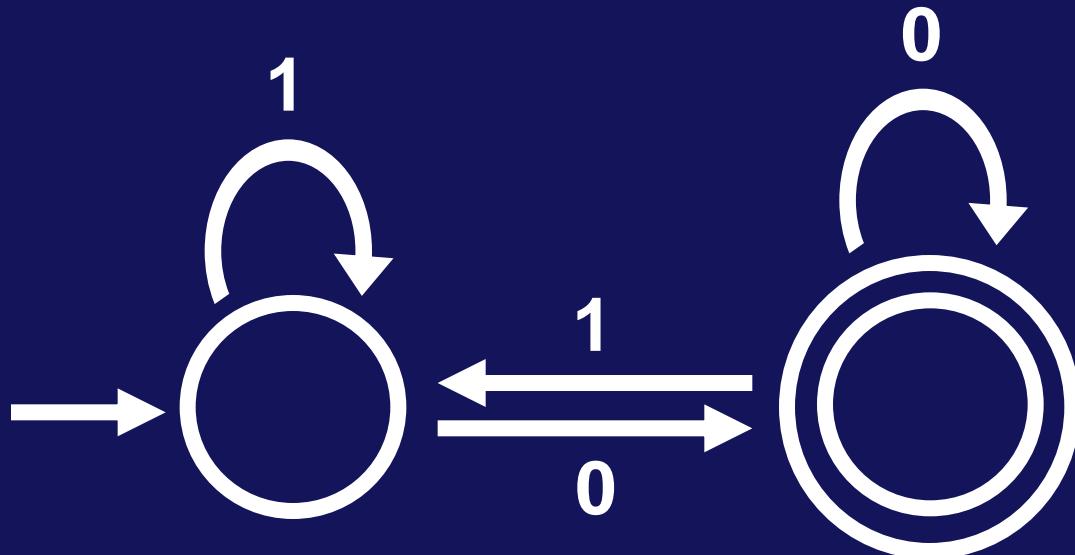


IS THIS MINIMAL?

NO



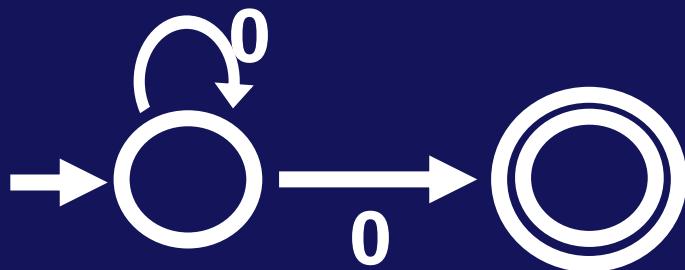
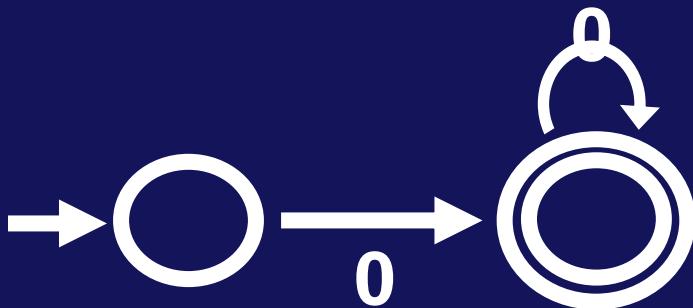
IS THIS MINIMAL?



THEOREM

For every regular language L , there exists
a **unique** (up to re-labeling of the states)
minimal DFA M such that $L = L(M)$

NOT TRUE FOR NFAs



Because of this, minimization of NFA is complicated and is out of scope of current ToC course.

EXTENDING δ

Given DFA $M = (Q, \Sigma, \delta, q_0, F)$ extend δ

to $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ as follows:

$$\hat{\delta}(q, \epsilon) = q$$

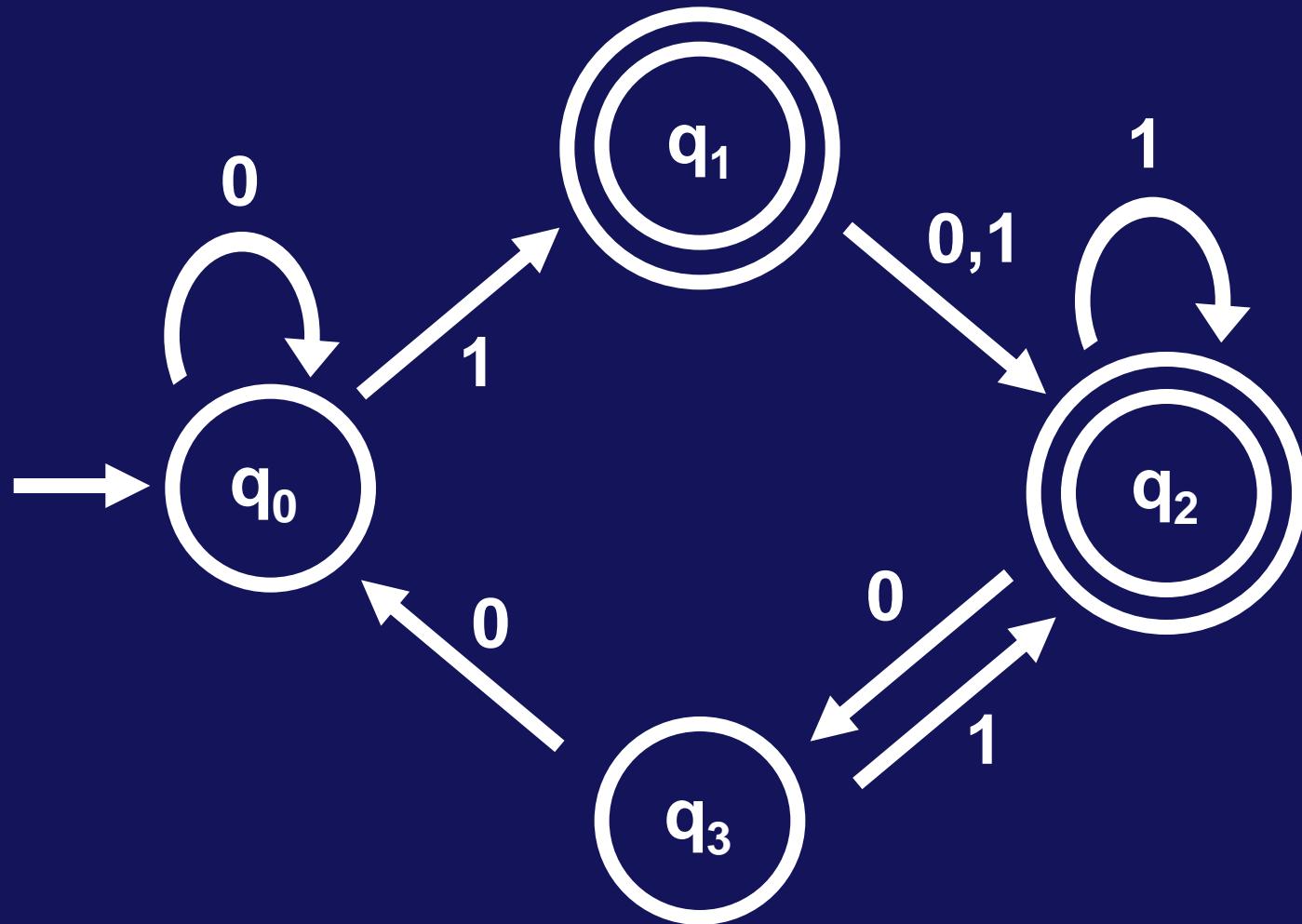
$$\hat{\delta}(q, a) = \delta(q, a) \text{ where } a \in \Sigma$$

$$\hat{\delta}(q, w_1 \dots w_{k+1}) = \delta(\hat{\delta}(q, w_1 \dots w_k), w_{k+1})$$

Note: in $\delta(q, a)$, a is a string. Context should clear this.

A string $w \in \Sigma^*$ **distinguishes states** q_1 from q_2 if

$$\widehat{\delta}(q_1, w) \in F \Leftrightarrow \widehat{\delta}(q_2, w) \notin F$$



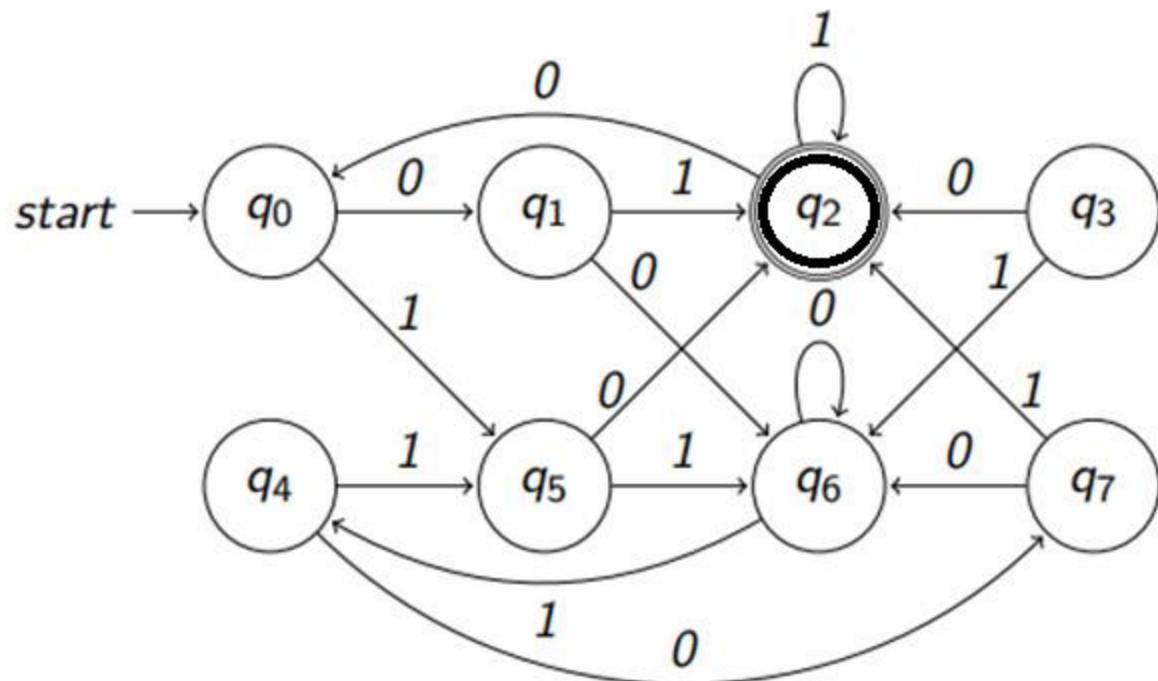
ϵ distinguishes accept from non-accept states

Let $M = (Q, \Sigma, \delta, q_0, F)$ and let $p, q, r \in Q$

Definition :

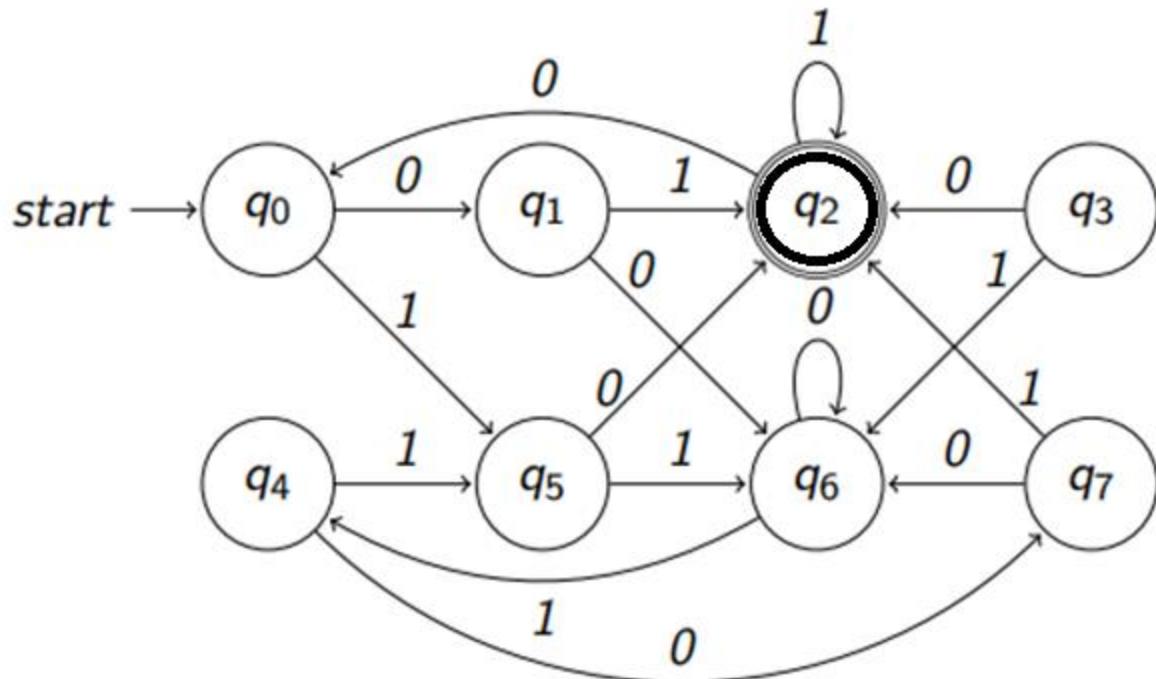
- (1) p is **equivalent** to q iff there is *no* $w \in \Sigma^*$ that distinguishes p and q ,
- (2) Otherwise p is **not equivalent** to q . In this case, we say p and q are **distinguishable**.

Example: distinguishable states



- ▶ ϵ distinguishes q_2 and q_6 .
- ▶ 01 distinguishes q_0 and q_6 .

Example: distinguishable states



- ▶ ϵ distinguishes q_2 and q_6 .
- ▶ 01 distinguishes q_0 and q_6 .

Exercise

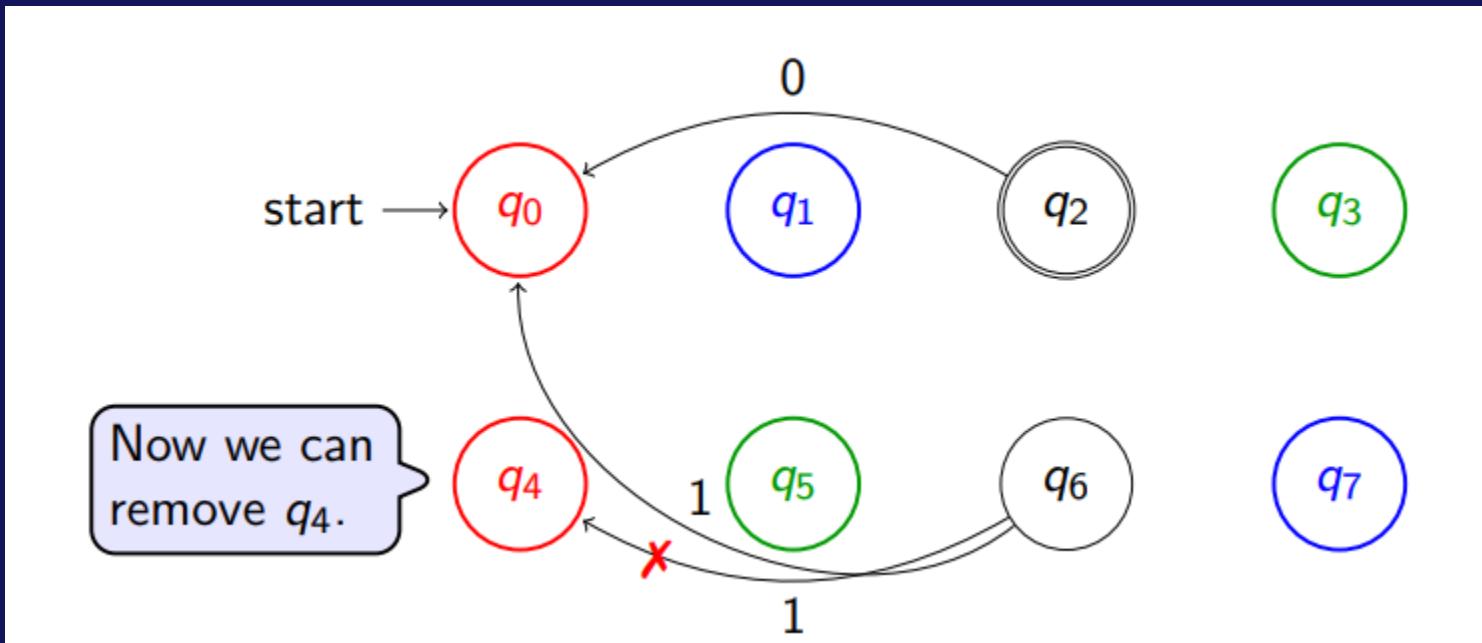
Give strings that distinguishes the following pair of states.

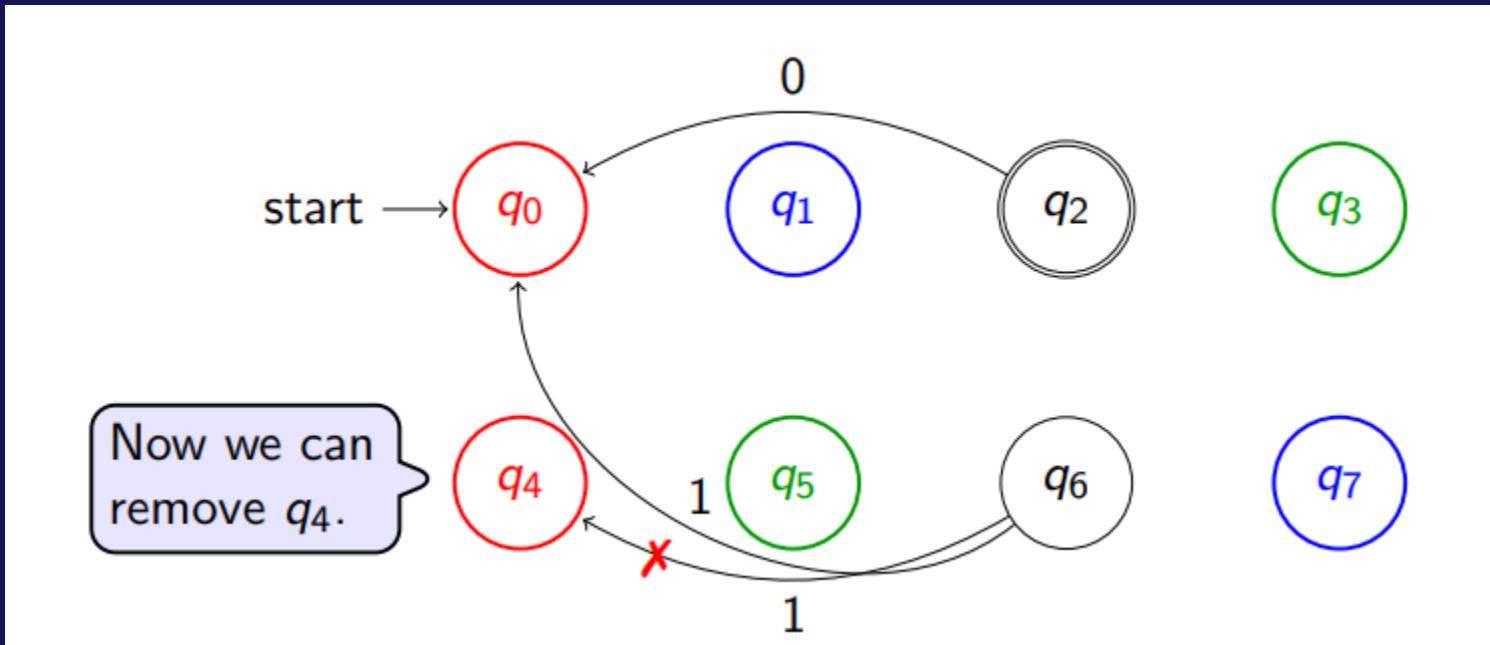
- ▶ q_1 and q_5
- ▶ q_2 and q_7
- ▶ q_4 and q_3
- ▶ q_1 and q_7

DFA Minimization, intuition

- We can remove unreachable states (**why** and **how to find unreachable states?**)
- If states q_0 and q_4 are equivalent, then, we can move all incoming transitions (arrows) from q_4 to q_0
- Because of this q_4 becomes unreachable, hence can be removed.

Let q_0 and q_4 are equivalent





- But, how is that you know q_0 and q_4 are equivalent?

Let $M = (Q, \Sigma, \delta, q_0, F)$ and let $p, q, r \in Q$

Define the relation “ \sim ”:

$p \sim q$ iff p is equivalent to q

$p \not\sim q$ iff p is distinguishable from q

Proposition: “ \sim ” is an equivalence relation

$p \sim p$ (reflexive)

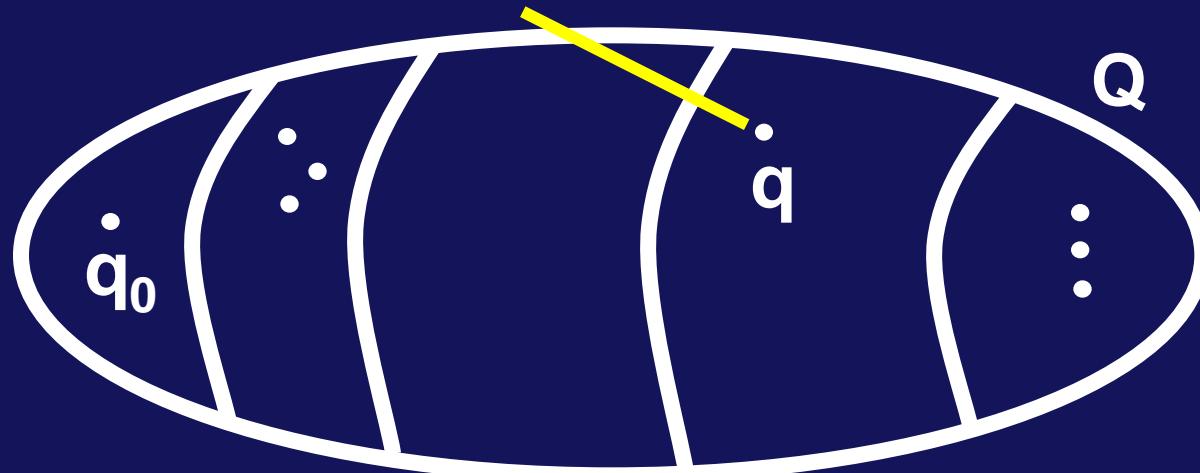
$p \sim q \Rightarrow q \sim p$ (symmetric)

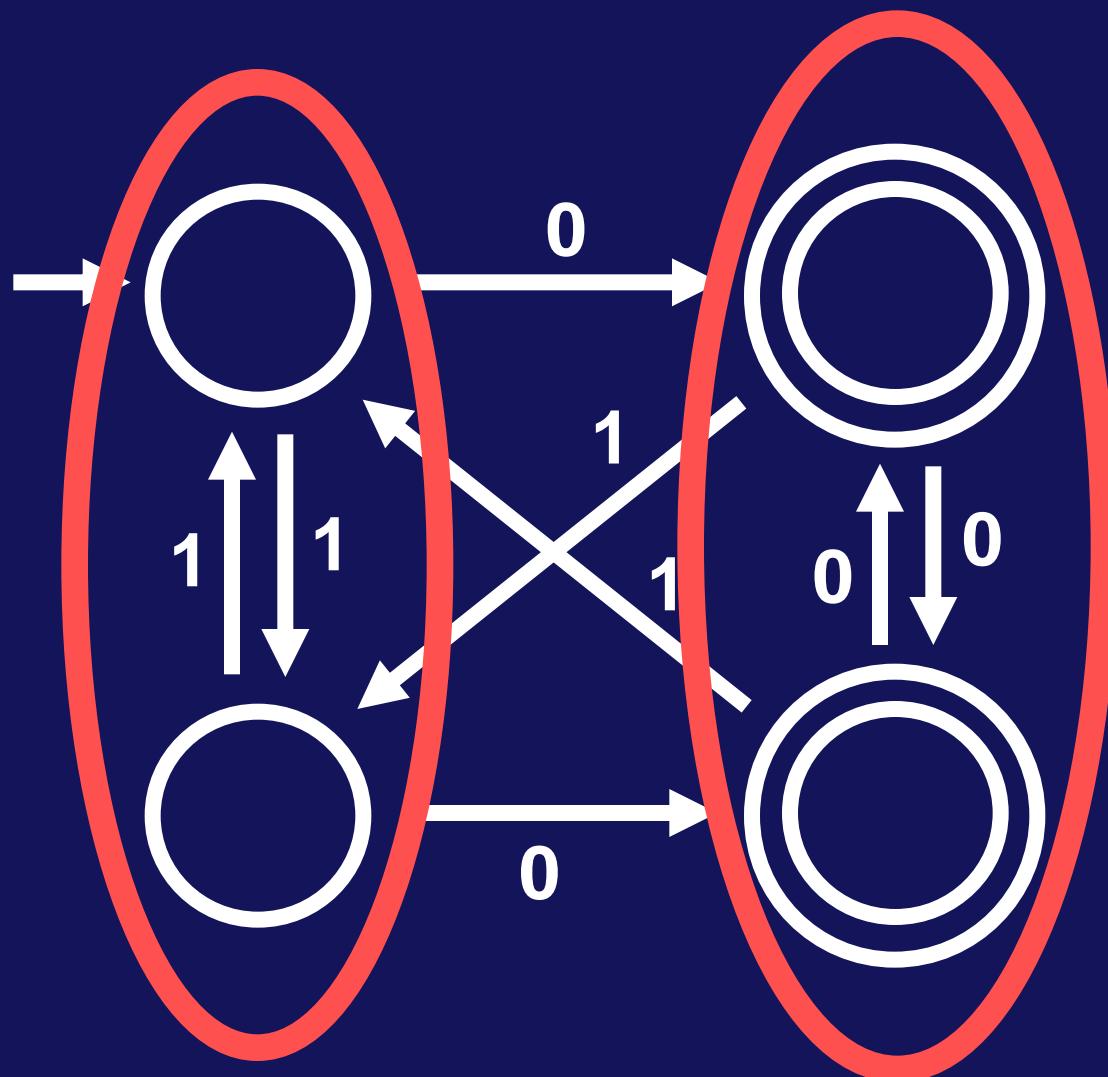
$p \sim q$ and $q \sim r \Rightarrow p \sim r$ (transitive)

Let $M = (Q, \Sigma, \delta, q_0, F)$ and let $p, q, r \in Q$

Proposition: “ \sim ” is an **equivalence relation**
so “ \sim ” partitions the set of states of M into
disjoint equivalence classes

$$[q] = \{ p \mid p \sim q \}$$





Algorithm MINIMIZE(DFA M)

Input: DFA M

Output: DFA M_{MIN} such that:

$$M \equiv M_{MIN}$$

M_{MIN} has no inaccessible states

M_{MIN} is irreducible

||

states of M_{MIN} are pairwise distinguishable

Theorem: M_{MIN} is the unique minimum

Algorithm MINIMIZE(DFA M)

(1) Remove all inaccessible states from M

(2) Apply Table-Filling algorithm to get
 $E_M = \{ [q] \mid q \text{ is an accessible state of } M \}$

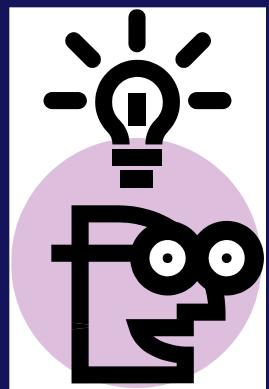
$$M_{MIN} = (Q_{MIN}, \Sigma, \delta_{MIN}, q_{0\ MIN}, F_{MIN})$$

$$Q_{MIN} = E_M, \quad q_{0\ MIN} = [q_0], \quad F_{MIN} = \{ [q] \mid q \in F \}$$

$$\delta_{MIN}([q], a) = [\delta(q, a)]$$

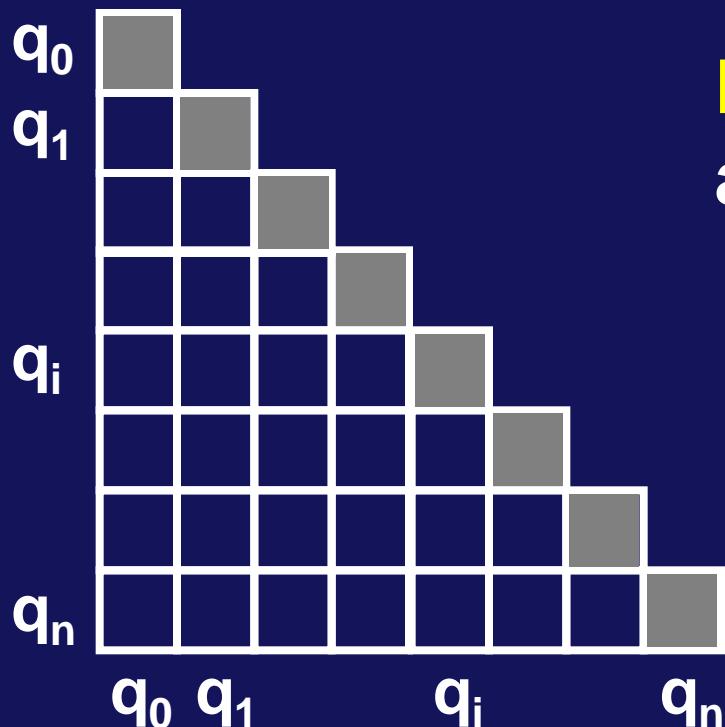
TABLE-FILLING ALGORITHM

IDEA!



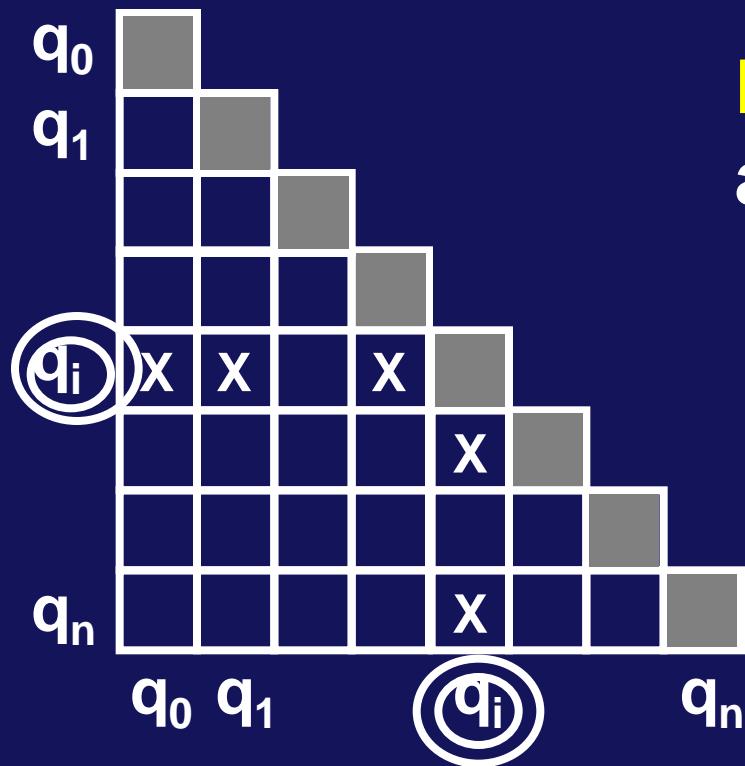
- Make best effort to find pairs of states that are distinguishable.
- Pairs leftover will help us.

TABLE-FILLING ALGORITHM



Base Case: p accepts
and q rejects $\Rightarrow p \neq q$

TABLE-FILLING ALGORITHM

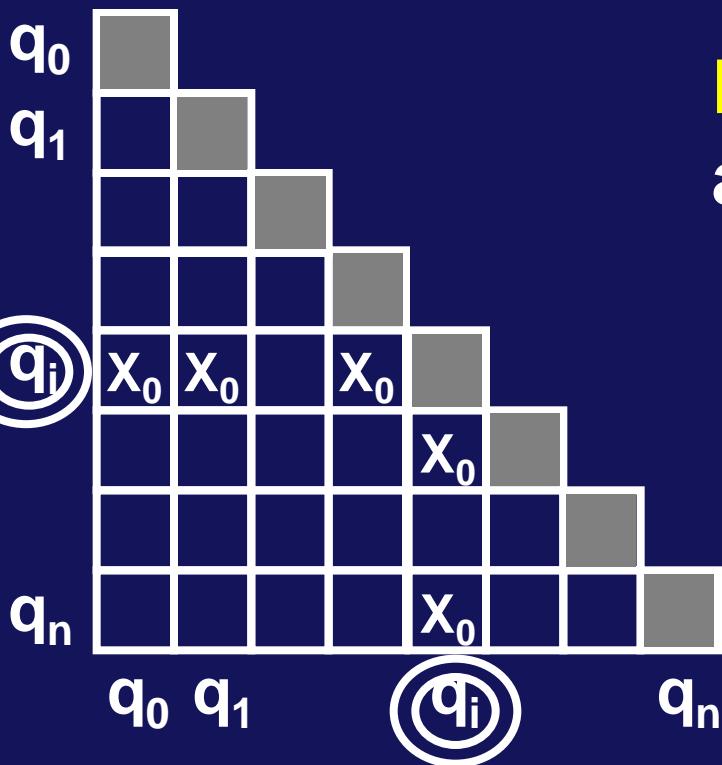


Base Case: p accepts
and q rejects $\Rightarrow p \neq q$

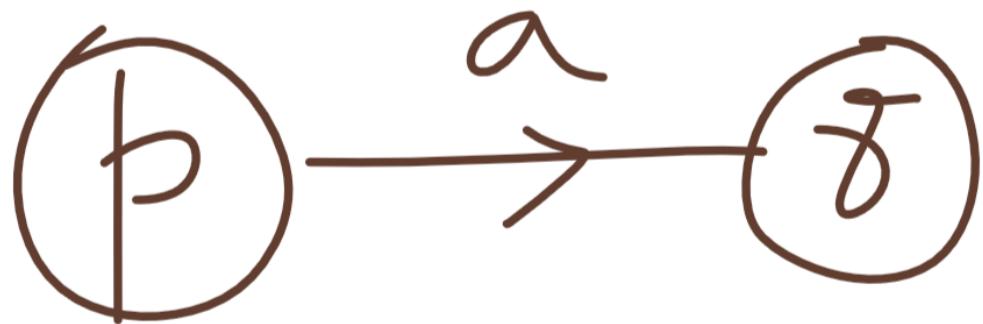
TABLE-FILLING ALGORITHM

For base case, we put X_0 in the corresponding cell.

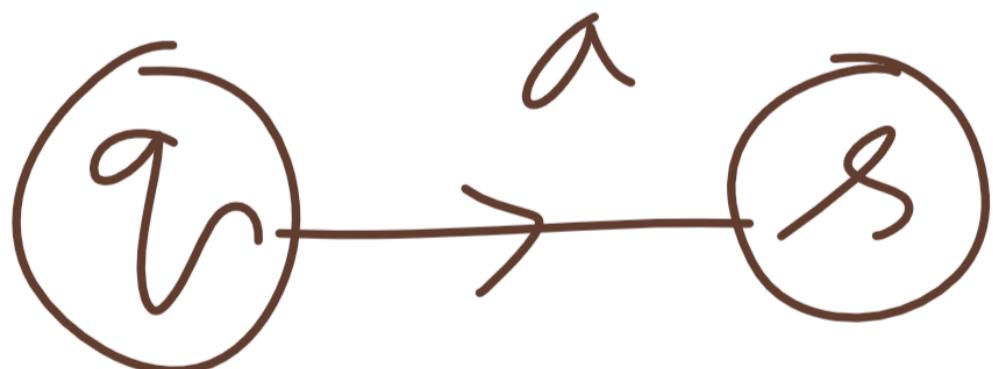
This states, that the two states are not equivalent.



**Base Case: p accepts
and q rejects $\Rightarrow p \neq q$**



$p \vdash q$



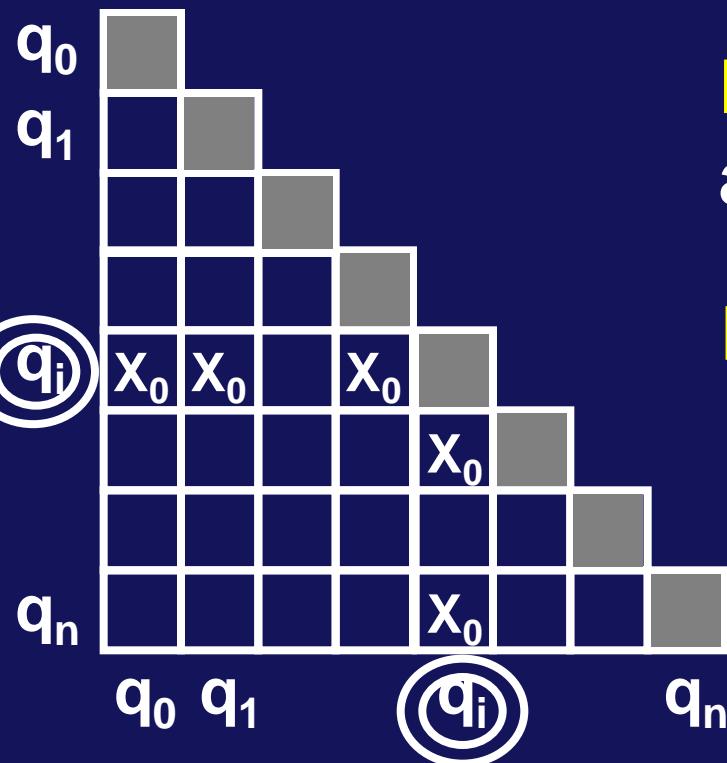
\Rightarrow

$p \vdash r$

TABLE-FILLING ALGORITHM

For base case, we put X_0 in the corresponding cell.

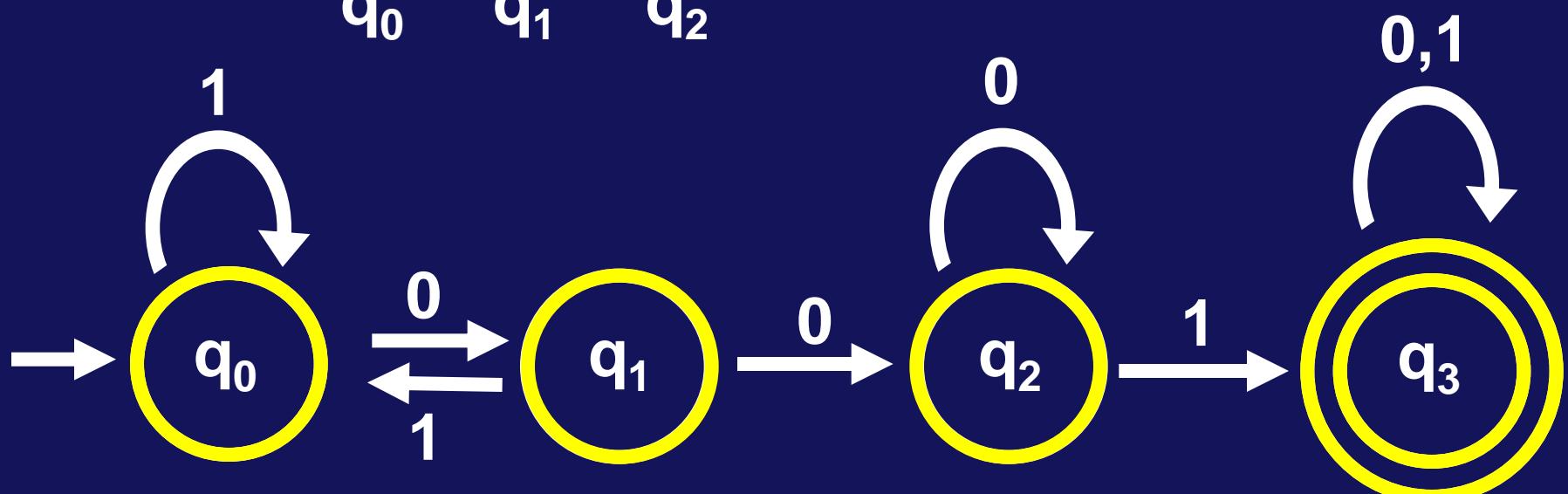
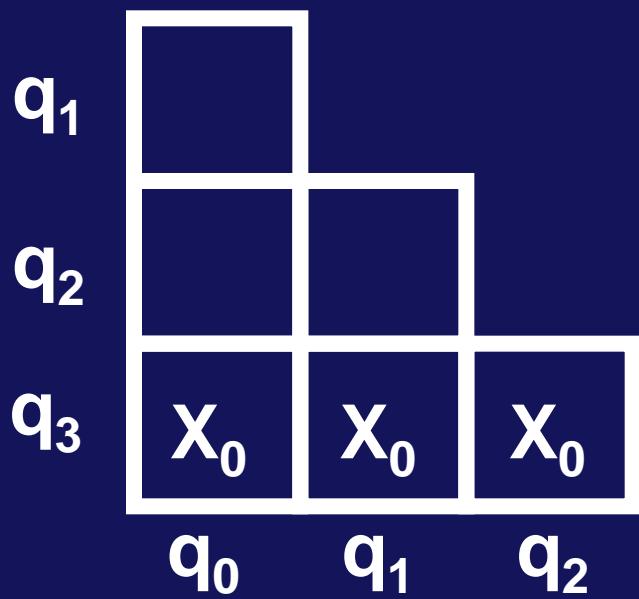
This states, that the two states are not equivalent.



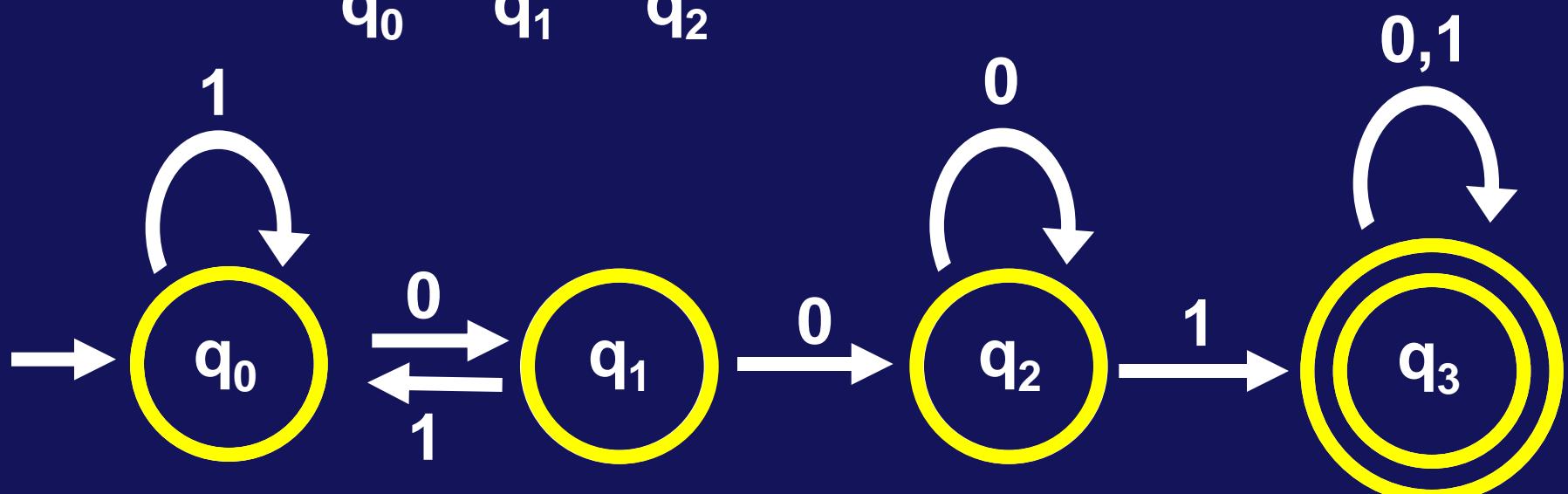
Base Case: p accepts
and q rejects $\Rightarrow p \neq q$

Recursion:

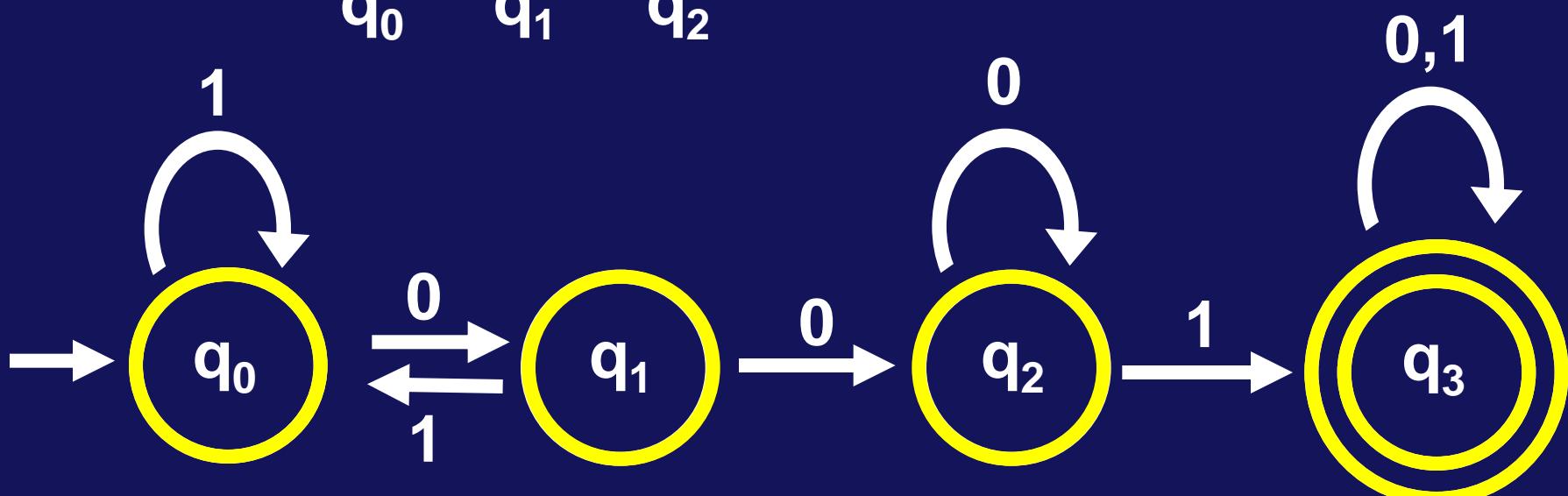
$$\begin{array}{c} p \xrightarrow{a} r \\ q \xrightarrow{a} s \end{array} \Rightarrow p \neq q$$



			q_1
	X_1	X_1	q_2
	X_0	X_0	q_3
	q_0	q_1	q_2

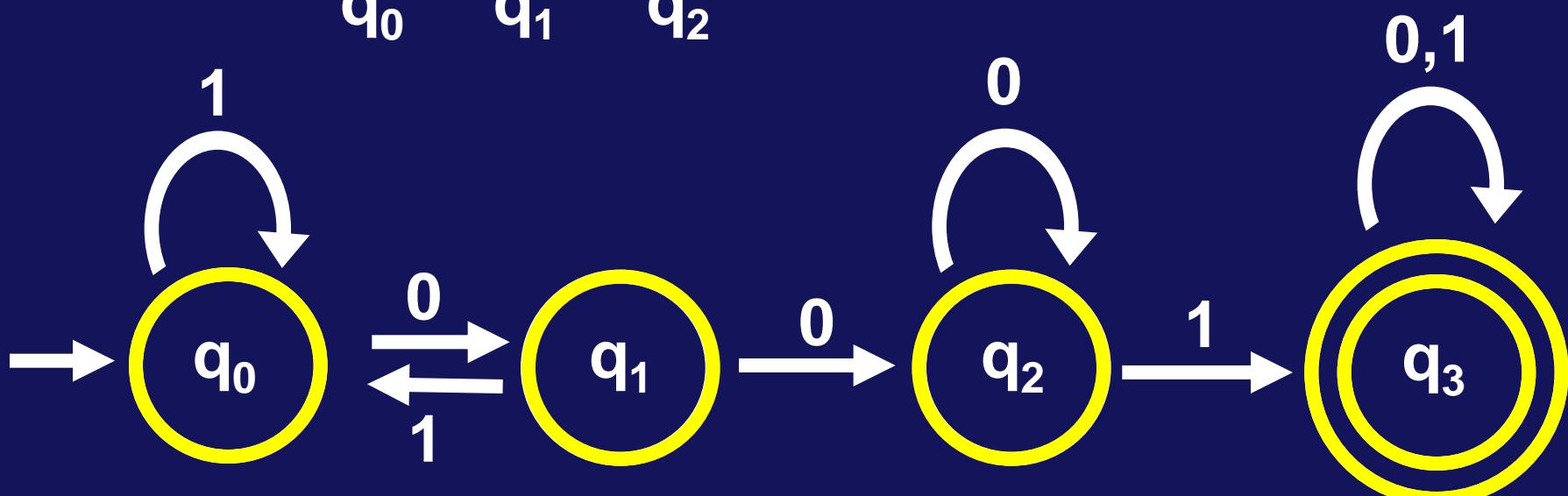


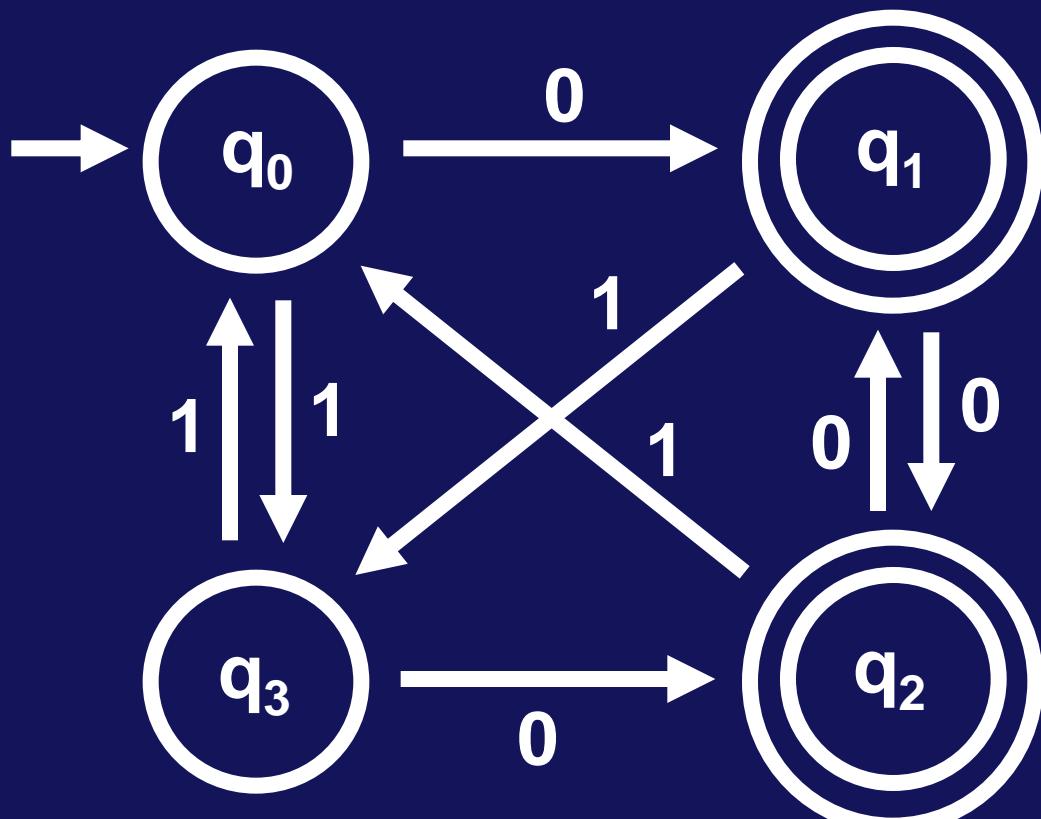
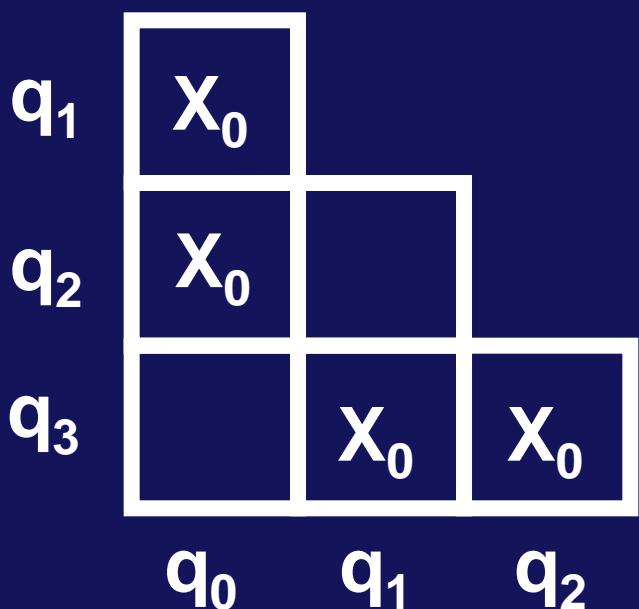
	X_2	
q_1		
q_2	X_1	X_1
q_3	X_0	X_0
	X_0	

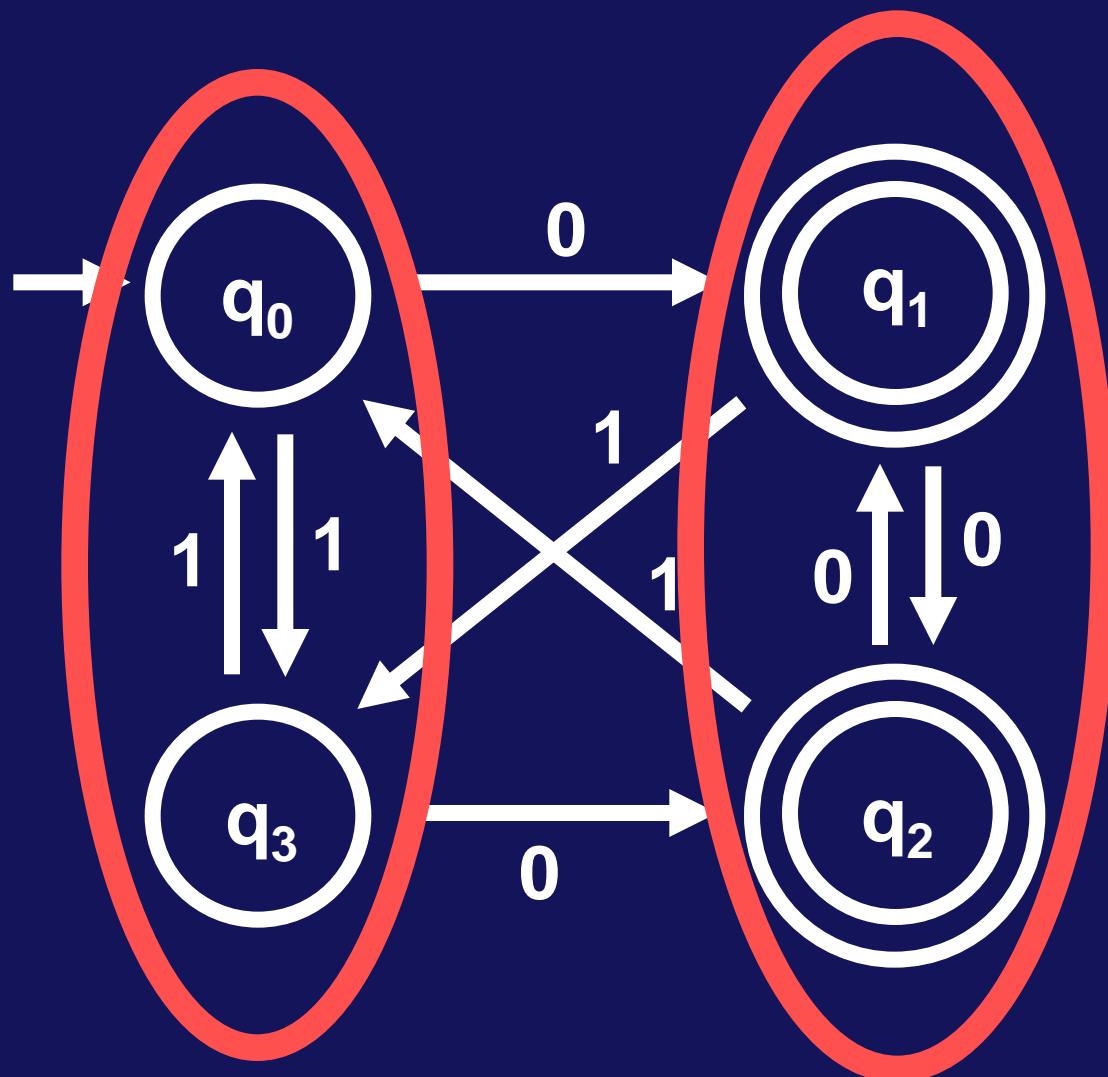


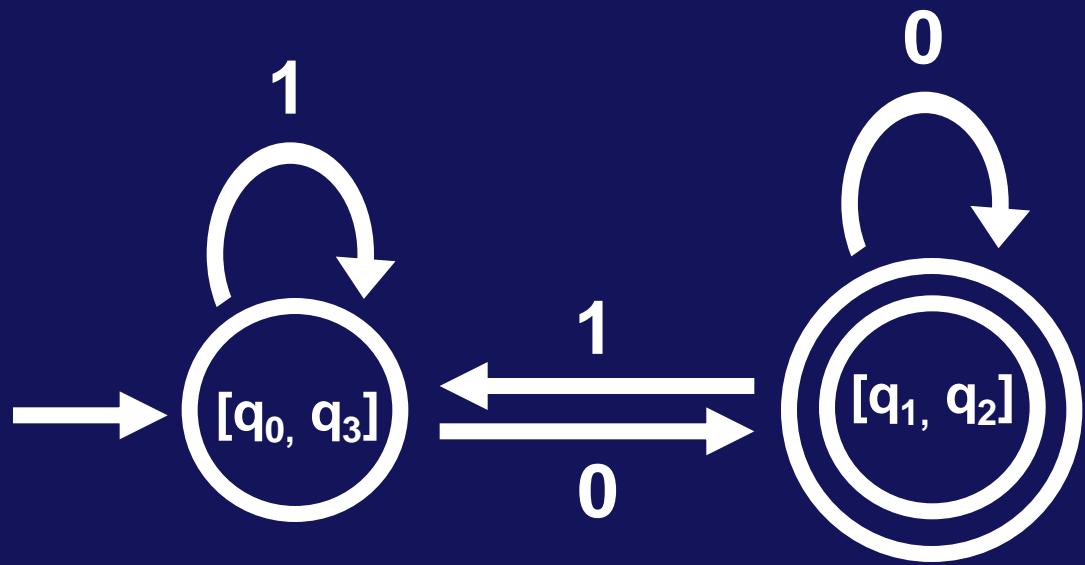
Have you noticed the reason for putting symbols X_1, X_2

	X_2	
q_1		
q_2	X_1	X_1
q_3	X_0	X_0
	q_0	q_1

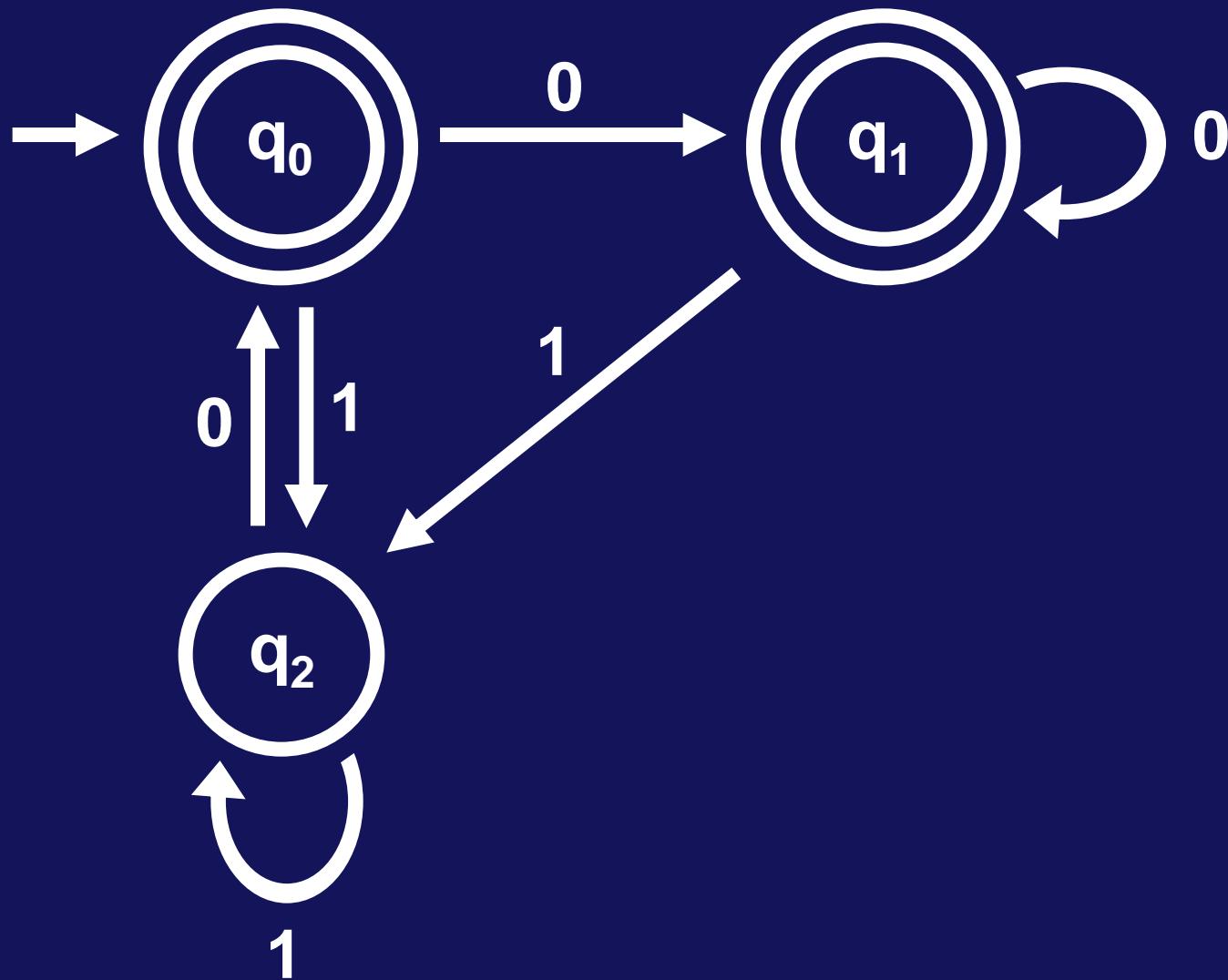




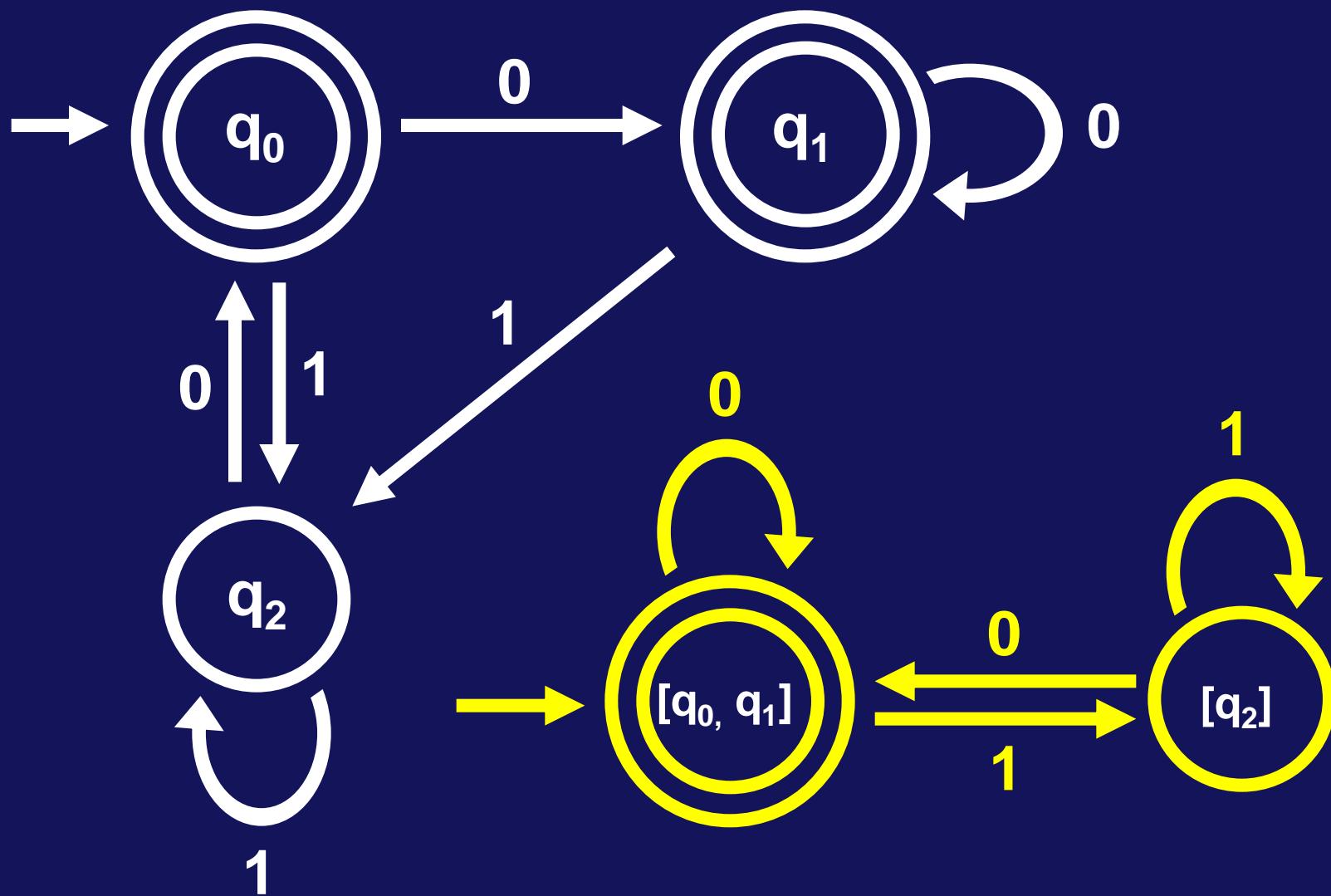


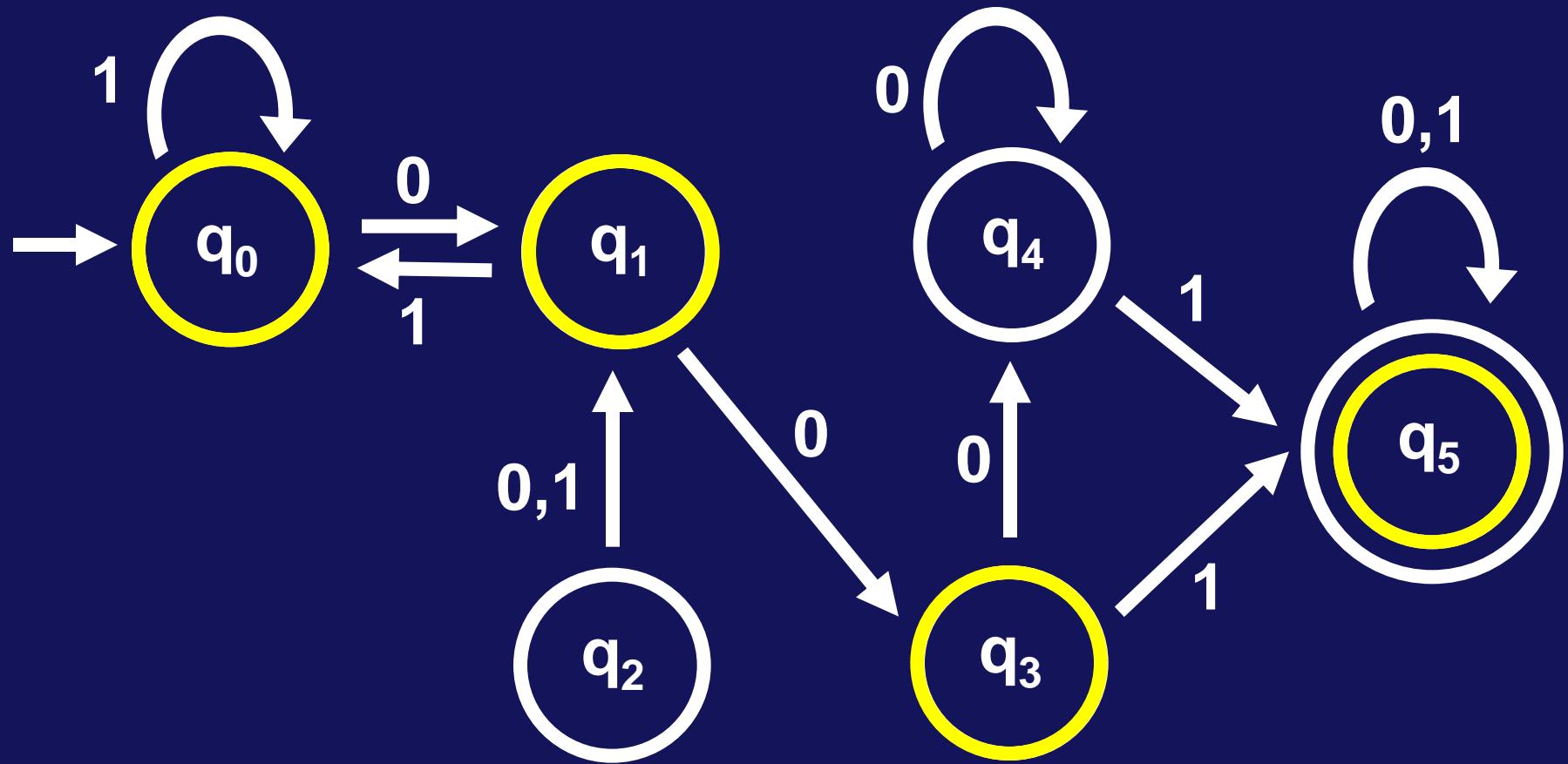


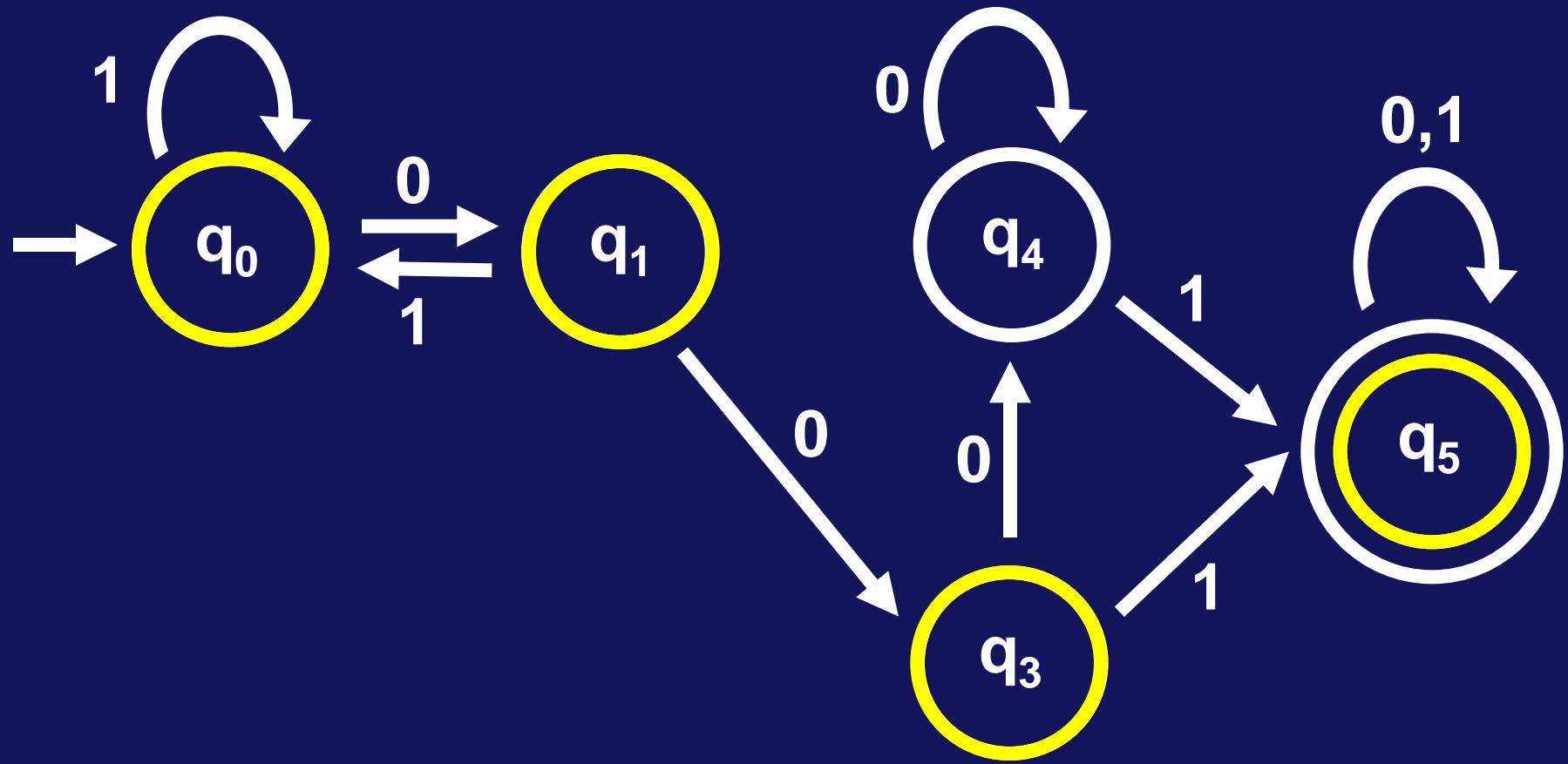
MINIMIZE

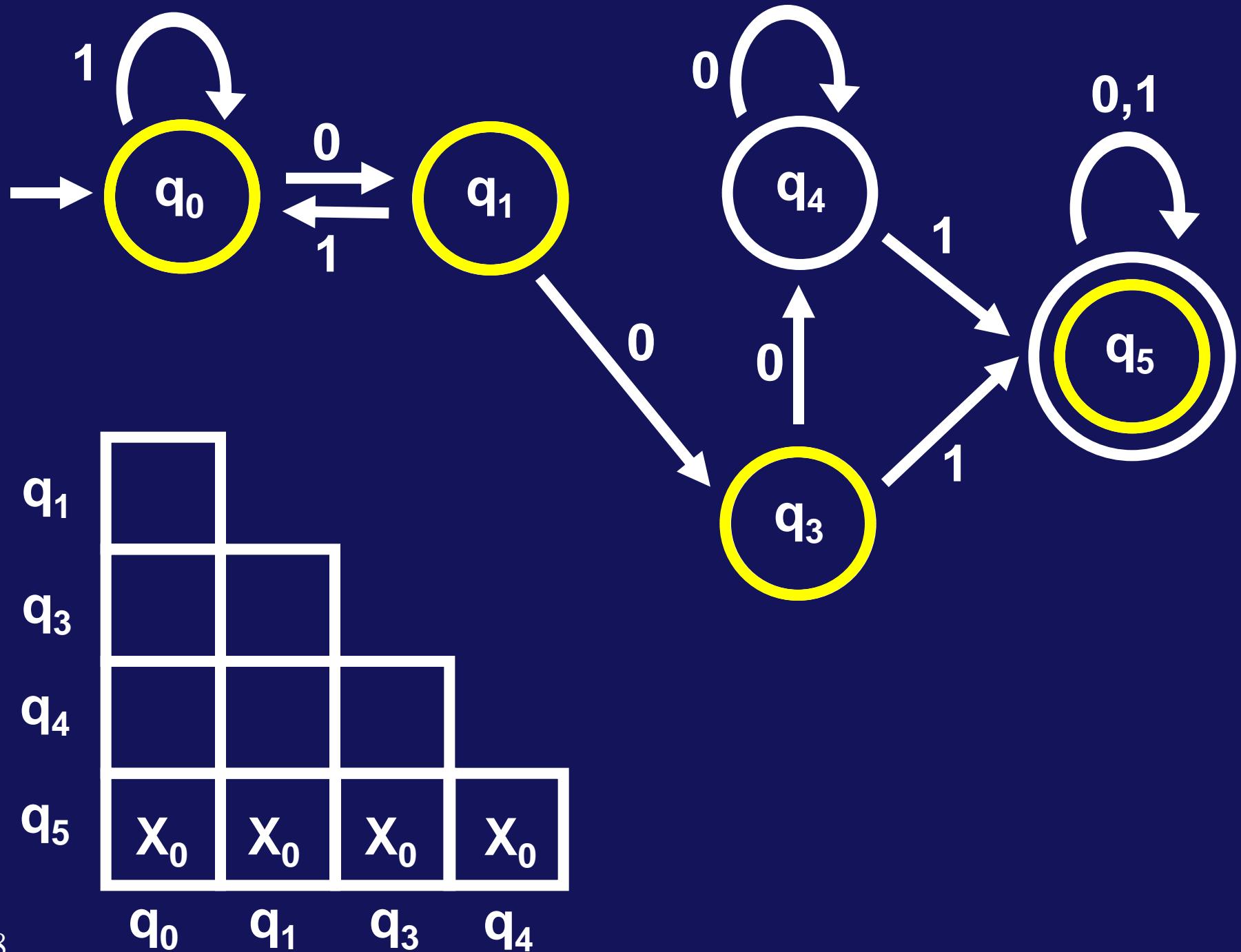


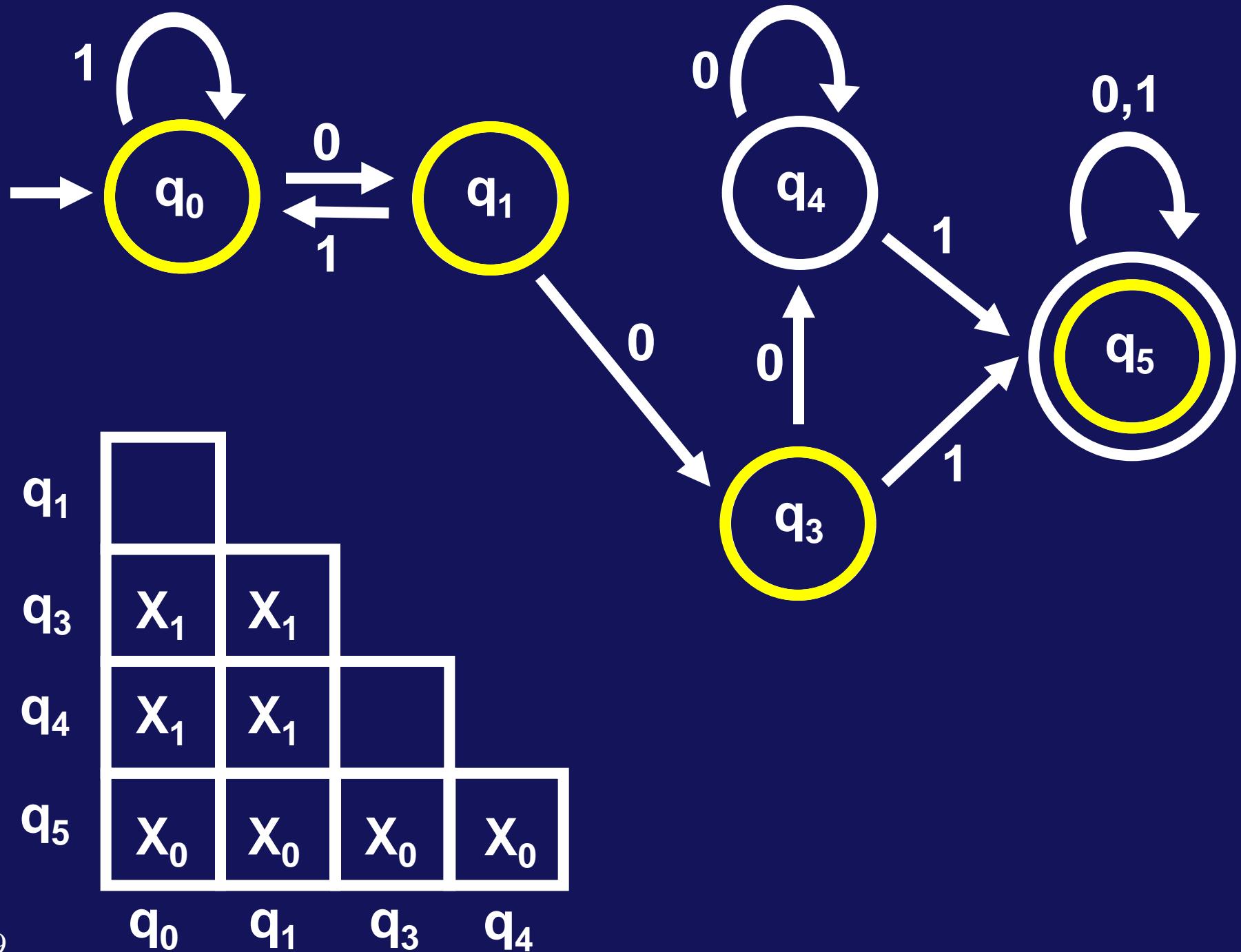
MINIMIZE

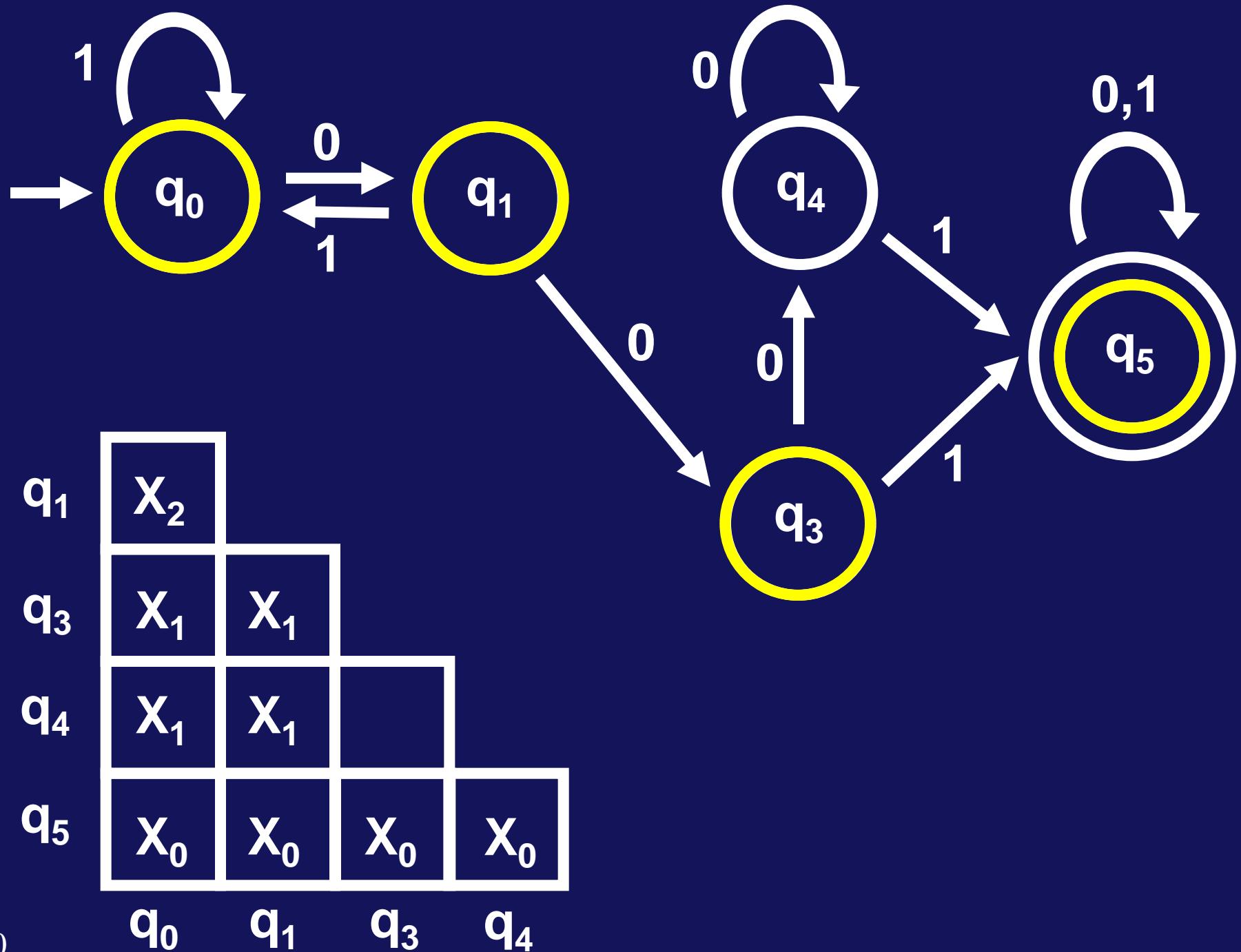


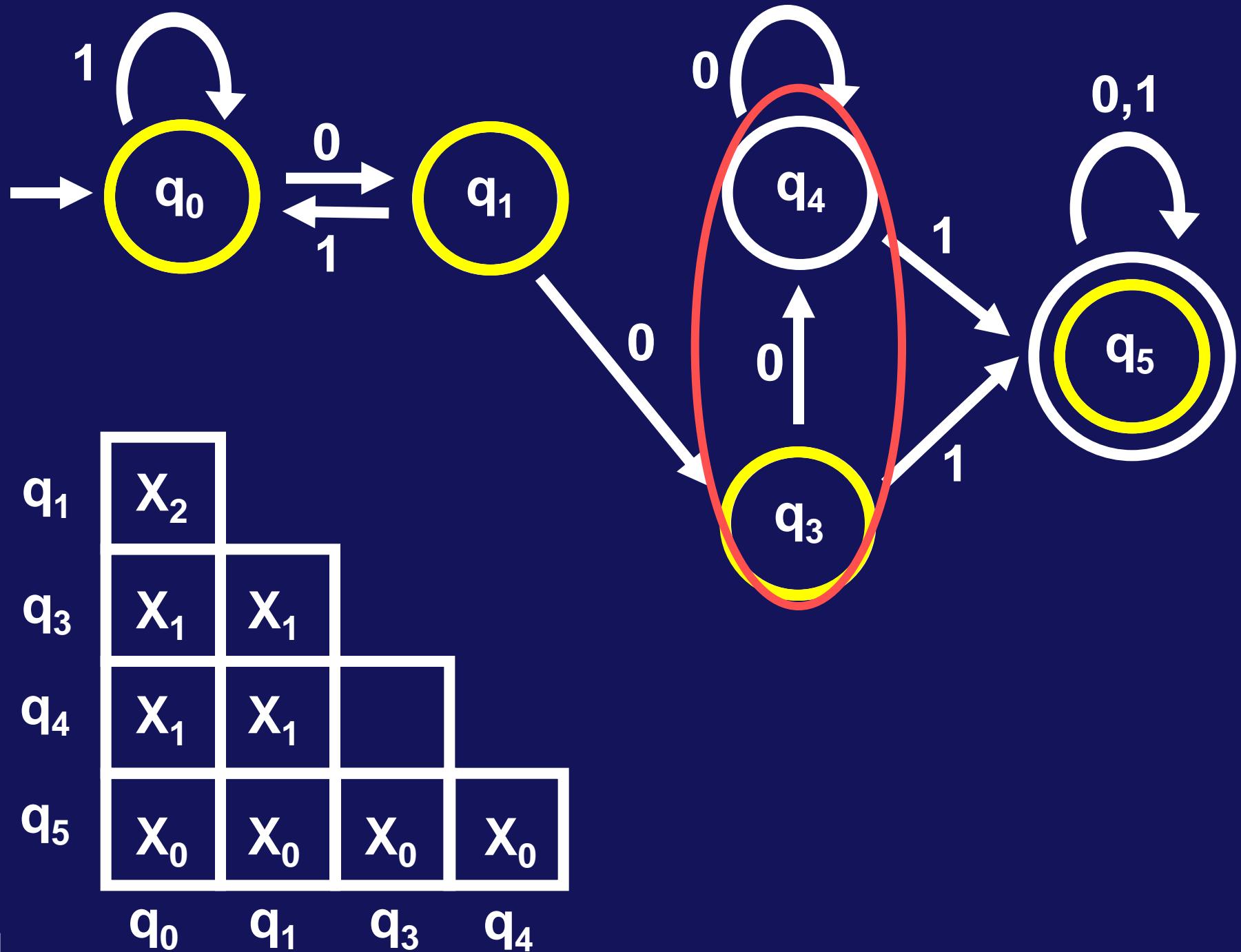


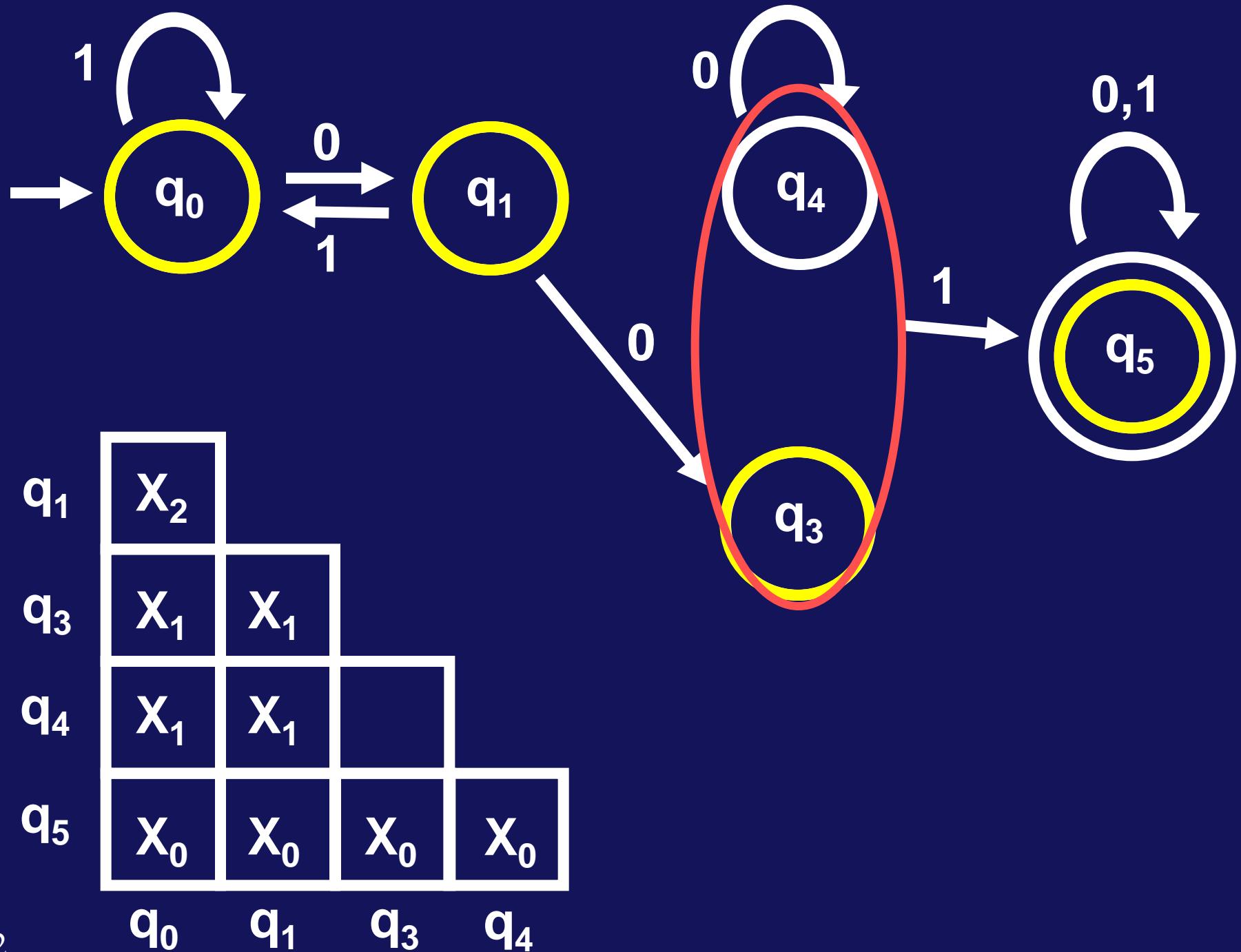


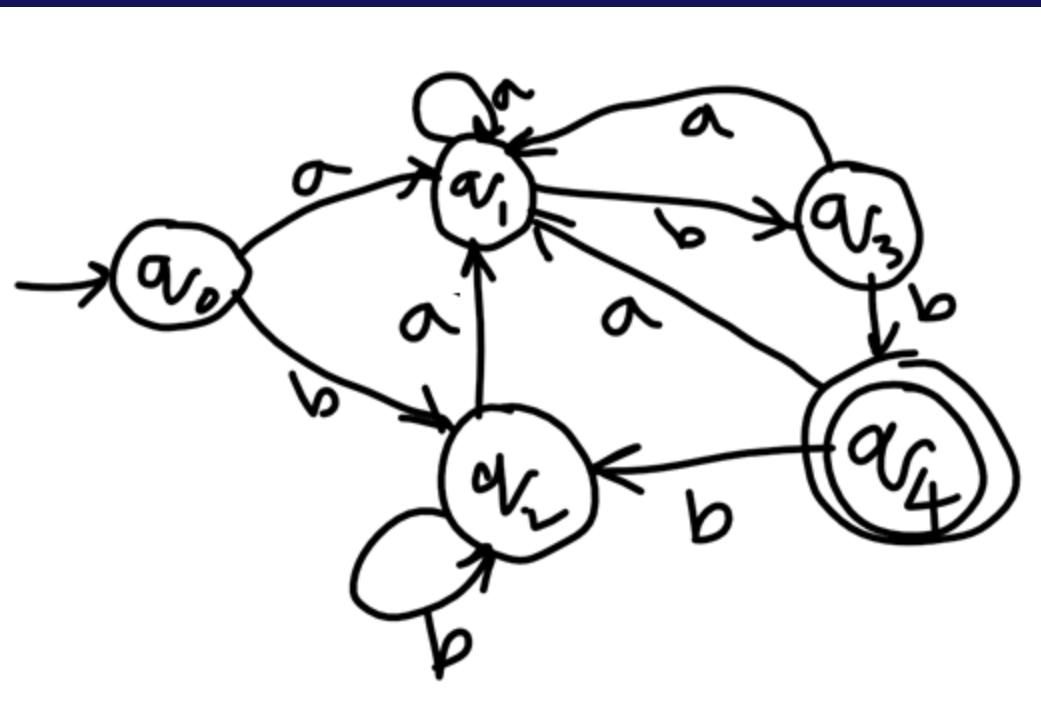


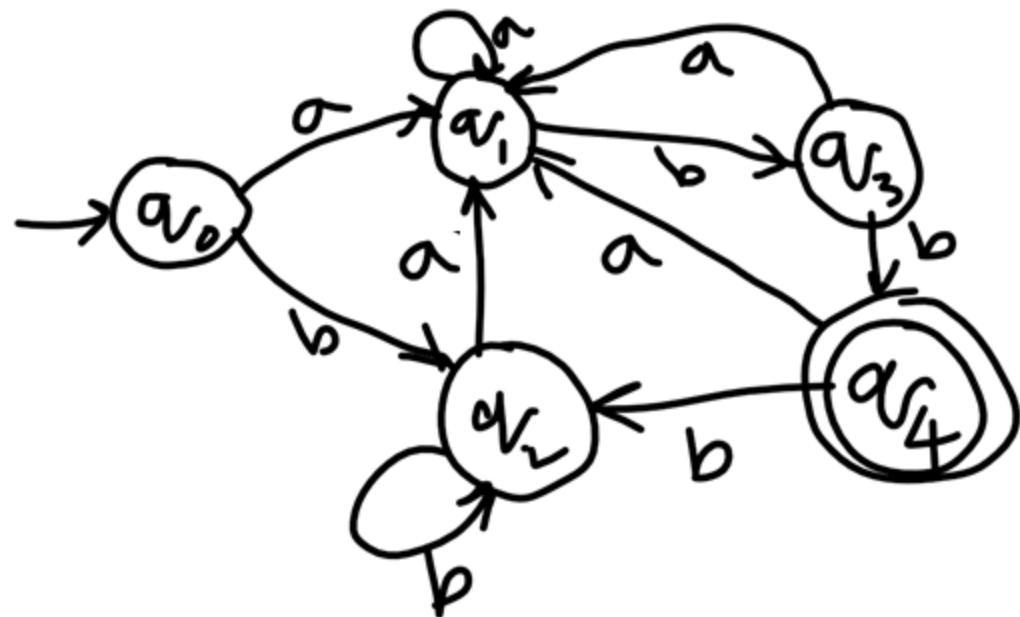




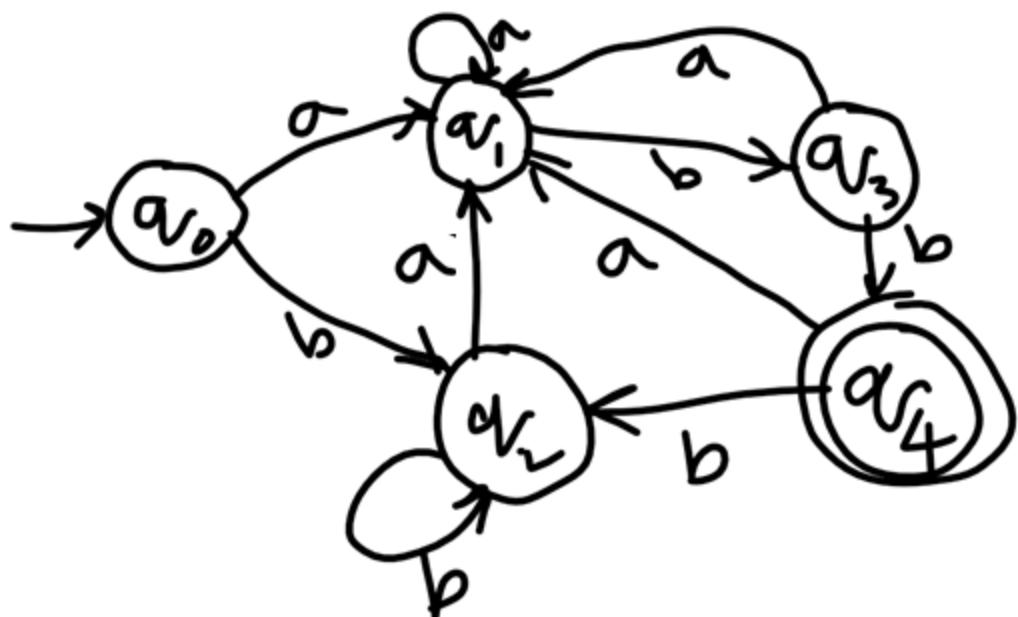






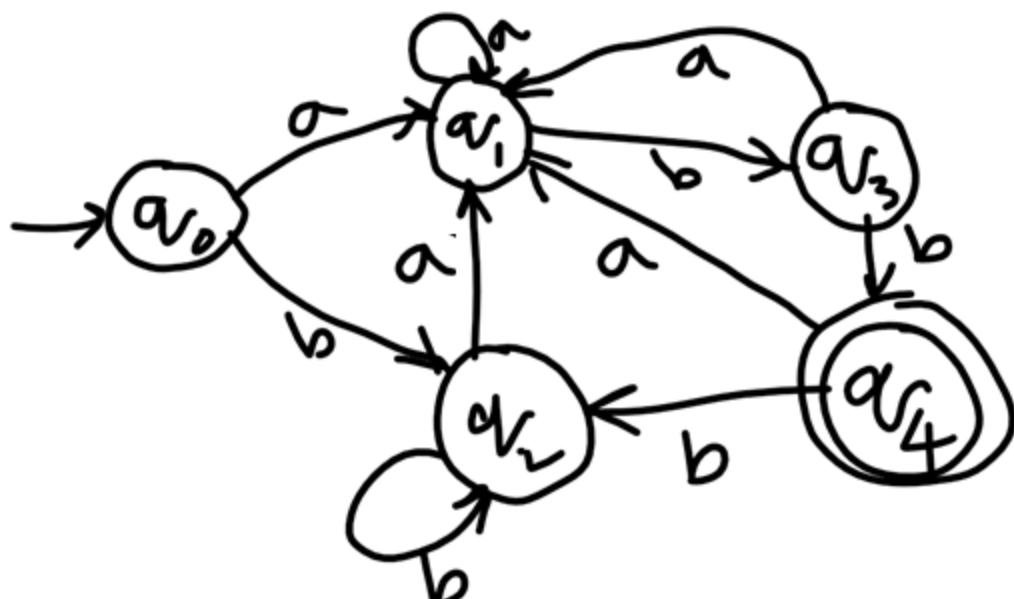


	a	b
a_0	a_1	a_2
a_1	a_1	a_3
a_2	a_1	a_2
a_3	a_1	a_4
a_4	a_1	a_2



	q_{r_4}	q_3	q_2	q_1
q_0	X0			
q_1	X0			
q_2	X0			
q_3	X0			

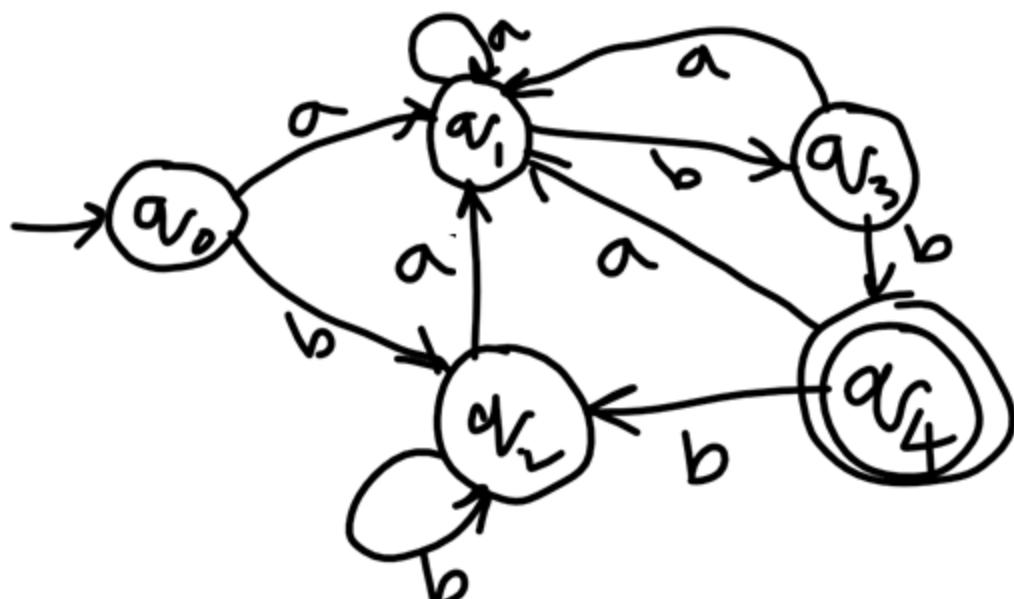
	a	b
q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	q_4
q_4	*	q_1



	q_{r_0}	q_3	q_2	q_1
q_0	x_0	x_1		
q_1	x_0	x_1		
q_2	x_0	x_1		
q_3	x_0			

→

a	b
q_0	q_1
q_1	q_1
q_2	q_1
q_3	q_2
q_4	q_2
$*$	q_1
q_4	q_1
	q_2



	a_{1x}	a_3	a_2	a_1
a_0	x_0	x_1		x_2
a_1	x_0	x_1	x_2	
a_2	x_0	x_1		
a_3	x_0			

→

	a	b
a_0	a_1	a_2
a_1	a_1	a_3
a_2	a_1	a_2
a_3	a_1	a_4
a_4	a_1	a_2

Eg

	0	1
→ v_0	v_1	v_5
v_1	v_6	v_2
* v_2	v_0	v_2
Removed	v_3	v_2
v_4	v_7	v_5
v_5	v_2	v_6
v_6	v_6	v_7
v_7	v_6	v_2

Step 1: Identify unreachable states

$$\begin{aligned} & \{v_0\}^+ \\ &= \{v_0, v_1, v_5, v_6, v_2, v_4, v_7\} \end{aligned}$$

v_3 is not reachable

Eg

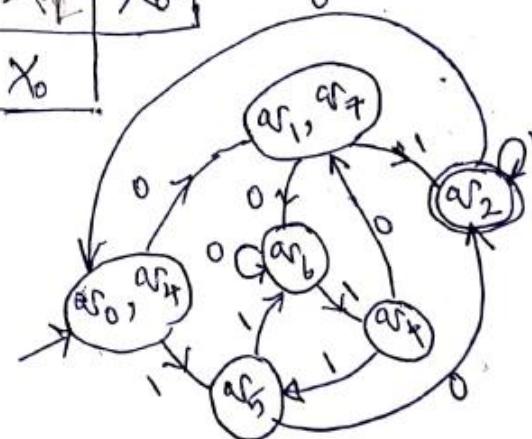
	0	1
v_0	v_1	v_5
v_1	v_6	v_2
v_2	v_0	v_2
Removed		
v_3	v_2	v_6
v_4	v_3	v_5
v_5	v_2	v_6
v_6	v_6	v_2
v_7	v_6	v_2

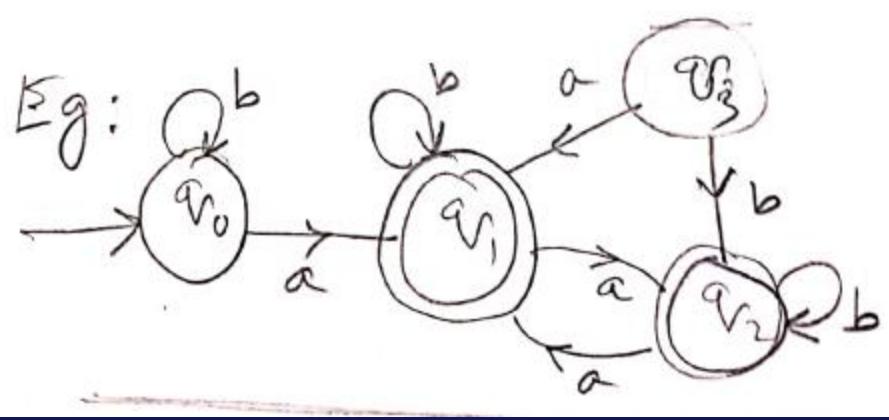
Step 1: Identify unreachable states

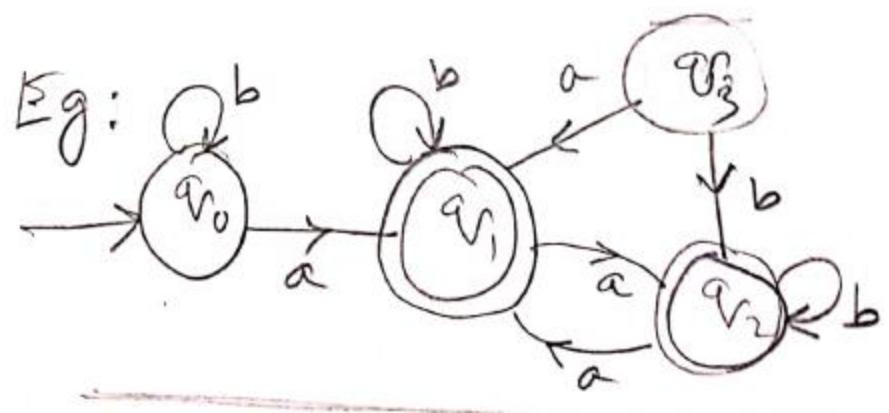
$$\{v_0\}^+ = \{v_0, v_1, v_5, v_6, v_2, v_4, v_7\}$$

v_3 is not reachable

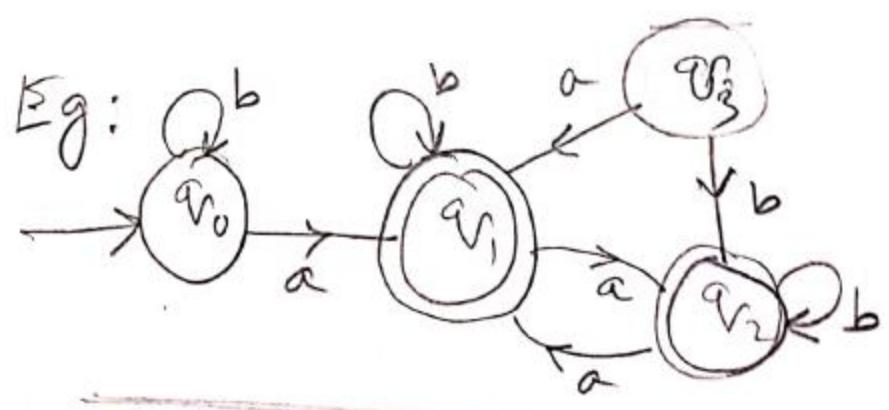
	v_7	v_6	v_5	v_4	v_2	v_1
v_0	x_1	x_2	x_1	\checkmark	x_0	x_1
v_1	\checkmark	x_1	x_1	x_1	x_0	0
v_2	x_0	x_0	x_0	x_0		
v_3	x_1	x_2	x_1			
v_4	x_1	x_1				
v_6		x_1				





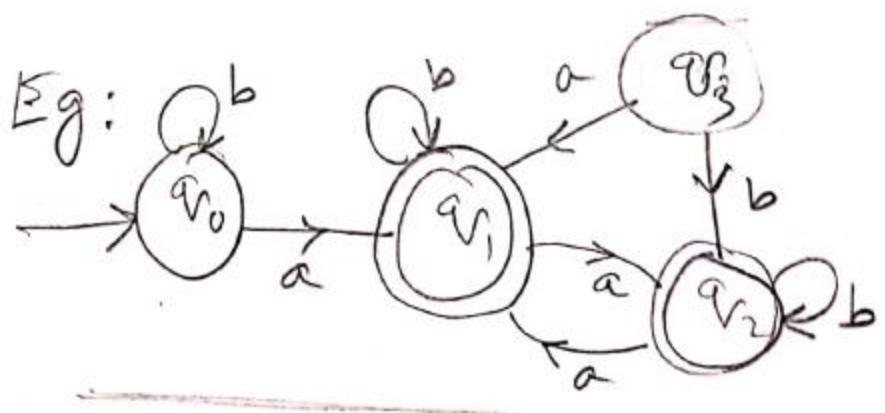


Unreachable = v_3



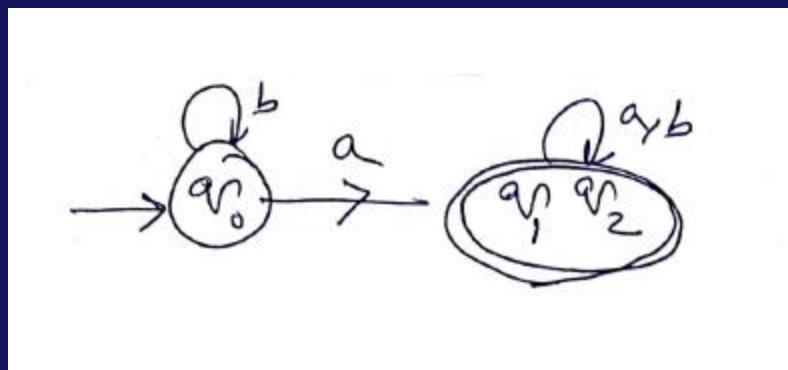
Unreachable = q_3

We get q_1 is equivalent to q_2 (how?)



Unreachable = q_3

We get q_1 is equivalent to q_2 (how?)



Minimal DFA

HOW TO PROVE THAT TWO DFAs ARE EQUIVALENT

- The following is an extract from the Ullman's book.
- Read that book for more information (Reading assignment)

4.4.2 Testing Equivalence of Regular Languages

The table-filling algorithm gives us an easy way to test if two regular languages are the same. Suppose languages L and M are each represented in some way, e.g., one by a regular expression and one by an NFA. Convert each representation to a DFA. Now, imagine one DFA whose states are the union of the states of the DFA's for L and M . Technically, this DFA has two start states, but actually the start state is irrelevant as far as testing state equivalence is concerned, so make any state the lone start state.

Now, test if the start states of the two original DFA's are equivalent, using the table-filling algorithm. If they are equivalent, then $L = M$, and if not, then $L \neq M$.

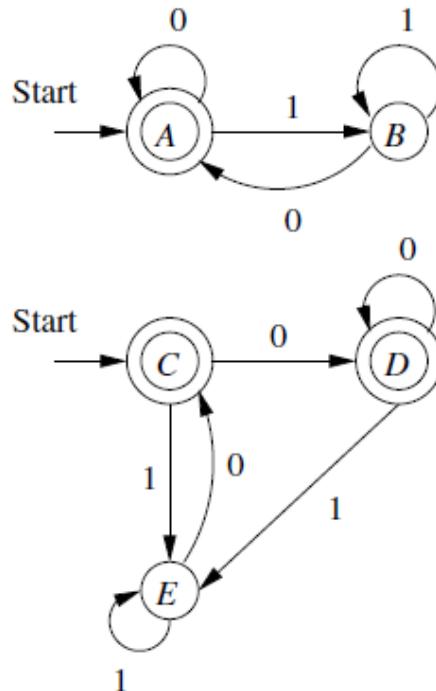
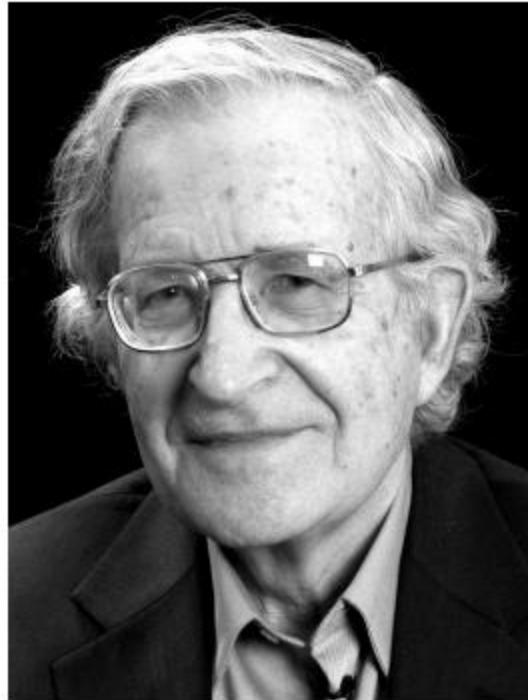


Figure 4.10: Two equivalent DFA's

Context Free Languages

Context Free Grammars

Context-Free Grammars



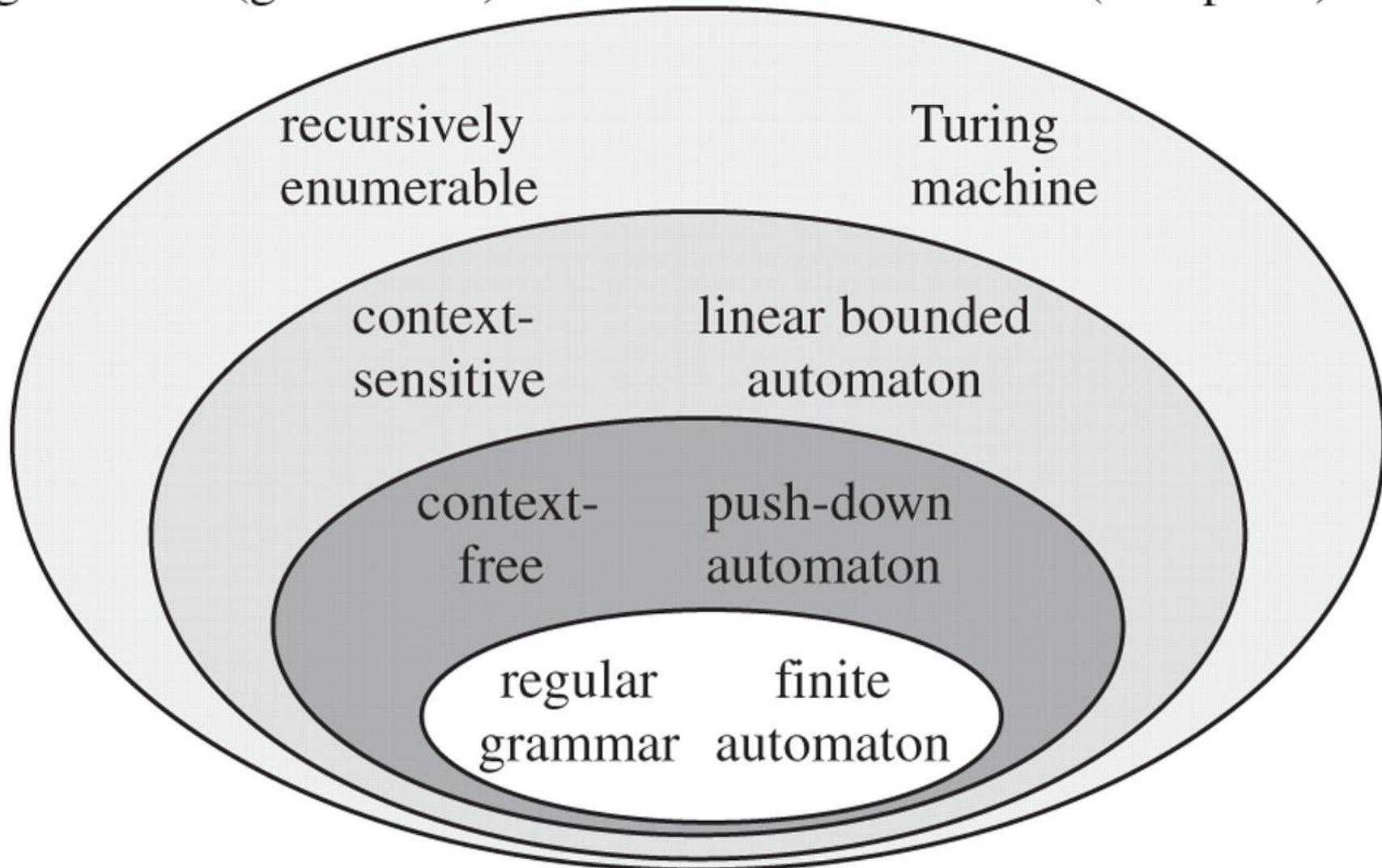
Noam Chomsky
(linguist, philosopher, logician, and activist)

In the formal languages of computer science and linguistics, the **Chomsky hierarchy** is a **hierarchy** of classes of formal grammars. This **hierarchy** of grammars was described by Noam **Chomsky** in 1956.

Chomsky Hierarchy

grammars (generators)

automata (acceptors)



The Hierarchy

Class	Grammars	Languages	Automaton
Type-0	Unrestricted	Recursive Enumerable	Turing Machine
Type-1	Context Sensitive	Context Sensitive	Linear- Bound
Type-2	Context Free	Context Free	Pushdown
Type-3	Regular	Regular	Finite

How production rules look like

Type	Grammar	Production rules
Type 0	unrestricted	$\alpha \rightarrow \beta$
Type 1	context-sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type 2	context-free	$A \rightarrow \gamma$
Type 3	regular	$A \rightarrow aB$ or $A \rightarrow Ba$

A grammar generates sentences (strings) in a language

Examples

Consider the grammar

$$S \rightarrow AB \tag{1}$$

$$A \rightarrow C \tag{2}$$

$$CB \rightarrow Cb \tag{3}$$

$$C \rightarrow a \tag{4}$$

where $\{a, b\}$ are terminals, and $\{S, A, B, C\}$ are non-terminals.

Examples

Consider the grammar

$$S \rightarrow AB \quad (1)$$

$$A \rightarrow C \quad (2)$$

$$CB \rightarrow Cb \quad (3)$$

$$C \rightarrow a \quad (4)$$

where $\{a, b\}$ are terminals, and $\{S, A, B, C\}$ are non-terminals.

We can derive the phrase “ab” from this grammar in the following way:

$$S \rightarrow AB, \text{ from (1)}$$

$$\rightarrow CB, \text{ from (2)}$$

$$\rightarrow Cb, \text{ from (3)}$$

$$\rightarrow ab, \text{ from (4)}$$

Examples

Consider the grammar

$$S \rightarrow \text{NounPhrase VerbPhrase} \quad (5)$$

$$\text{NounPhrase} \rightarrow \text{SingularNoun} \quad (6)$$

$$\text{SingularNoun VerbPhrase} \rightarrow \text{SingularNoun comes} \quad (7)$$

$$\text{SingularNoun} \rightarrow \text{John} \quad (8)$$

We can derive the phrase “John comes” from this grammar in the following way:

$$\begin{aligned} S &\rightarrow \text{NounPhrase VerbPhrase, from (1)} \\ &\rightarrow \text{SingularNoun VerbPhrase, from (2)} \\ &\rightarrow \text{SingularNoun comes, from (3)} \\ &\rightarrow \text{John comes, from (4)} \end{aligned}$$

Type	Grammar	Production rules
Type 0	unrestricted	$\alpha \rightarrow \beta$
Type 1	context-sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type 2	context-free	$A \rightarrow \gamma$
Type 3	regular	$A \rightarrow aB$ or $A \rightarrow Ba$

Definition (Context-Free Grammar)

A context-free grammar is a tuple $G = (V, T, P, S)$ where

- V is a finite set of variables (nonterminals, nonterminals vocabulary);
- T is a finite set of terminals (letters);
- $P \subseteq V \times (V \cup T)^*$ is a finite set of rewriting rules called productions,
 - We write $A \rightarrow \beta$ if $(A, \beta) \in P$;
- $S \in V$ is a distinguished start or “sentence” symbol.

Definition (Context-Free Grammar)

A context-free grammar is a tuple $G = (V, T, P, S)$ where

- V is a finite set of variables (nonterminals, nonterminals vocabulary);
- T is a finite set of terminals (letters);
- $P \subseteq V \times (V \cup T)^*$ is a finite set of rewriting rules called productions,
 - We write $A \rightarrow \beta$ if $(A, \beta) \in P$;
- $S \in V$ is a distinguished start or “sentence” symbol.

Example: $G_{0^n1^n} = (V, T, P, S)$ where

- $V = \{S\}$;
- $T = \{0, 1\}$;
- P is defined as

$$\begin{array}{lcl} S & \rightarrow & \varepsilon \\ S & \rightarrow & 0S1 \end{array}$$

- $S = S$.

Palindromes

$$G_{pal} = (\{P\}, \{0, 1\}, A, P)$$

1. $P \rightarrow \epsilon$
2. $P \rightarrow 0$
3. $P \rightarrow 1$
4. $P \rightarrow 0P0$
5. $P \rightarrow 1P1$

A context-free grammar for palindromes

Derivation:

- Let $G = (V, T, P, S)$ be a context-free grammar.
- Let $\alpha A \beta$ be a string in $(V \cup T)^* V (V \cup T)^*$
- We say that $\alpha A \beta$ yields the string $\alpha \gamma \beta$, and we write $\alpha A \beta \Rightarrow \alpha \gamma \beta$ if
 $A \rightarrow \gamma$ is a production rule in G .
- For strings $\alpha, \beta \in (V \cup T)^*$, we say that α derives β and we write
 $\alpha \xrightarrow{*} \beta$ if there is a sequence $\alpha_1, \alpha_2, \dots, \alpha_n \in (V \cup T)^*$ s.t.

$$\alpha \Rightarrow \alpha_1 \Rightarrow \alpha_2 \cdots \alpha_n \Rightarrow \beta.$$

\Rightarrow is also called direct derivation.

\xrightarrow{i} is to mean that the i th production is used in the direct derivation.

$\xrightarrow{*}$ is reflexive and transitive closure of \Rightarrow

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$

5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

A context-free grammar for simple expressions

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$

5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

A context-free grammar for simple expressions

T is the set of symbols $\{+, *, (,), a, b, 0, 1\}$ and P is the set of productions

1.	E	\rightarrow	I
2.	E	\rightarrow	$E + E$
3.	E	\rightarrow	$E * E$
4.	E	\rightarrow	(E)

5.	I	\rightarrow	a
6.	I	\rightarrow	b
7.	I	\rightarrow	Ia
8.	I	\rightarrow	Ib
9.	I	\rightarrow	$I0$
10.	I	\rightarrow	$I1$

A context-free grammar for simple expressions

T is the set of symbols $\{+, *, (,), a, b, 0, 1\}$ and P is the set of productions

- Can you find how the following is true.

$$E \xrightarrow{*} (a1 + b0 * a1)$$

Compact Notation for Productions

It is convenient to think of a production as “belonging” to the variable of its head. We shall often use remarks like “the productions for A ” or “ A -productions” to refer to the productions whose head is variable A . We may write the productions for a grammar by listing each variable once, and then listing all the bodies of the productions for that variable, separated by vertical bars. That is, the productions $A \rightarrow \alpha_1, A \rightarrow \alpha_2, \dots, A \rightarrow \alpha_n$ can be replaced by the notation $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$. For instance, the grammar for palindromes from Fig. 5.1 can be written as $P \rightarrow \epsilon | 0 | 1 | 0P0 | 1P1$.

CFL Definition

The language $L(G)$ accepted by a context-free grammar $G = (V, T, P, S)$ is the set

$$L(G) = \{w \in T^* : S \xrightarrow{*} w\}.$$

Leftmost and Rightmost Derivations

- Derivations are not unique.
- So, to bring uniqueness, we define two special type of derivations, viz., leftmost and rightmost.

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$

5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

A context-free grammar for simple expressions

$$E \underset{lm}{\Rightarrow} E * E \underset{lm}{\Rightarrow} I * E \underset{lm}{\Rightarrow} a * E \underset{lm}{\Rightarrow}$$

$$a * (E) \underset{lm}{\Rightarrow} a * (E + E) \underset{lm}{\Rightarrow} a * (I + E) \underset{lm}{\Rightarrow} a * (a + E) \underset{lm}{\Rightarrow}$$

$$a * (a + I) \underset{lm}{\Rightarrow} a * (a + I0) \underset{lm}{\Rightarrow} a * (a + I00) \underset{lm}{\Rightarrow} a * (a + b00)$$

We can also summarize the leftmost derivation by saying $E \underset{lm}{\stackrel{*}{\Rightarrow}} a * (a + b00)$, or express several steps of the derivation by expressions such as $E * E \underset{lm}{\stackrel{*}{\Rightarrow}} a * (E)$.

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$

5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

A context-free grammar for simple expressions

$$\begin{aligned}
 E &\xrightarrow{rm} E * E \xrightarrow{rm} E * (E) \xrightarrow{rm} E * (E + E) \xrightarrow{rm} \\
 &E * (E + I) \xrightarrow{rm} E * (E + I0) \xrightarrow{rm} E * (E + I00) \xrightarrow{rm} E * (E + b00) \xrightarrow{rm} \\
 &E * (I + b00) \xrightarrow{rm} E * (a + b00) \xrightarrow{rm} I * (a + b00) \xrightarrow{rm} a * (a + b00)
 \end{aligned}$$

This derivation allows us to conclude $E \xrightarrow[rm]{*} a * (a + b00)$. \square

Exercise

Consider the following grammar:

$$\begin{aligned} S &\rightarrow AS \mid \varepsilon. \\ A &\rightarrow aa \mid ab \mid ba \mid bb \end{aligned}$$

Give leftmost and rightmost derivations of the string *aabbba*.

$$G_{pal} = (\{P\}, \{0, 1\}, A, P)$$

1. $P \rightarrow \epsilon$
2. $P \rightarrow 0$
3. $P \rightarrow 1$
4. $P \rightarrow 0P0$
5. $P \rightarrow 1P1$

A context-free grammar for palindromes

Prove that $L(G_{pal})$ is the set of palindromes over the given alphabet.

$$G_{pal} = (\{P\}, \{0, 1\}, A, P)$$

1. $P \rightarrow \epsilon$
2. $P \rightarrow 0$
3. $P \rightarrow 1$
4. $P \rightarrow 0P0$
5. $P \rightarrow 1P1$

A context-free grammar for palindromes

Prove that $L(G_{pal})$ is the set of palindromes over the given alphabet.

- This proof has two parts (\Rightarrow and \Leftarrow)

$$G_{pal} = (\{P\}, \{0, 1\}, A, P)$$

1. $P \rightarrow \epsilon$
2. $P \rightarrow 0$
3. $P \rightarrow 1$
4. $P \rightarrow 0P0$
5. $P \rightarrow 1P1$

A context-free grammar for palindromes

Prove that $L(G_{pal})$ is the set of palindromes over the given alphabet.

- This proof has two parts (\Rightarrow and \Leftarrow)
 - 1) $(w = w^R) \Rightarrow w \in L(G_{pal})$
 - 2) $w \in L(G_{pal}) \Rightarrow (w = w^R)$

$$(w = w^R) \Rightarrow w \in L(G_{pal})$$

1. $P \rightarrow \epsilon$
2. $P \rightarrow 0$
3. $P \rightarrow 1$
4. $P \rightarrow 0P0$
5. $P \rightarrow 1P1$

A context-free grammar for palindromes

- Proof [by induction on $|w|$]:

BASIS: We use lengths 0 and 1 as the basis.

If $|w| = 0$ or $|w| = 1$, then w is ϵ , 0, or 1.

Since there are productions $P \rightarrow \epsilon$, $P \rightarrow 0$, and $P \rightarrow 1$, we conclude that $P \xrightarrow{*} w$ in any of these basis cases.

- Note, $w \in L(G_{pal})$ is same $P \xrightarrow{*} w$
- Note, $(w = w^R)$ means w begins and ends with the same character.

BASIS: We use lengths 0 and 1 as the basis.

If $|w| = 0$ or $|w| = 1$, then w is ϵ , 0, or 1.

Since there are productions $P \rightarrow \epsilon$, $P \rightarrow 0$, and $P \rightarrow 1$, we conclude that $P \xrightarrow{*} w$ in any of these basis cases.

BASIS: We use lengths 0 and 1 as the basis.

If $|w| = 0$ or $|w| = 1$, then w is ϵ , 0, or 1.

Since there are productions $P \rightarrow \epsilon$, $P \rightarrow 0$, and $P \rightarrow 1$, we conclude that $P \xrightarrow{*} w$ in any of these basis cases.

Inductive Hypothesis: Let for $|w| \leq k$ where ($w = w^R$), $P \xrightarrow{*} w$ is true.

Inductive Step: We need to show for $|w| = k + 1, P \xrightarrow{*} w$ is true.

BASIS: We use lengths 0 and 1 as the basis.

If $|w| = 0$ or $|w| = 1$, then w is ϵ , 0, or 1.

Since there are productions $P \rightarrow \epsilon$, $P \rightarrow 0$, and $P \rightarrow 1$, we conclude that $P \xrightarrow{*} w$ in any of these basis cases.

Inductive Hypothesis: Let for $|w| \leq k$ where ($w = w^R$), $P \xrightarrow{*} w$ is true.

Inductive Step: We need to show for $|w| = k + 1, P \xrightarrow{*} w$ is true.

Note, $w = 0x0$ or $w = 1x1$, where $|x| = k - 1$.

Then, $P \Rightarrow 0P0 \xrightarrow{*} 0x0$ (Since $|x| \leq k$, so $P \xrightarrow{*} x$ is true).

So, $P \xrightarrow{*} w$ is true. With a similar argument, $P \Rightarrow 1P1 \xrightarrow{*} 1x1$

BASIS: We use lengths 0 and 1 as the basis.

If $|w| = 0$ or $|w| = 1$, then w is ϵ , 0, or 1.

Since there are productions $P \rightarrow \epsilon$, $P \rightarrow 0$, and $P \rightarrow 1$, we conclude that $P \xrightarrow{*} w$ in any of these basis cases.

Inductive Hypothesis: Let for $|w| \leq k$ where ($w = w^R$), $P \xrightarrow{*} w$ is true.

Inductive Step: We need to show for $|w| = k + 1$, $P \xrightarrow{*} w$ is true.

Note, $w = 0x0$ or $w = 1x1$, where $|x| = k - 1$.

Then, $P \Rightarrow 0P0 \xrightarrow{*} 0x0$ (Since $|x| \leq k$, so $P \xrightarrow{*} x$ is true).

So, $P \xrightarrow{*} w$ is true. With a similar argument, $P \Rightarrow 1P1 \xrightarrow{*} 1x1$

This completes our proof for :

$$(w = w^R) \Rightarrow w \in L(G_{pal})$$

$$w \in L(G_{pal}) \Rightarrow (w = w^R)$$

- Proof [by induction on number of steps in the derivation]:

BASIS: If the derivation is one step, then it must use one of the three productions that do not have P in the body. That is, the derivation is $P \Rightarrow \epsilon$, $P \Rightarrow 0$, or $P \Rightarrow 1$. Since ϵ , 0, and 1 are all palindromes, the basis is proven.

INDUCTION:

- Assume for n steps it is true.
- Then, show for $(n+1)$ steps it must be true.

Left as an exercise.

Sentential Forms

$G = (\bar{V}, \bar{T}, P, S)$ is a CFG, then any string α in $(V \cup T)^*$ such that $S \xrightarrow{*} \alpha$ is a *sentential form*.

Sentential Forms

$G = (\bar{V}, \bar{T}, P, S)$ is a CFG, then any string α in $(V \cup T)^*$ such that $S \xrightarrow{*} \alpha$ is a *sentential form*.

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$

5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

A context-free grammar for simple expressions

$$E \Rightarrow E * E \Rightarrow E * (E) \Rightarrow E * (E + E) \Rightarrow E * (I + E)$$

Sentential Forms

$G = (\bar{V}, \bar{T}, P, S)$ is a CFG, then any string α in $(V \cup T)^*$ such that $S \xrightarrow{*} \alpha$ is a *sentential form*.

If $S \xrightarrow{lm}^* \alpha$, then α is a *left-sentential form*,

and if $S \xrightarrow{rm}^* \alpha$, then α is a *right-sentential form*.

Note that the language $L(G)$ is those sentential forms that are in T^* ; i.e., they consist solely of terminals.

1.	E	\rightarrow	I
2.	E	\rightarrow	$E + E$
3.	E	\rightarrow	$E * E$
4.	E	\rightarrow	(E)

5.	I	\rightarrow	a
6.	I	\rightarrow	b
7.	I	\rightarrow	Ia
8.	I	\rightarrow	Ib
9.	I	\rightarrow	$I0$
10.	I	\rightarrow	$I1$

A context-free grammar for simple expressions

$$E \Rightarrow E * E \Rightarrow E * (E) \Rightarrow E * (E + E) \Rightarrow E * (I + E)$$

- Is this sentential form left-sentential? Or right-sentential?

1.	E	\rightarrow	I
2.	E	\rightarrow	$E + E$
3.	E	\rightarrow	$E * E$
4.	E	\rightarrow	(E)

5.	I	\rightarrow	a
6.	I	\rightarrow	b
7.	I	\rightarrow	Ia
8.	I	\rightarrow	Ib
9.	I	\rightarrow	$I0$
10.	I	\rightarrow	$I1$

A context-free grammar for simple expressions

$$E \Rightarrow E * E \Rightarrow E * (E) \Rightarrow E * (E + E) \Rightarrow E * (I + E)$$

- Is this sentential form left-sentential? Or right-sentential?
- It is a sentential form. But neither left nor right.

Exercise 5.1.2: The following grammar generates the language of regular expression $0^*1(0+1)^*$:

$$\begin{array}{lcl} S & \rightarrow & A1B \\ A & \rightarrow & 0A \mid \epsilon \\ B & \rightarrow & 0B \mid 1B \mid \epsilon \end{array}$$

Give leftmost and rightmost derivations of the following strings:

- * a) 00101.
- b) 1001.
- c) 00011.

Note, the given grammar is not a regular grammar (even-though it generates a regular language).

Can you find $L(G)$?

- $S \rightarrow aS|bS|a|b|\epsilon$

Can you find $L(G)$?

- $S \rightarrow aS|bS|a|b|\epsilon$
- Answer: All strings. Σ^*

Can you find $L(G)$?

1. $S \rightarrow S_1S|\epsilon,$
2. $S_1 \rightarrow aS_1b|ab$

Can you find $L(G)$?

1. $S \rightarrow S_1S|\epsilon,$
2. $S_1 \rightarrow aS_1b|ab$

Recall that the $S \rightarrow aSb|\epsilon$ generates $\{a^n b^n | n \geq 0\}$.

Can you find $L(G)$?

1. $S \rightarrow S_1S|\epsilon,$
2. $S_1 \rightarrow aS_1b|ab$

Recall that the $S \rightarrow aSb|\epsilon$ generates $\{a^n b^n | n \geq 0\}$.

Starting from S_1 we get $\{a^n b^n | n \geq 1\}$

Can you find $L(G)$?

1. $S \rightarrow S_1S|\epsilon,$
2. $S_1 \rightarrow aS_1b|ab$

Recall that the $S \rightarrow aSb|\epsilon$ generates $\{a^n b^n | n \geq 0\}$.

Starting from S_1 we get $\{a^n b^n | n \geq 1\}$

The answer:

$$a^{n_1}b^{n_1}a^{n_2}b^{n_2} \dots a^{n_k}b^{n_k} \in L(G)$$

$$L(G) = (\{a^n b^n | n \geq 1\})^*$$

Can you find $L(G)$?

$$S \rightarrow SS \mid [S] \mid (S) \mid [] \mid ()$$

Can you find $L(G)$?

$$S \rightarrow SS \mid [S] \mid (S) \mid [] \mid ()$$

Set of all balanced parentheses with alphabet
 $\{ (,), [,] \}$

Can you find $L(G)$?

1. $S \rightarrow aB|bA$
2. $B \rightarrow b|bS|aBB$
3. $A \rightarrow a|aS|bAA$

Can you find $L(G)$?

1. $S \rightarrow aB|bA$
2. $B \rightarrow b|bS|aBB$
3. $A \rightarrow a|aS|bAA$

Produces strings with equal number of a's and b's.

Can you find $L(G)$?

1. $S \rightarrow SaSbS | SbSaS | \epsilon$

Can you find $L(G)$?

$$1. \ S \rightarrow SaSbS | SbSaS | \epsilon$$

Produces strings with equal number of a's and b's.

With one difference than the previous CFG. **What is it?**

Parse tree and ambiguity

Parse tree representation of the derivation.

- It is tree representation of the derivation.
- For a given derivation, there is only one parse tree.
- But, for a given parse tree, there may be many derivations.

1. $P \rightarrow \epsilon$
2. $P \rightarrow 0$
3. $P \rightarrow 1$
4. $P \rightarrow 0P0$
5. $P \rightarrow 1P1$

Figure 5.1: A context-free grammar for palindromes

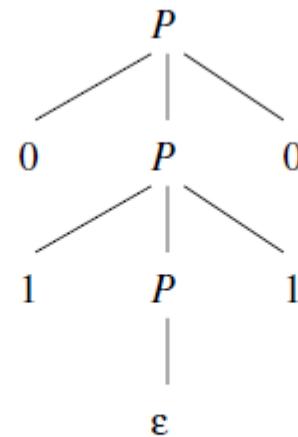


Figure 5.5: A parse tree showing the derivation $P \xrightarrow{*} 0110$

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$

5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

Figure 5.2: A context-free grammar for simple expressions

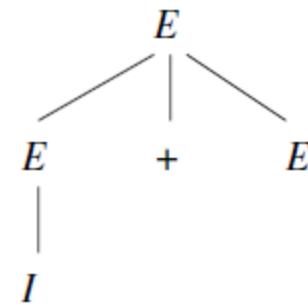


Figure 5.4: A parse tree showing the derivation of $I + E$ from E

1. Each interior node is labeled by a variable in V .
2. Each leaf is labeled by either a variable, a terminal, or ϵ . However, if the leaf is labeled ϵ , then it must be the only child of its parent.
3. If an interior node is labeled A , and its children are labeled

$$X_1, X_2, \dots, X_k$$

respectively, from the left, then $A \rightarrow X_1 X_2 \cdots X_k$ is a production in P .

5.2.2 The Yield of a Parse Tree

1. The yield is a terminal string. That is, all leaves are labeled either with a terminal or with ϵ .
2. The root is labeled by the start symbol.

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$

5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

Figure 5.2: A context-free grammar for simple expressions

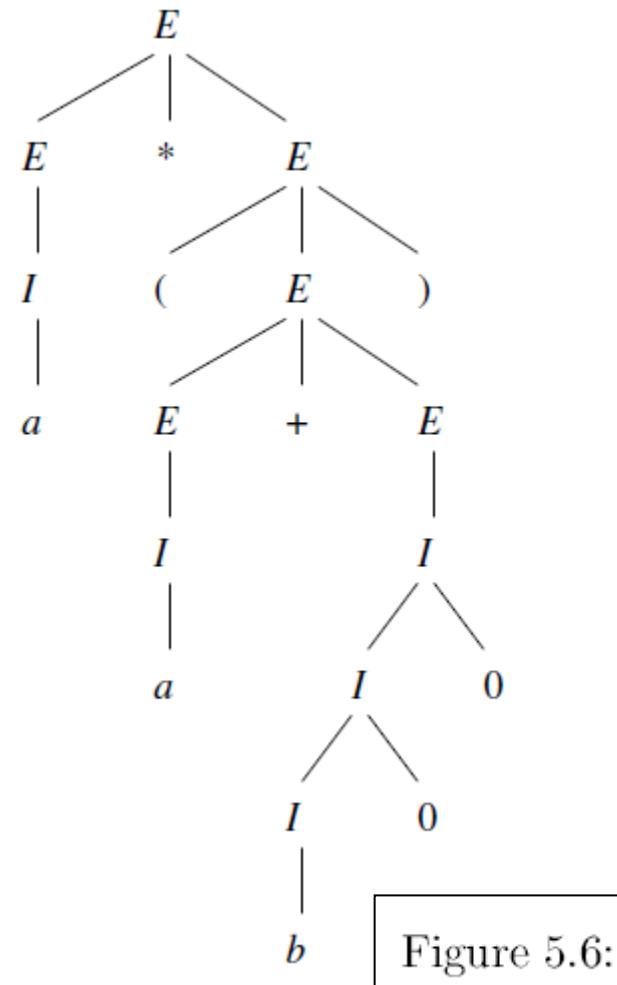


Figure 5.6:

Parse tree for the yield $a * (a + b00)$

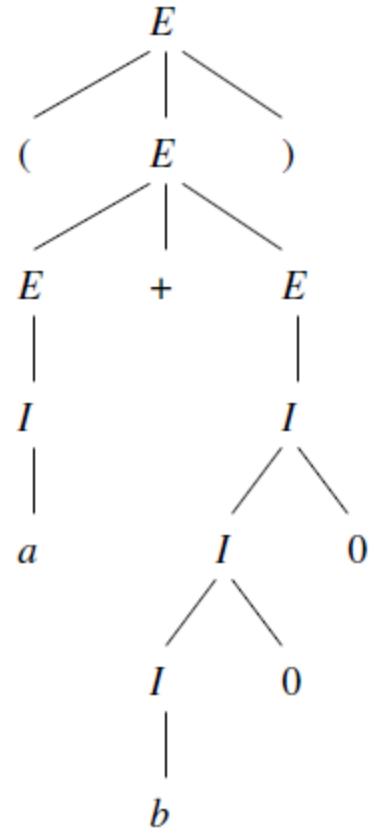
Parse tree representation of the derivation.

- It is tree representation of the derivation.
- For a given derivation, there is only one parse tree.
- But, for a given parse tree, there may be many derivations.

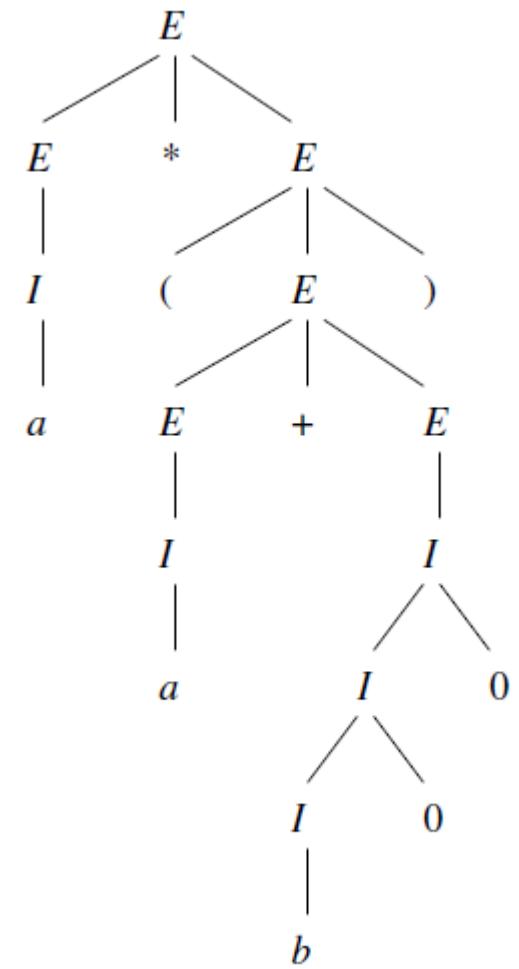
- For a given parse tree there is a unique leftmost derivation.
- Similarly, for a given parse tree there is a unique rightmost derivation.

$$E \xrightarrow{lm} (E) \xrightarrow{lm} (E + E) \xrightarrow{lm} (I + E) \xrightarrow{lm} (a + E) \xrightarrow{lm}$$

$$(a + I) \xrightarrow{lm} (a + I0) \xrightarrow{lm} (a + I00) \xrightarrow{lm} (a + b00)$$



- Can you find the rightmost derivation?

$$E \Rightarrow \underset{lm}{E * E} \Rightarrow \underset{lm}{I * E} \Rightarrow \underset{lm}{a * E} \Rightarrow$$
$$a * (E) \Rightarrow \underset{lm}{a * (E + E)} \Rightarrow \underset{lm}{a * (I + E)} \Rightarrow \underset{lm}{a * (a + E)} \Rightarrow$$
$$a * (a + I) \Rightarrow \underset{lm}{a * (a + I0)} \Rightarrow \underset{lm}{a * (a + I00)} \Rightarrow \underset{lm}{a * (a + b00)}$$


- Can you find the rightmost derivation?

- **Can you do this?**

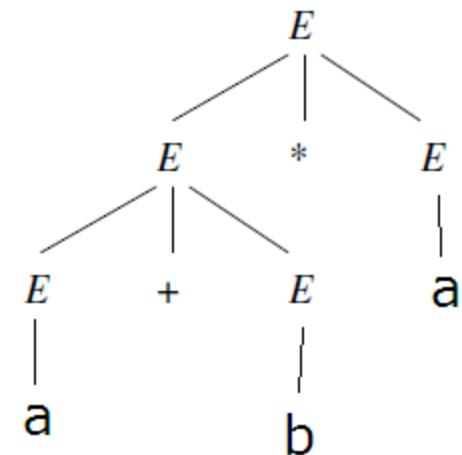
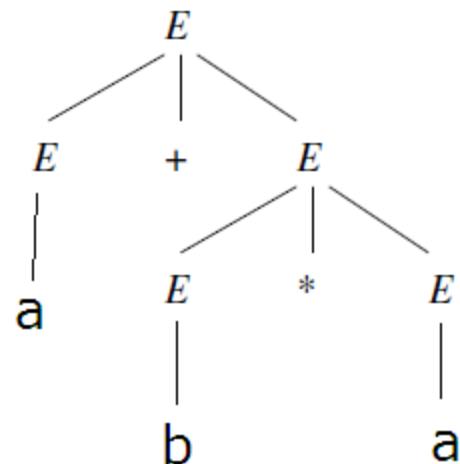
! **Exercise 5.2.2:** Suppose that G is a CFG without any productions that have ϵ as the right side. If w is in $L(G)$, the length of w is n , and w has a derivation of m steps, show that w has a parse tree with $n + m$ nodes.

Ambiguous grammar

- The CFG is ambiguous, if there is a string in the language for which there are more than one parse tree.
- This is equivalent to say, “there are more than one leftmost derivation for a string, hence the grammar is ambiguous”.
- Similarly, with rightmost derivation

- Two parse trees for the yield $a+b^*a$

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow a$
$E \rightarrow b$



Can we remove the ambiguity?

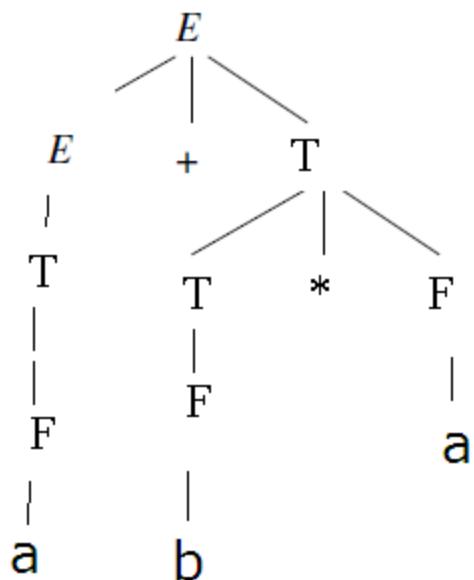
- Finding whether a given CFG is ambiguous or not is an undecidable problem !
- There are some CFLs for which it is impossible to have an unambiguous CFG.

Can we remove the ambiguity?

- Finding whether a given CFG is ambiguous or not is an undecidable problem !
- There are some CFLs for which it is impossible to have an unambiguous CFG.
- But, the situation is not so unpromising.
- For many situations in practice, we can handcraft unambiguous CFG for a given ambiguous one.

$$E \rightarrow T \mid E + T$$
$$T \rightarrow F \mid T * F$$
$$F \rightarrow a \mid b$$

An unambiguous expression grammar



This is the only parse tree for $a+b^*a$

- For an expression grammar, injecting precedence and associativity of operators can make them unambiguous.

1.	E	\rightarrow	I
2.	E	\rightarrow	$E + E$
3.	E	\rightarrow	$E * E$
4.	E	\rightarrow	(E)
5.	I	\rightarrow	a
6.	I	\rightarrow	b
7.	I	\rightarrow	Ia
8.	I	\rightarrow	Ib
9.	I	\rightarrow	$I0$
10.	I	\rightarrow	$I1$

$$\begin{array}{lcl} E & \rightarrow & T \mid E + T \\ T & \rightarrow & F \mid T * F \\ F & \rightarrow & I \mid (E) \\ I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{array}$$

Figure 5.19: An unambiguous expression grammar

Figure 5.2: A context-free grammar for simple expressions

$$\begin{array}{lcl}
 I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\
 F & \rightarrow & I \mid (E) \\
 T & \rightarrow & F \mid T * F \\
 E & \rightarrow & T \mid E + T
 \end{array}$$

Figure 5.19: An unambiguous expression grammar

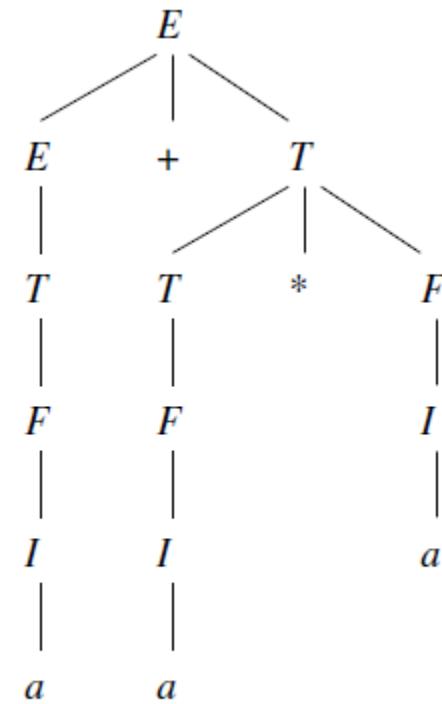


Figure 5.20: The sole parse tree for $a + a * a$

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$

5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

Figure 5.2: A context-free grammar for simple expressions

- With above we get two parse trees for the same yield.

$$\begin{array}{lcl} I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ F & \rightarrow & I \mid (E) \\ T & \rightarrow & F \mid T * F \\ E & \rightarrow & T \mid E + T \end{array}$$

Figure 5.19: An unambiguous expression grammar

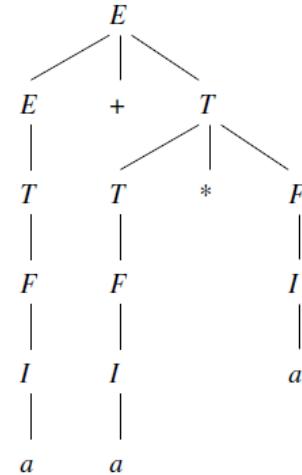


Figure 5.20: The sole parse tree for $a + a * a$

Inherent ambiguity

- A CFL is said to be inherently ambiguous, if every CFG that generates the language is ambiguous.
- Note, in this case we say CFL is ambiguous.
 - Earlier we said CFG is ambiguous.

An example of ambiguous CFL

$$L = \{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$$

That is, L consists of strings in $\mathbf{a}^+ \mathbf{b}^+ \mathbf{c}^+ \mathbf{d}^+$ such that either:

1. There are as many a 's as b 's and as many c 's as d 's, or
2. There are as many a 's as d 's and as many b 's as c 's.

An example of ambiguous CFL

$$L = \{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$$

That is, L consists of strings in $\mathbf{a}^+ \mathbf{b}^+ \mathbf{c}^+ \mathbf{d}^+$ such that either:

1. There are as many a 's as b 's and as many c 's as d 's, or
2. There are as many a 's as d 's and as many b 's as c 's.

$$\begin{array}{lcl} S & \rightarrow & AB \mid C \\ A & \rightarrow & aAb \mid ab \\ B & \rightarrow & cBd \mid cd \\ C & \rightarrow & aCd \mid aDd \\ D & \rightarrow & bDc \mid bc \end{array}$$

One CFG, for the CFL

An example of ambiguous CFL

$$L = \{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$$

That is, L consists of strings in $\mathbf{a}^+ \mathbf{b}^+ \mathbf{c}^+ \mathbf{d}^+$ such that either:

1. There are as many a 's as b 's and as many c 's as d 's, or
2. There are as many a 's as d 's and as many b 's as c 's.

- | | |
|--|--|
| 1. $S \xrightarrow[lm]{} AB \xrightarrow[lm]{} aAbB \xrightarrow[lm]{} aabbB \xrightarrow[lm]{} aabbcBd \xrightarrow[lm]{} aabbccdd$ | 2. $S \xrightarrow[lm]{} C \xrightarrow[lm]{} aCd \xrightarrow[lm]{} aaDdd \xrightarrow[lm]{} aabDcdd \xrightarrow[lm]{} aabbccdd$ |
|--|--|

Two leftmost derivations for the same string

$$\begin{array}{lcl} S & \rightarrow & AB \mid C \\ A & \rightarrow & aAb \mid ab \\ B & \rightarrow & cBd \mid cd \\ C & \rightarrow & aCd \mid aDd \\ D & \rightarrow & bDc \mid bc \end{array}$$

One CFG, for the CFL

$$\begin{array}{lcl}
 S & \rightarrow & AB \mid C \\
 A & \rightarrow & aAb \mid ab \\
 B & \rightarrow & cBd \mid cd \\
 C & \rightarrow & aCd \mid aDd \\
 D & \rightarrow & bDc \mid bc
 \end{array}$$

1. $S \xrightarrow{lm} AB \xrightarrow{lm} aAbB \xrightarrow{lm} aabbB \xrightarrow{lm} aabbcBd \xrightarrow{lm} aabbccdd$
2. $S \xrightarrow{lm} C \xrightarrow{lm} aCd \xrightarrow{lm} aaDdd \xrightarrow{lm} aabDcdd \xrightarrow{lm} aabbccdd$

One CFG, for the CFL

Two leftmost derivations for the same string

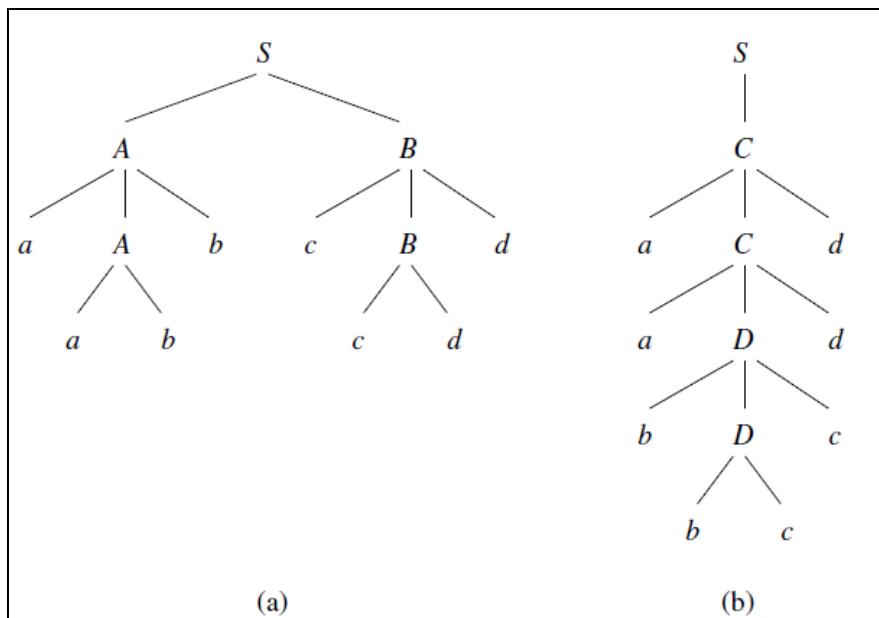


Figure 5.23: Two parse trees for *aabbccdd*

How to understand that every CFG is ambiguous.

$$L = \{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$$

That is, L consists of strings in $\mathbf{a}^+ \mathbf{b}^+ \mathbf{c}^+ \mathbf{d}^+$ such that either:

1. There are as many a 's as b 's and as many c 's as d 's, or
 2. There are as many a 's as d 's and as many b 's as c 's.
-
- Proof is complicated.
 - But the essence is, the grammar has two parts one generating strings in each of the above union.
 - There are some strings that are common between these two parts. These can be generated from two ways.

* **Exercise 5.4.1:** Consider the grammar

$$S \rightarrow aS \mid aSbS \mid \epsilon$$

This grammar is ambiguous. Show in particular that the string aab has two:

- a) Parse trees.
- b) Leftmost derivations.
- c) Rightmost derivations.

Chapter 7

Properties of Context-Free Languages

- We first simplify CFGs
- Then we prove “pumping lemma” for CFLs
- Then closure properties and decision properties are considered.

Chomsky Normal Form (CNF)

- Every CFL (without ϵ) is generated by a CFG in which all productions are of the form

$A \rightarrow BC$ or $A \rightarrow a$

But, this requires preprocessing of the
CFG ...

- We must eliminate *useless symbols*,
- We must eliminate ϵ -*productions*, and
- We must eliminate *unit productions*.

Eliminating useless symbols

We say a symbol X is *useful* for a grammar $G = (V, T, P, S)$ if there is some derivation of the form $S \xrightarrow{*} \alpha X \beta \xrightarrow{*} w$, where w is in T^* .

- Note that X may be either a variable or a terminal symbol.
- If X is not useful, we say it is *useless*.
- This useless symbol elimination should not change the CFL. We see such a one.
 - The language is in tact, but useless are removed.

Our approach to eliminating useless symbols begins by identifying the two things a symbol has to be able to do to be useful:

Our approach to eliminating useless symbols begins by identifying the two things a symbol has to be able to do to be useful:

1. We say X is *generating* if $X \xrightarrow{*} w$ for some terminal string w . Note that every terminal is generating, since w can be that terminal itself, which is derived by zero steps.

Our approach to eliminating useless symbols begins by identifying the two things a symbol has to be able to do to be useful:

1. We say X is *generating* if $X \xrightarrow{*} w$ for some terminal string w . Note that every terminal is generating, since w can be that terminal itself, which is derived by zero steps.
2. We say X is *reachable* if there is a derivation $S \xrightarrow{*} \alpha X \beta$ for some α and β .

Order is important

1. Eliminate nongenerating symbols *first*,
2. *then*, from the remaining eliminate unreachable symbols.

Order is important

1. Eliminate nongenerating symbols *first*,
 2. *then*, from the remaining eliminate unreachable symbols.
-
- Whatever left are only useful ones.

Example 7.1: Consider the grammar:

$$\begin{array}{l} S \rightarrow AB \mid a \\ A \rightarrow b \end{array}$$

Example 7.1: Consider the grammar:

$$\begin{aligned} S &\rightarrow AB \mid a \\ A &\rightarrow b \end{aligned}$$

- Find non-generating symbols.
- How to find this?

Example 7.1: Consider the grammar:

$$\begin{aligned} S &\rightarrow AB \mid a \\ A &\rightarrow b \end{aligned}$$

- Find non-generating symbols.
- How to find this?
- We can find generating symbols, inductively.

Example 7.1: Consider the grammar:

$$\begin{aligned} S &\rightarrow AB \mid a \\ A &\rightarrow b \end{aligned}$$

- Find non-generating symbols.
- How to find this?
- We can find generating symbols, inductively.
- Basis: Every symbol of T is generating. (Why?)
- Induction: if every symbol on RHS of a production $A \rightarrow \alpha$ is generating, then A is generating.

Example 7.1: Consider the grammar:

$$\begin{aligned} S &\rightarrow AB \mid a \\ A &\rightarrow b \end{aligned}$$

- What are the generating symbols?

Example 7.1: Consider the grammar:

$$\begin{aligned} S &\rightarrow AB \mid a \\ A &\rightarrow b \end{aligned}$$

- What are the generating symbols?
- {a,b,A,S}

Example 7.1: Consider the grammar:

$$\begin{aligned} S &\rightarrow AB \mid a \\ A &\rightarrow b \end{aligned}$$

- What are the generating symbols?
- {a,b,A,S}
- So the non-generating symbol is B.

Example 7.1: Consider the grammar:

$$\begin{aligned} S &\rightarrow AB \mid a \\ A &\rightarrow b \end{aligned}$$

- What are the generating symbols?
- {a,b,A,S}
- So the non-generating symbol is B.
- So removing B we are left with $S \rightarrow a$, $A \rightarrow b$

Theorem 7.4: The algorithm above finds all and only the generating symbols of G .

- Proof is skipped.

Finding reachable symbols

- By induction, again.
- Basis. S is reachable.
- Induction. A is reachable, and $A \rightarrow \alpha$, then all symbols in α are reachable.

Example 7.1: Consider the grammar:

$$\begin{array}{l} S \rightarrow AB \mid a \\ A \rightarrow b \end{array}$$

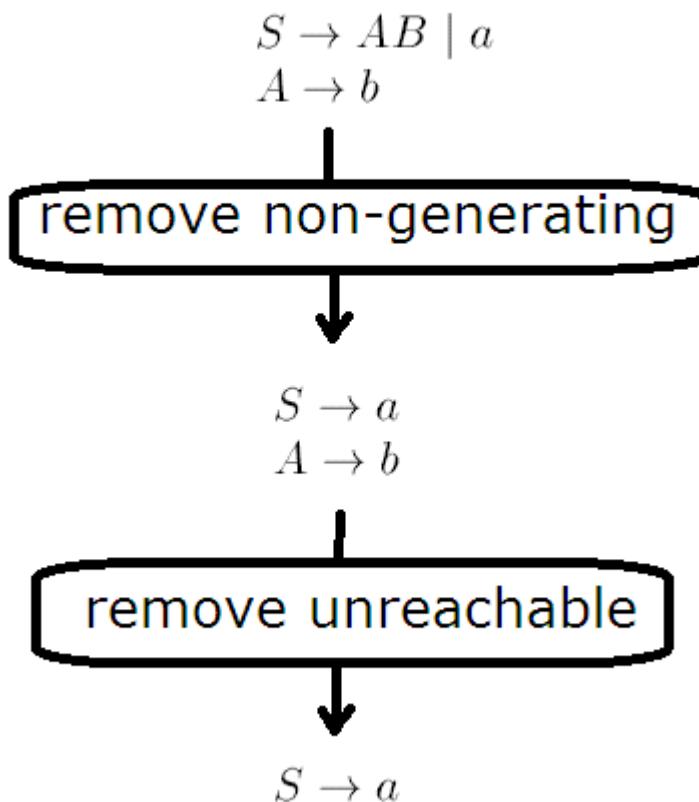
- Find reachable.

Example 7.1: Consider the grammar:

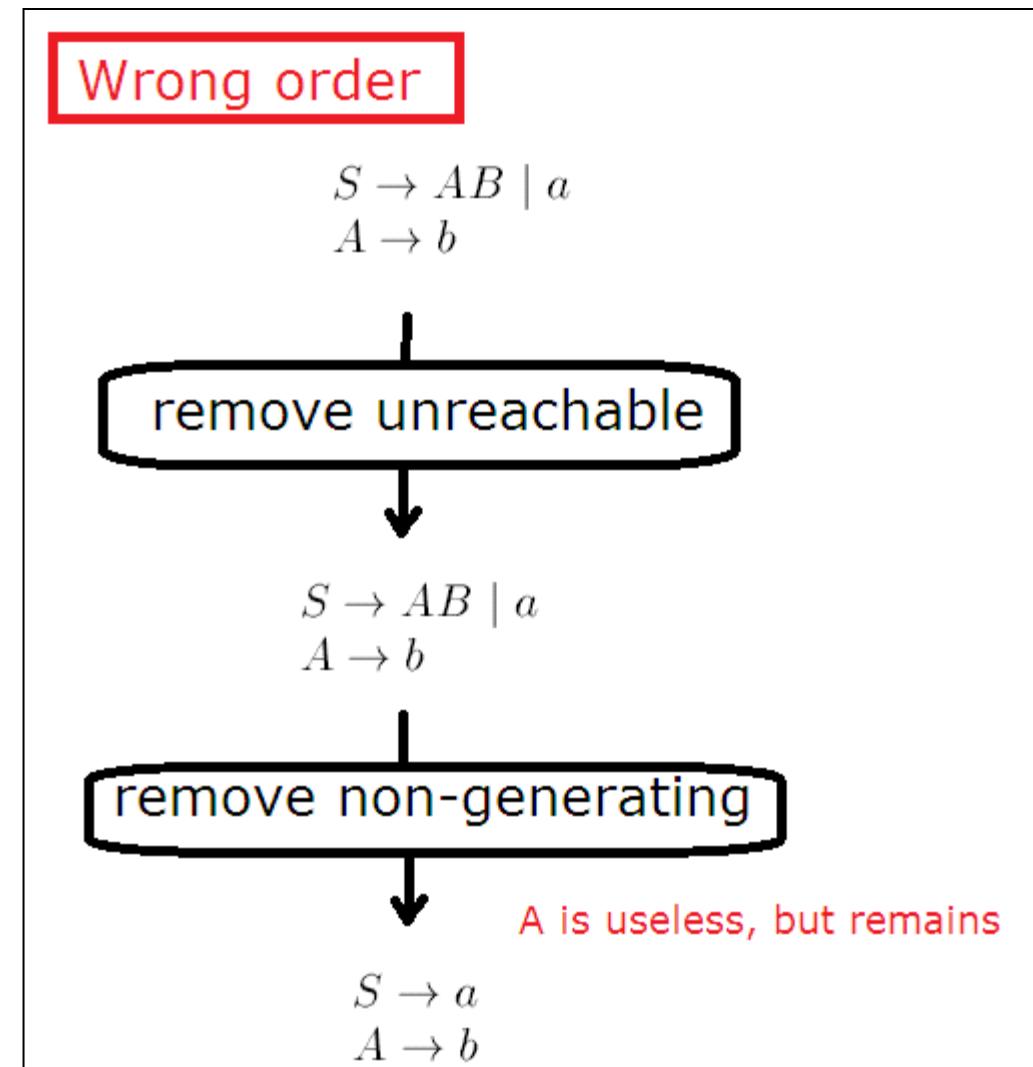
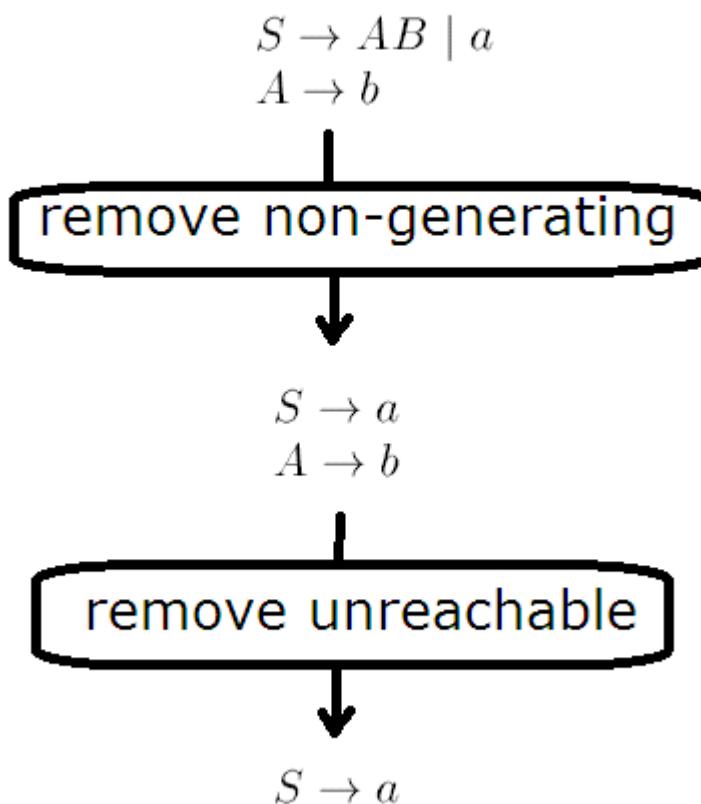
$$\begin{array}{l} S \rightarrow AB \mid a \\ A \rightarrow b \end{array}$$

- Find reachable.
- $\{S, A, B, a, b\}$

As per order, to remove useless



As per order, to remove useless



Eliminating ϵ -productions

- $A \rightarrow \epsilon$ is an ϵ -production
 - Let L be a CFL.
 - For $L - \{\epsilon\}$ there is a CFG which is without ϵ -productions

Eliminating ϵ -productions

- Discover *nullable* variables.
 - A variable A is nullable if $A \xrightarrow{*} \epsilon$
 - In this case, replace $B \rightarrow CAD$ by $B \rightarrow CAD|CD$

Eliminating ϵ -productions

- Discover *nullable* variables.
 - A variable A is nullable if $A \xrightarrow{*} \epsilon$
 - In this case, replace $B \rightarrow CAD$ by $B \rightarrow CAD|CD$
- **Why this correction is needed??**

Finding the nullable

Let $G = (V, T, P, S)$ be a CFG. We can find all the nullable symbols of G by the following iterative algorithm.

Finding the nullable

Let $G = (V, T, P, S)$ be a CFG. We can find all the nullable symbols of G by the following iterative algorithm.

BASIS: If $A \rightarrow \epsilon$ is a production of G , then A is nullable.

Finding the nullable

Let $G = (V, T, P, S)$ be a CFG. We can find all the nullable symbols of G by the following iterative algorithm.

BASIS: If $A \rightarrow \epsilon$ is a production of G , then A is nullable.

INDUCTION: If there is a production $B \rightarrow C_1 C_2 \cdots C_k$, where each C_i is nullable, then B is nullable. Note that each C_i must be a variable to be nullable, so we only have to consider productions with all-variable bodies.

Finding the nullable

Let $G = (V, T, P, S)$ be a CFG. We can find all the nullable symbols of G by the following iterative algorithm.

BASIS: If $A \rightarrow \epsilon$ is a production of G , then A is nullable.

INDUCTION: If there is a production $B \rightarrow C_1 C_2 \cdots C_k$, where each C_i is nullable, then B is nullable. Note that each C_i must be a variable to be nullable, so we only have to consider productions with all-variable bodies.

Theorem 7.7: In any grammar G , the only nullable symbols are the variables found by the algorithm above.

Example 7.8: Consider the grammar

$$\begin{aligned}S &\rightarrow AB \\A &\rightarrow aAA \mid \epsilon \\B &\rightarrow bBB \mid \epsilon\end{aligned}$$

- Find the nullable symbols.

Example 7.8: Consider the grammar

$$\begin{aligned}S &\rightarrow AB \\A &\rightarrow aAA \mid \epsilon \\B &\rightarrow bBB \mid \epsilon\end{aligned}$$

- Find the nullable symbols.
- {A,B,S}

Example 7.8: Consider the grammar

$$\begin{aligned}S &\rightarrow AB \\A &\rightarrow aAA \mid \epsilon \\B &\rightarrow bBB \mid \epsilon\end{aligned}$$

- Find the nullable symbols.
- {A,B,S}
- Apply the method.

Example 7.8: Consider the grammar

$$\begin{aligned}S &\rightarrow AB \\A &\rightarrow aAA \mid \epsilon \\B &\rightarrow bBB \mid \epsilon\end{aligned}$$

- Find the nullable symbols.
- {A,B,S}
- Apply the method.

$$\begin{aligned}S &\rightarrow AB \mid A \mid B \\A &\rightarrow aAA \mid aA \mid a \\B &\rightarrow bBB \mid bB \mid b\end{aligned}$$

Example 7.8: Consider the grammar

$$\begin{aligned}S &\rightarrow AB \\A &\rightarrow aAA \mid \epsilon \\B &\rightarrow bBB \mid \epsilon\end{aligned}$$

- Find the nullable symbols.
- {A,B,S}
- Apply the method.

$$\begin{aligned}S &\rightarrow AB \mid A \mid B \\A &\rightarrow aAA \mid aA \mid a \\B &\rightarrow bBB \mid bB \mid b\end{aligned}$$

- Any thing missing??

Example 7.8: Consider the grammar

$$\begin{aligned}S &\rightarrow AB \\A &\rightarrow aAA \mid \epsilon \\B &\rightarrow bBB \mid \epsilon\end{aligned}$$

- Find the nullable symbols.
- {A,B,S}
- Apply the method.

$$\begin{aligned}S &\rightarrow AB \mid A \mid B \\A &\rightarrow aAA \mid aA \mid a \\B &\rightarrow bBB \mid bB \mid b\end{aligned}$$

- Any thing missing?? ϵ is not in the new language.

Theorem 7.9: If the grammar G_1 is constructed from G by the above construction for eliminating ϵ -productions, then $L(G_1) = L(G) - \{\epsilon\}$.

Eliminating Unit Productions

- A unit production is of the form $A \rightarrow B$, where A and B are variables.

Eliminating Unit Productions

- A unit production is of the form $A \rightarrow B$, where A and B are variables.
- They may be useful, but we can construct an equivalent grammar without them.
- This simplifies the CFG.
 - Unit productions introduce extra steps into derivations that technically need not be there.
 - This complicates proving certain facts.

$$\begin{array}{lcl} I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ F & \rightarrow & I \mid (E) \\ T & \rightarrow & F \mid T * F \\ E & \rightarrow & T \mid E + T \end{array}$$

- In this $E \rightarrow T$ is a unit production.
- How to remove this?
- $E \rightarrow F \mid T * F \mid E + T$
- Still $E \rightarrow F$ is problematic
- Finally...

$$E \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \mid (E) \mid T * F \mid E + T$$

$$A \Rightarrow B_1 \Rightarrow B_2 \Rightarrow \cdots \Rightarrow B_n \Rightarrow \alpha$$

can be replaced by

$$A \rightarrow \alpha.$$

How to do this systematically?

- Unit pairs are identified.
- Along with this, the CFG is used to produce a new CFG which is without any unit production.

Unit pair

- (A, B) is a unit pair if $A \xrightarrow{*} B$
- Note, if the CFG have $A \rightarrow BC$ and $C \rightarrow \epsilon$
- then, (A, B) is a unit pair

BASIS: (A, A) is a unit pair for any variable A . That is, $A \xrightarrow{*} A$ by zero steps.

INDUCTION: Suppose we have determined that (A, B) is a unit pair, and $B \rightarrow C$ is a production, where C is a variable. Then (A, C) is a unit pair.

$$\begin{array}{lcl}
 I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\
 F & \rightarrow & I \mid (E) \\
 T & \rightarrow & F \mid T * F \\
 E & \rightarrow & T \mid E + T
 \end{array}$$

The basis gives us the unit pairs (E, E) , (T, T) , (F, F) , and (I, I) . For the inductive step, we can make the following inferences:

1. (E, E) and the production $E \rightarrow T$ gives us unit pair (E, T) .
2. (E, T) and the production $T \rightarrow F$ gives us unit pair (E, F) .
3. (E, F) and the production $F \rightarrow I$ gives us unit pair (E, I) .
4. (T, T) and the production $T \rightarrow F$ gives us unit pair (T, F) .
5. (T, F) and the production $F \rightarrow I$ gives us unit pair (T, I) .
6. (F, F) and the production $F \rightarrow I$ gives us unit pair (F, I) .

There are no more pairs that can be inferred, and in fact these ten pairs represent all the derivations that use nothing but unit productions. \square

To eliminate unit productions, we proceed as follows. Given a CFG $G = (V, T, P, S)$, construct CFG $G_1 = (V, T, P_1, S)$:

1. Find all the unit pairs of G .
2. For each unit pair (A, B) , add to P_1 all the productions $A \rightarrow \alpha$, where $B \rightarrow \alpha$ is a nonunit production in P . Note that $A = B$ is possible; in that way, P_1 contains all the nonunit productions in P .

Given CFG:

$$\begin{array}{lcl} I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ F & \rightarrow & I \mid (E) \\ T & \rightarrow & F \mid T * F \\ E & \rightarrow & T \mid E + T \end{array}$$

Non-unit productions are

$$\begin{array}{lcl} I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ F & \rightarrow & (E) \\ T & \rightarrow & T * F \\ E & \rightarrow & E + T \end{array}$$

Given CFG:

$$\begin{array}{lcl} I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ F & \rightarrow & I \mid (E) \\ T & \rightarrow & F \mid T * F \\ E & \rightarrow & T \mid E + T \end{array}$$

Unit Pairs

$$\begin{array}{l} \text{Pair} \\ \hline (E, E) \\ (E, T) \\ (E, F) \\ (E, I) \\ (T, T) \\ (T, F) \\ (T, I) \\ (F, F) \\ (F, I) \\ (I, I) \end{array}$$

Non-unit productions are

$$\begin{array}{lcl} I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ F & \rightarrow & (E) \\ T & \rightarrow & T * F \\ E & \rightarrow & E + T \end{array}$$

Given CFG:

$$\begin{array}{l} I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ F \rightarrow I \mid (E) \\ T \rightarrow F \mid T * F \\ E \rightarrow T \mid E + T \end{array}$$

Unit Pairs

Pair	Productions
(E, E)	$E \rightarrow E + T$
(E, T)	$E \rightarrow T * F$
(E, F)	$E \rightarrow (E)$
(E, I)	$E \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
(T, T)	$T \rightarrow T * F$
(T, F)	$T \rightarrow (E)$
(T, I)	$T \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
(F, F)	$F \rightarrow (E)$
(F, I)	$F \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
(I, I)	$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

Non-unit productions are

$$\begin{array}{l} I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ F \rightarrow (E) \\ T \rightarrow T * F \\ E \rightarrow E + T \end{array}$$

Given CFG:

$$\begin{array}{l}
 I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\
 F \rightarrow I \mid (E) \\
 T \rightarrow F \mid T * F \\
 E \rightarrow T \mid E + T
 \end{array}$$

Unit Pairs

Pair	Productions
(E, E)	$E \rightarrow E + T$
(E, T)	$E \rightarrow T * F$
(E, F)	$E \rightarrow (E)$
(E, I)	$E \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
(T, T)	$T \rightarrow T * F$
(T, F)	$T \rightarrow (E)$
(T, I)	$T \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
(F, F)	$F \rightarrow (E)$
(F, I)	$F \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
(I, I)	$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

Non-unit productions are

$$\begin{array}{l}
 I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\
 F \rightarrow (E) \\
 T \rightarrow T * F \\
 E \rightarrow E + T
 \end{array}$$

CFG without unit productions

$$\begin{array}{l}
 E \rightarrow E + T \mid T * F \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\
 T \rightarrow T * F \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\
 F \rightarrow (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\
 I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1
 \end{array}$$

The order in which these preprocessing steps can occur

1. Eliminate ϵ -productions.
2. Eliminate unit productions.
3. Eliminate useless symbols.

* **Exercise 7.1.1:** Find a grammar equivalent to

$$\begin{array}{lcl} S & \rightarrow & AB \mid CA \\ A & \rightarrow & a \\ B & \rightarrow & BC \mid AB \\ C & \rightarrow & aB \mid b \end{array}$$

with no useless symbols.

* **Exercise 7.1.1:** Find a grammar equivalent to

$$\begin{array}{lcl} S & \rightarrow & AB \mid CA \\ A & \rightarrow & a \\ B & \rightarrow & BC \mid AB \\ C & \rightarrow & aB \mid b \end{array}$$

with no useless symbols.

- Eliminate non-generating, first.

* **Exercise 7.1.1:** Find a grammar equivalent to

$$\begin{array}{lcl} S & \rightarrow & AB \mid CA \\ A & \rightarrow & a \\ B & \rightarrow & BC \mid AB \\ C & \rightarrow & aB \mid b \end{array}$$

with no useless symbols.

- Eliminate non-generating, first.
- Generating symbols = $\{a, b, A, C, S\}$
- Non-generating symbol is B .

* Exercise 7.1.1: Find a grammar equivalent to

$$\begin{array}{lcl} S & \rightarrow & AB \mid CA \\ A & \rightarrow & a \\ B & \rightarrow & BC \mid AB \\ C & \rightarrow & aB \mid b \end{array}$$

with no useless symbols.

- Eliminate non-generating, first.
- Generating symbols = $\{a, b, A, C, S\}$
- Non-generating symbol is B .
- We get $S \rightarrow CA$, $A \rightarrow a$, $C \rightarrow b$

* Exercise 7.1.1: Find a grammar equivalent to

$$\begin{array}{lcl} S & \rightarrow & AB \mid CA \\ A & \rightarrow & a \\ B & \rightarrow & BC \mid AB \\ C & \rightarrow & aB \mid b \end{array}$$

with no useless symbols.

- Eliminate non-generating, first.
- Generating symbols = $\{a, b, A, C, S\}$
- Non-generating symbol is B .
- We get $S \rightarrow CA$, $A \rightarrow a$, $C \rightarrow b$
- Remove unreachable.

* Exercise 7.1.1: Find a grammar equivalent to

$$\begin{array}{lcl} S & \rightarrow & AB \mid CA \\ A & \rightarrow & a \\ B & \rightarrow & BC \mid AB \\ C & \rightarrow & aB \mid b \end{array}$$

with no useless symbols.

- Eliminate non-generating, first.
- Generating symbols = $\{a, b, A, C, S\}$
- Non-generating symbol is B .
- We get $S \rightarrow CA$, $A \rightarrow a$, $C \rightarrow b$
- Remove unreachable.
- All are reachable. So, $S \rightarrow CA$, $A \rightarrow a$, $C \rightarrow b$ is the answer.

Chomsky Normal Form

- For a CFL without ϵ , where all productions are of the form $A \rightarrow BC, A \rightarrow a$.
- After preprocessing steps, it is quite easy to get in to CNF.

- $A \rightarrow BCDE$ can be replaced by
 $A \rightarrow BF, F \rightarrow CDE.$

- $A \rightarrow BCDE$ can be replaced by
 $A \rightarrow BF, F \rightarrow CDE.$
- Then $F \rightarrow CDE$ can be replaced by $F \rightarrow CG$
and $G \rightarrow DE$

- $A \rightarrow BCDE$ can be replaced by
 $A \rightarrow BF, F \rightarrow CDE.$
- Then $F \rightarrow CDE$ can be replaced by $F \rightarrow CG$ and $G \rightarrow DE$
- So, $A \rightarrow BCDE$ can be replaced by $A \rightarrow BF,$ $F \rightarrow CG$ and $G \rightarrow DE$

- Similarly, $A \rightarrow BabC$ can be replaced by $A \rightarrow BDEC, D \rightarrow a, E \rightarrow b.$
- Then $A \rightarrow BDEC$ can be replaced by ...

* **Exercise 7.1.2:** Begin with the grammar:

$$\begin{array}{lcl} S & \rightarrow & ASB \mid \epsilon \\ A & \rightarrow & aAS \mid a \\ B & \rightarrow & SbS \mid A \mid bb \end{array}$$

- a) Eliminate ϵ -productions.
- b) Eliminate any unit productions in the resulting grammar.
- c) Eliminate any useless symbols in the resulting grammar.
- d) Put the resulting grammar into Chomsky Normal Form.

- Theorem: Let G be a CFG in CNF form, then for any $w \in L(G)$, w can be derived in exactly $(2|w| - 1)$ steps.

- Theorem: Let G be a CFG in CNF form, then for any $w \in L(G)$, w can be derived in exactly $(2|w| - 1)$ steps.
- Proof: Parse tree when the CFG is in CNF form is a binary tree (except for a small technical issue with $A \rightarrow a$)

- Theorem: Let G be a CFG in CNF form, then for any $w \in L(G)$, w can be derived in exactly $(2|w| - 1)$ steps.
- Proof: Parse tree when the CFG is in CNF form is a binary tree (except for a small technical issue with $A \rightarrow a$)
- Let n_0 be nodes with 0 children (leaves)
- Let n_1 be nodes with 1 child
- Let n_2 be nodes with 2 children

- Theorem: Let G be a CFG in CNF form, then for any $w \in L(G)$, w can be derived in exactly $(2|w| - 1)$ steps.
- Proof: Parse tree when the CFG is in CNF form is a binary tree (except for a small technical issue with $A \rightarrow a$)
- Let n_0 be nodes with 0 children (leaves)
- Let n_1 be nodes with 1 child
- Let n_2 be nodes with 2 children
- We have $n_0 = n_2 + 1$ (can you prove this?)

- We have, $n_0 = |w|$
- $n_0 = n_1$ (why?)
- Number of steps
 - $= n_2 + n_1$
 - $= n_0 + n_0 - 1$
 - $= 2n_0 - 1$
 - $= 2|w| - 1$

Pumping Lemma for CFL

Intuition

- Recall the pumping lemma for regular languages.
- It told us that if there was a string long enough to cause a cycle in the DFA for the language, then we could “pump” the cycle and discover an infinite sequence of strings that had to be in the language.

Intuition

- For CFL's the situation is a little more complicated.
- We can always find **two** pieces of any sufficiently long string to “pump” in tandem.
 - **That is:** if we repeat each of the two pieces the same number of times, we get another string in the language.

The size of Parse Trees

Theorem 7.17: Suppose we have a parse tree according to a Chomsky-Normal-Form grammar $G = (V, T, P, S)$, and suppose that the yield of the tree is a terminal string w . If the length of the longest path is n , then $|w| \leq 2^{n-1}$.

PROOF: The proof is a simple induction on n .

The size of Parse Trees

Theorem 7.17: Suppose we have a parse tree according to a Chomsky-Normal-Form grammar $G = (V, T, P, S)$, and suppose that the yield of the tree is a terminal string w . If the length of the longest path is n , then $|w| \leq 2^{n-1}$.

PROOF: The proof is a simple induction on n .

- If $|w| > 2^{n-1}$, then the longest path is $> n$

The size of Parse Trees

Theorem 7.17: Suppose we have a parse tree according to a Chomsky-Normal-Form grammar $G = (V, T, P, S)$, and suppose that the yield of the tree is a terminal string w . If the length of the longest path is n , then $|w| \leq 2^{n-1}$.

PROOF: The proof is a simple induction on n .

- If $|w| > 2^{n-1}$, then the longest path is $> n$
- Let $|V| = m$

The size of Parse Trees

Theorem 7.17: Suppose we have a parse tree according to a Chomsky-Normal-Form grammar $G = (V, T, P, S)$, and suppose that the yield of the tree is a terminal string w . If the length of the longest path is n , then $|w| \leq 2^{n-1}$.

PROOF: The proof is a simple induction on n .

- If $|w| > 2^{n-1}$, then the longest path is $> n$
- Let $|V| = m$
- For $|w| = 2^m$, longest path is $> m + 1$

The size of Parse Trees

Theorem 7.17: Suppose we have a parse tree according to a Chomsky-Normal-Form grammar $G = (V, T, P, S)$, and suppose that the yield of the tree is a terminal string w . If the length of the longest path is n , then $|w| \leq 2^{n-1}$.

PROOF: The proof is a simple induction on n .

- If $|w| > 2^{n-1}$, then the longest path is $> n$
- Let $|V| = m$
- For $|w| = 2^m$, longest path is $> m + 1$
- In that longest path a variable must have been repeated (since we have only m variables).

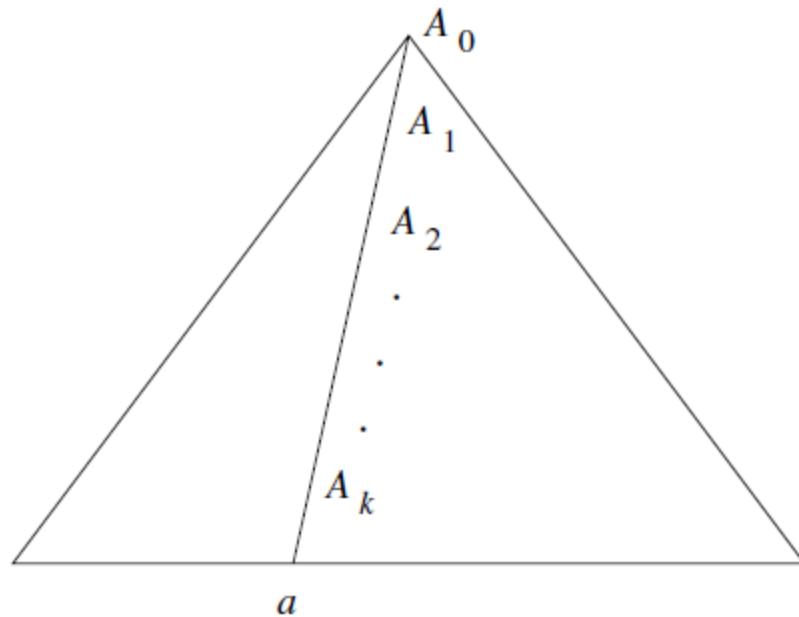


Figure 7.5: Every sufficiently long string in L must have a long path in its parse tree

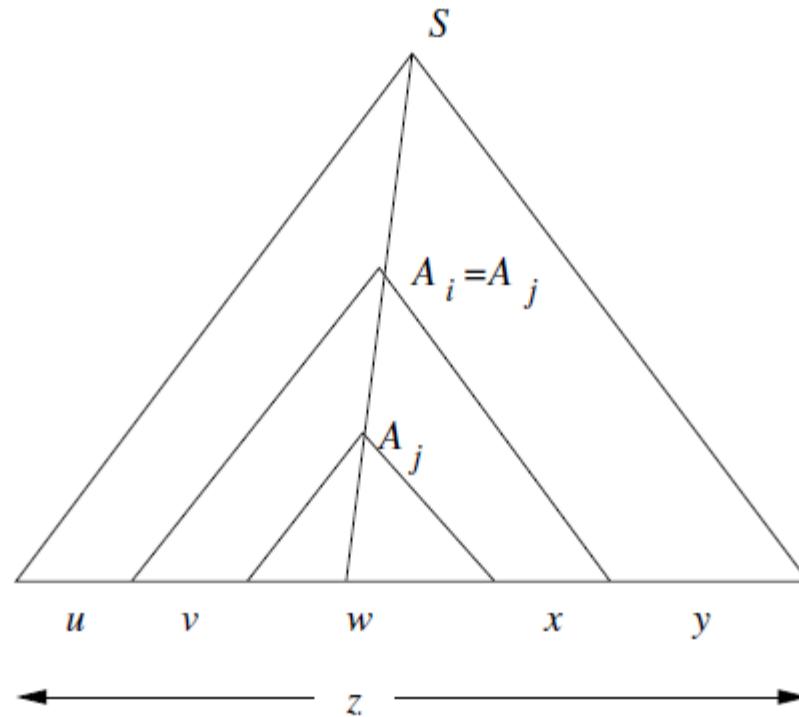
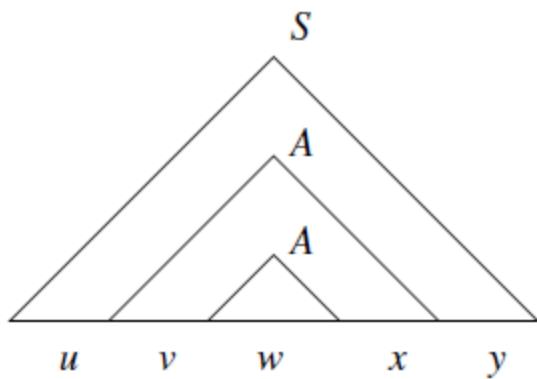
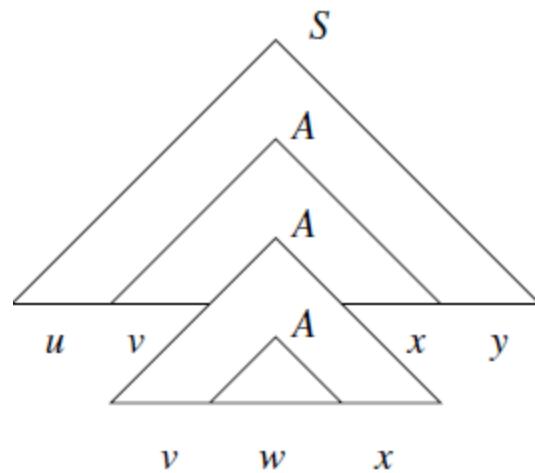
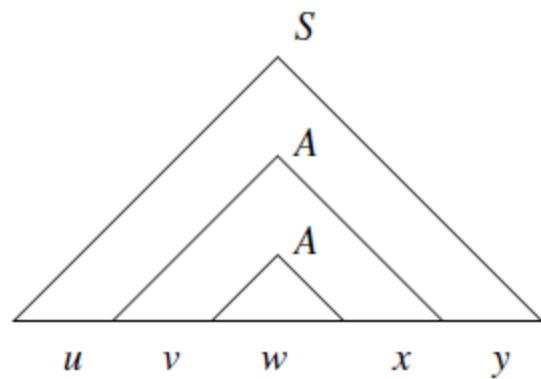
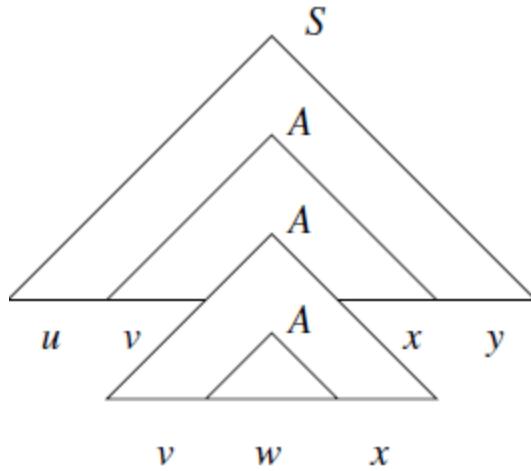
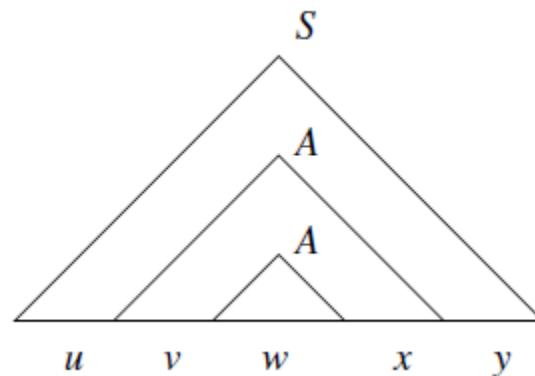
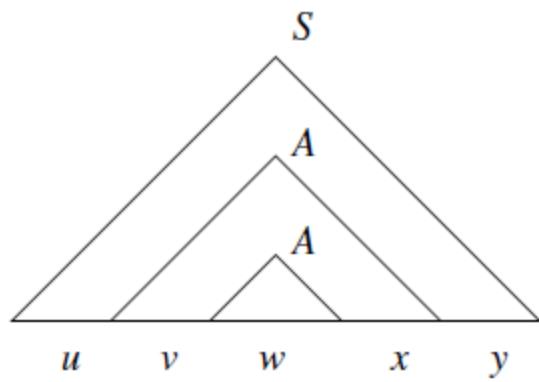
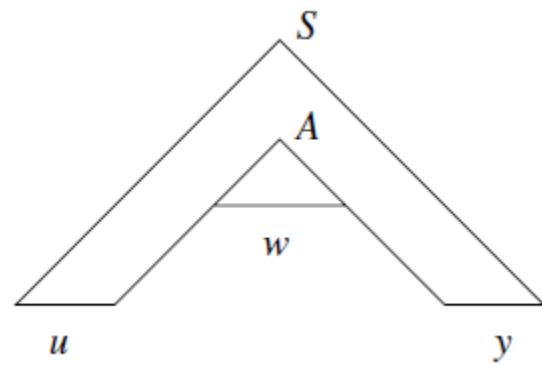
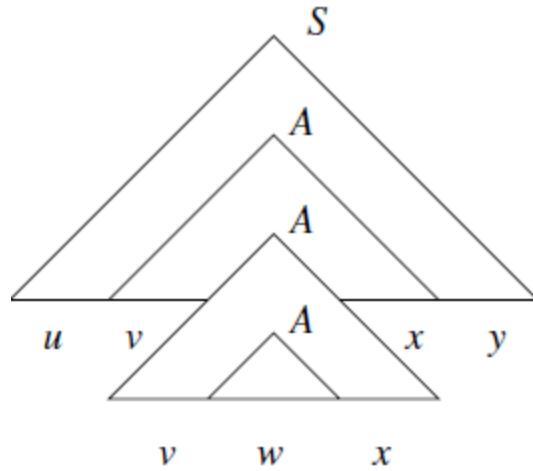
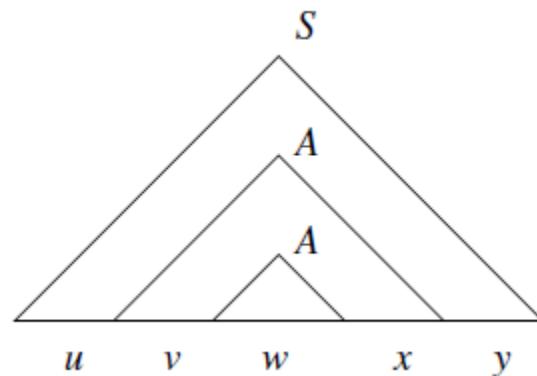
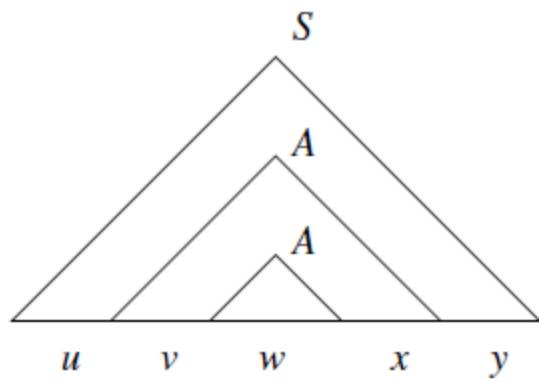


Figure 7.6: Dividing the string w so it can be pumped

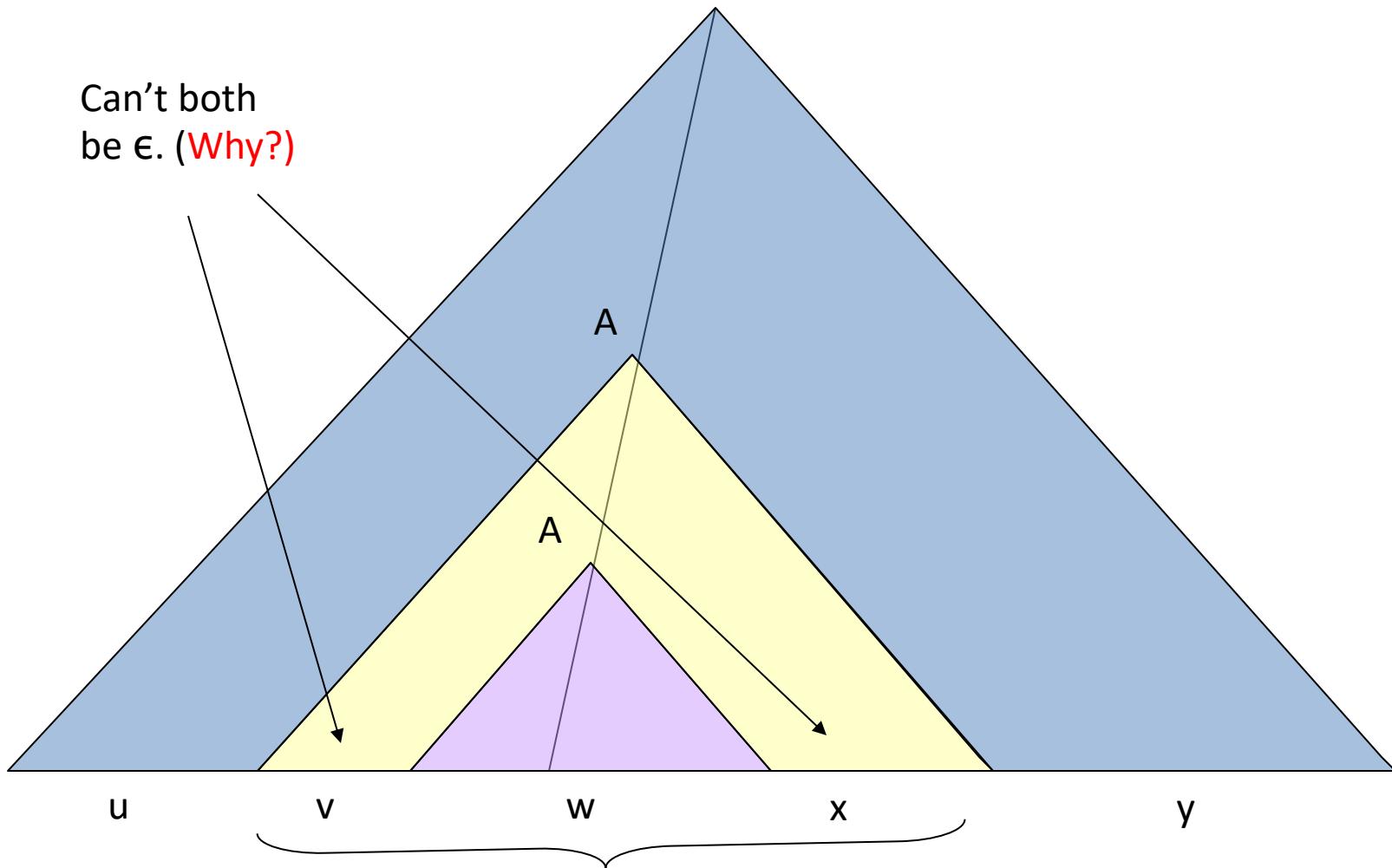




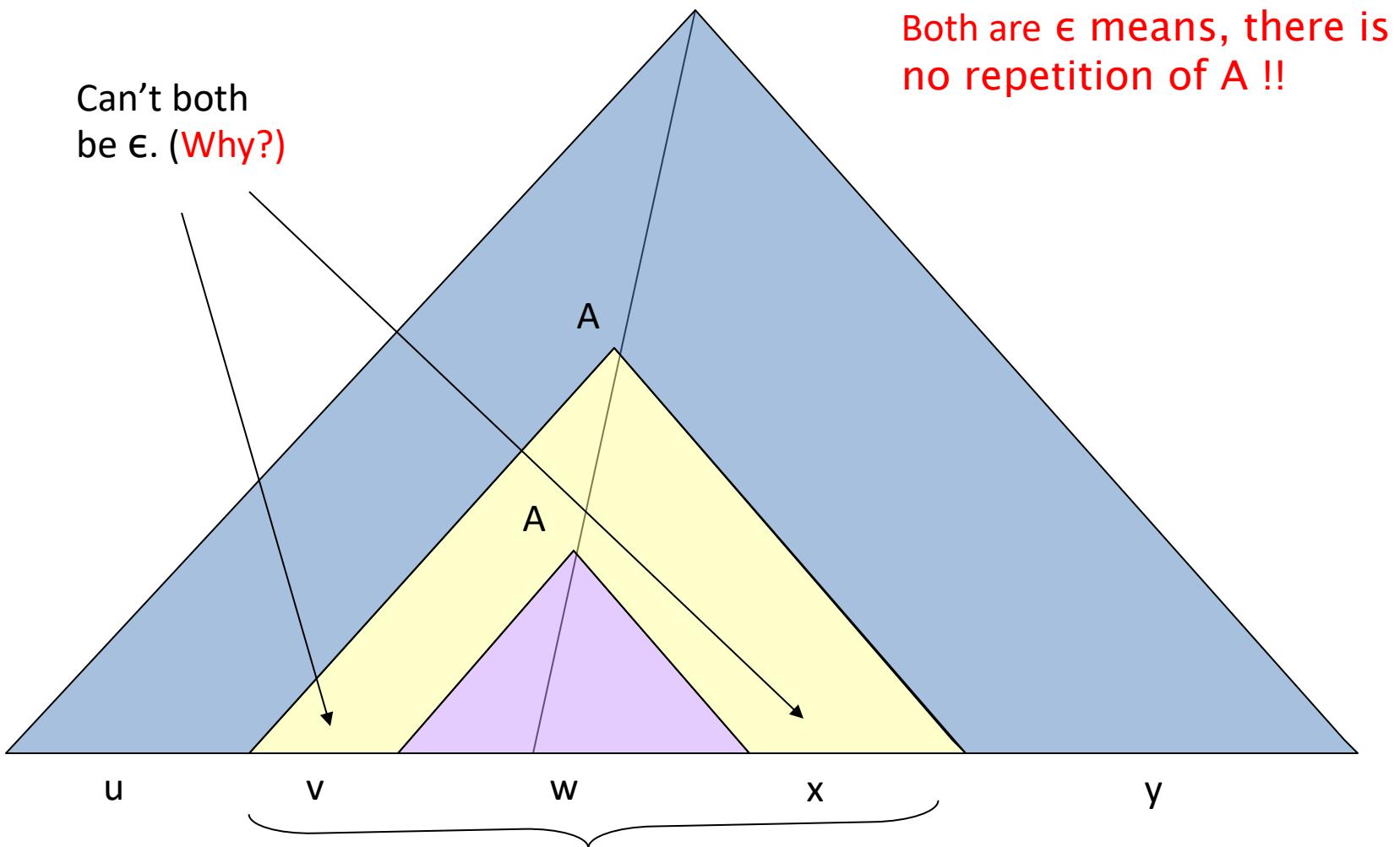




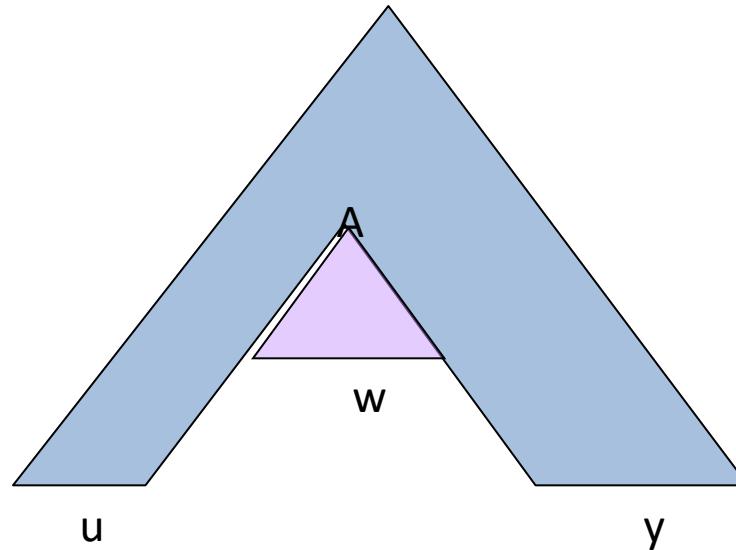
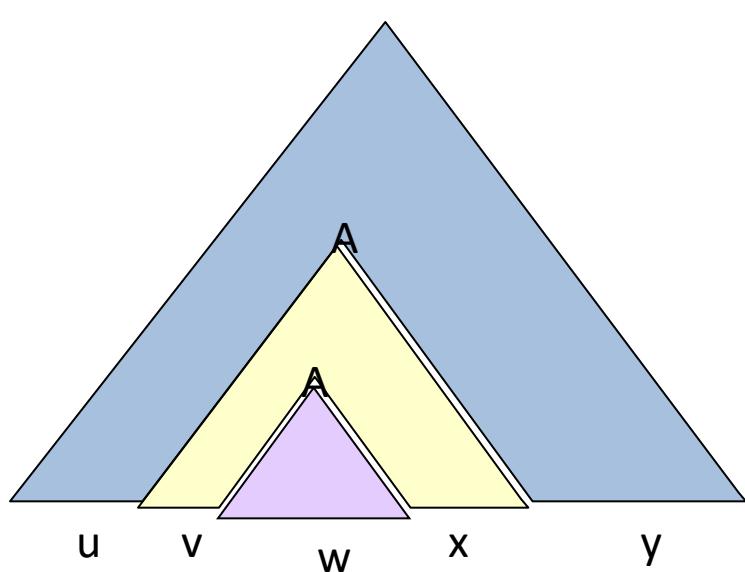
Parse Tree in the Pumping-Lemma



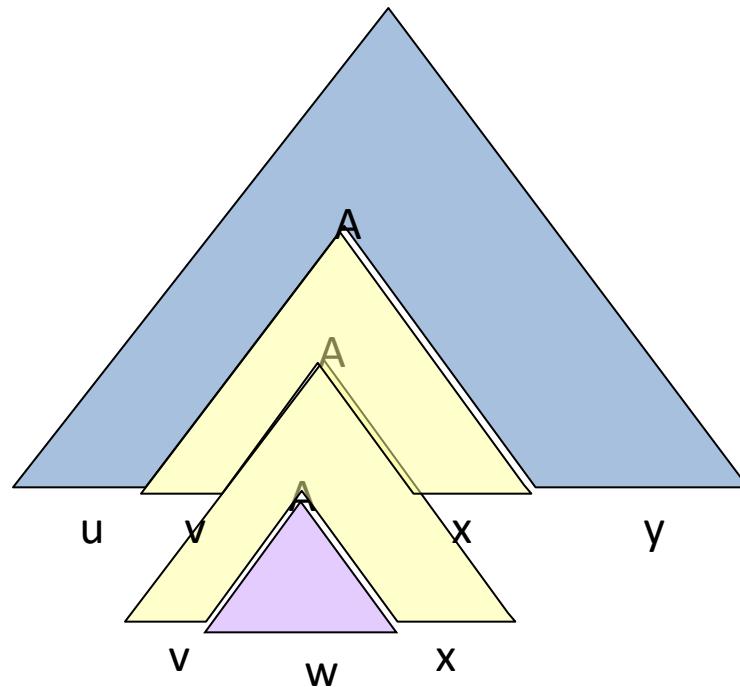
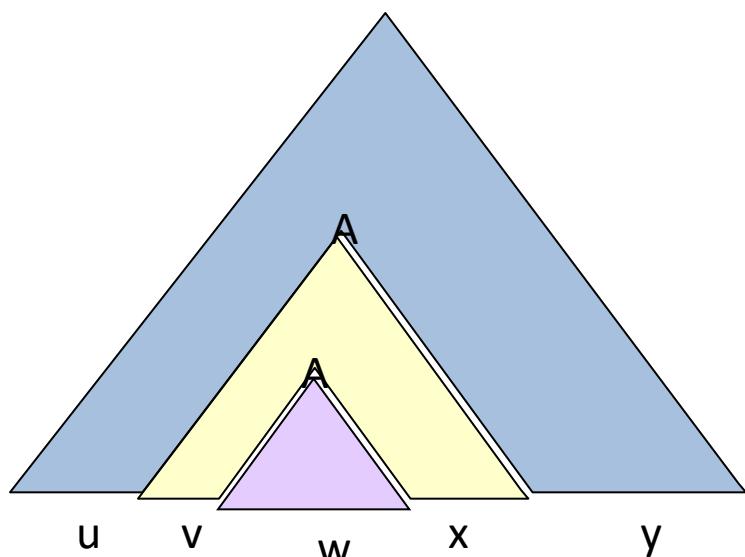
Parse Tree in the Pumping-Lemma



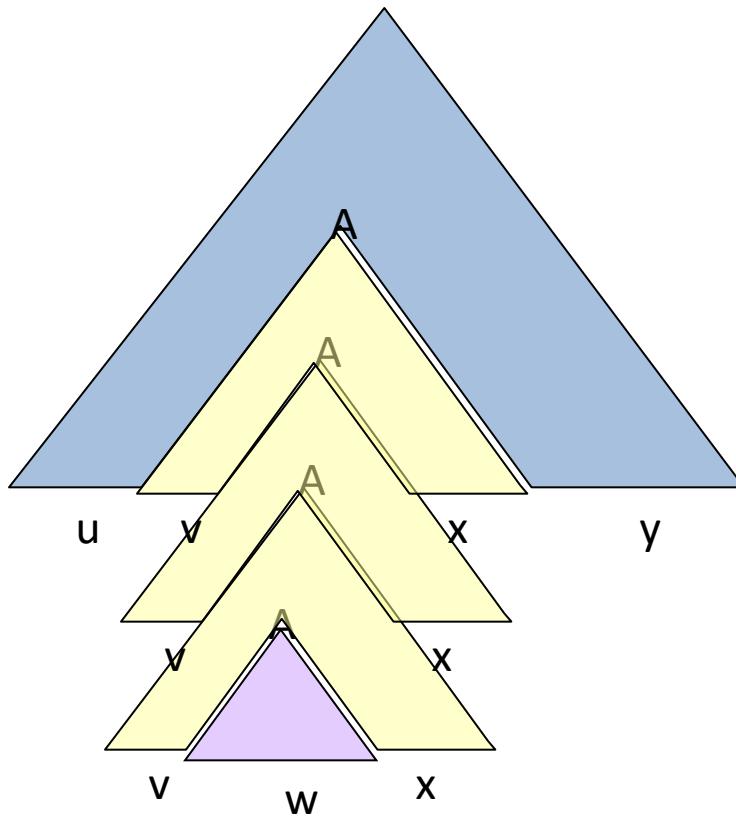
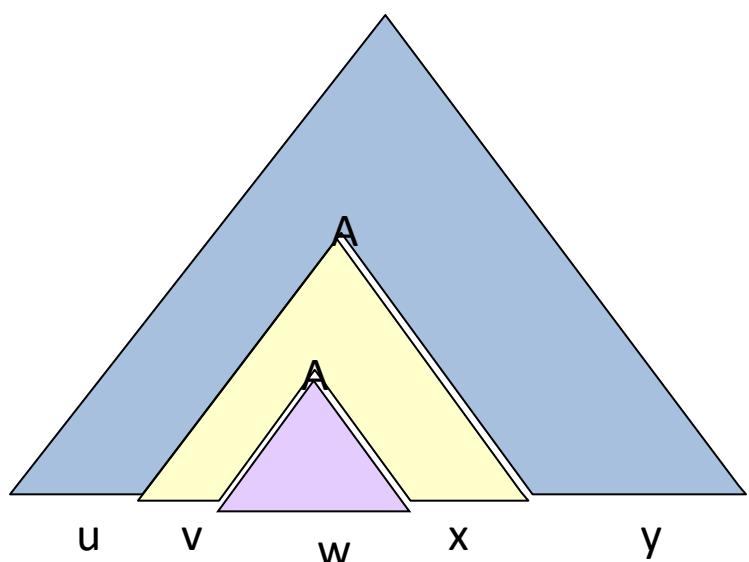
Pump Zero Times



Pump Twice



Pump Thrice Etc., Etc.



Statement

Theorem 7.18: (The pumping lemma for context-free languages) Let L be a CFL. Then there exists a constant n such that if z is any string in L such that $|z|$ is at least n , then we can write $z = uvwxy$, subject to the following conditions:

1. $|vwx| \leq n$. That is, the middle portion is not too long.
2. $vx \neq \epsilon$. Since v and x are the pieces to be “pumped,” this condition says that at least one of the strings we pump must not be empty.
3. For all $i \geq 0$, $uv^iwx^i y$ is in L . That is, the two strings v and x may be “pumped” any number of times, including 0, and the resulting string will still be a member of L .

Statement

For every context-free language L

There is an integer n , such that

For every string z in L of length $\geq n$

There exists $z = uvwxy$ such that:

1. $|vwx| \leq n$.
2. $|vx| > 0$.
3. For all $i \geq 0$, $uv^iwx^i y$ is in L .

Example 7.19 : Let L be the language $\{0^n1^n2^n \mid n \geq 1\}$. That is, L consists of all strings in $0^+1^+2^+$ with an equal number of each symbol, e.g., 012, 001122, and so on. Suppose L were context-free. Then there is an integer n given to us by the pumping lemma. Let us pick $z = 0^n1^n2^n$.

Example 7.19 : Let L be the language $\{0^n1^n2^n \mid n \geq 1\}$. That is, L consists of all strings in $0^+1^+2^+$ with an equal number of each symbol, e.g., 012, 001122, and so on. Suppose L were context-free. Then there is an integer n given to us by the pumping lemma. Let us pick $z = 0^n1^n2^n$.

We can write $z = uvwxy$, where $|vwx| \leq n$ and v and x are not both ϵ .

Example 7.19 : Let L be the language $\{0^n 1^n 2^n \mid n \geq 1\}$. That is, L consists of all strings in $0^+ 1^+ 2^+$ with an equal number of each symbol, e.g., 012, 001122, and so on. Suppose L were context-free. Then there is an integer n given to us by the pumping lemma. Let us pick $z = 0^n 1^n 2^n$.

We can write $z = uvwxy$, where $|vwx| \leq n$ and v and x are not both ϵ .

Then we know that vwx cannot involve both 0's and 2's, since the last 0 and the first 2 are separated by $n + 1$ positions.

Example 7.19 : Let L be the language $\{0^n 1^n 2^n \mid n \geq 1\}$. That is, L consists of all strings in $0^+ 1^+ 2^+$ with an equal number of each symbol, e.g., 012, 001122, and so on. Suppose L were context-free. Then there is an integer n given to us by the pumping lemma. Let us pick $z = 0^n 1^n 2^n$.

We can write $z = uvwxy$, where $|vwx| \leq n$ and v and x are not both ϵ .

- There can be 5 cases where vwx is having
 - Only 0s
 - Some 0s and some 1s
 - Only 1s
 - Some 1s and some 2s
 - Only 2s.
- In all these 5 cases, $uv^2wx^2y \notin L$.

Example 7.21 : Let $L = \{ww \mid w \text{ is in } \{0, 1\}^*\}$.

- Show L is not a CFL.
- Note, $\{ww^R \mid w \in \{0, 1\}^*\}$ is a CFL.
- How can you prove this??

Example 7.21 : Let $L = \{ww \mid w \text{ is in } \{0, 1\}^*\}$.

- Let pumping length is n .
- Let the string be $z = 0^n 1^n 0^n 1^n$
- z can be written as $uvwxy$, such that $|vwx| \leq n$ and $vx \neq \epsilon$
- There are 7 cases, based on where vwx can occur in z .
- In all these cases it can be shown that uwy is not in L .

Using the Pumping Lemma

- $\{0^i 1 0^i \mid i \geq 1\}$ is a CFL.
 - We can match one pair of counts.
 - Can you give CFG??

Using the Pumping Lemma

- $\{0^i 1 0^i \mid i \geq 1\}$ is a CFL.
- But $L = \{0^i 1 0^i 1 0^i \mid i \geq 1\}$ is not.
 - We can't match two pairs, or three counts as a group.
- Proof using the pumping lemma.
- Suppose L were a CFL.
- Let n be L 's pumping-lemma constant.

Using the Pumping Lemma

- Consider $z = 0^n 1 0^n 1 0^n$.
- We can write $z = uvwxy$, where $|vwx| \leq n$, and $|vx| \geq 1$.
- **Case 1:** vx has no 0's.
 - Then at least one of them is a 1, and uw^kx has at most one 1, which no string in L does.

Using the Pumping Lemma

- Still considering $z = 0^n10^n10^n$.
- **Case 2:** vx has at least one 0.
 - vwx is too short ($\text{length } \leq n$) to extend to all three blocks of 0's in $0^n10^n10^n$.
 - Thus, uwy has at least one block of n 0's, and at least one block with fewer than n 0's.
 - Thus, uwy is not in L .