Regular Expressions and Languages

CHAPTER 3 (PART 1)

RE's: Introduction

- *Regular expressions* are an algebraic way to describe languages.
- They describe exactly the regular languages.
- If E is a regular expression, then L(E) is the language it defines.
- We'll describe RE's and their languages recursively.

Operators of RE

1. The *union* of two languages L and M, denoted $L \cup M$, is the set of strings that are in either L or M, or both. For example, if $L = \{001, 10, 111\}$ and $M = \{\epsilon, 001\}$, then $L \cup M = \{\epsilon, 10, 001, 111\}$.

2. The *concatenation* of languages L and M is the set of strings that can be formed by taking any string in L and concatenating it with any string in M. For example, if $L = \{001, 10, 111\}$ and $M = \{\epsilon, 001\}$, then $L.M$, or just LM, is $\{001, 10, 111, 001001, 10001, 111001\}$.

Contd…

3. The *closure* (or *star*, or *Kleene closure*) of a language L is denoted L^* and represents the set of those strings that can be formed by taking any number of strings from L, possibly with repetitions (i.e., the same string may be selected more than once) and concatenating all of them. For instance, if $L = \{0, 1\}$, then L^* is all strings of 0's and 1's. If $L = \{0, 11\}$, then L^* consists of those strings of 0's and 1's such that the 1's come in pairs, e.g., 011, 11110, and ϵ, but not 01011 or 101. More formally, L^* is the infinite union $\cup_{i \geq 0} L^i$, where $L^0 = \{\epsilon\}$, $L^1 = L$, and L^i, for $i > 1$ is $LL \cdots L$ (the concatenation of i copies of L).

See Exercise 3.1 for better understanding.

RE's: Definition

- **Basis 1:** If *a* is any symbol, then **a** is a RE, and L(**a**) = {a}.
  - Note: {a} is the language containing one string, and that string is of length 1.
- **Basis 2:** ε is a RE, and L(ε) = {ε}.
- **Basis 3:** ∅ is a RE, and L(∅) = ∅.

RE's: Definition – Contd…

- **Induction 1:** If E_1 and E_2 are regular expressions, then E_1+E_2 is a regular expression, and $L(E_1+E_2) = L(E_1) \cup L(E_2)$.
- **Induction 2:** If E_1 and E_2 are regular expressions, then E_1E_2 is a regular expression, and $L(E_1E_2) = L(E_1)L(E_2)$.

Concatenation : the set of strings wx such that w is in $L(E_1)$ and x is in $L(E_2)$.

RE's: Definition – (3)

- Induction 3: If E is a RE, then E* is a RE, and L(E*) = (L(E))*.

Closure, or "Kleene closure" = set of strings $w_1w_2\ldots w_n$, for some $n \geq 0$, where each w_i is in L(E).
Note: when n=0, the string is ε.

Precedence of Operators

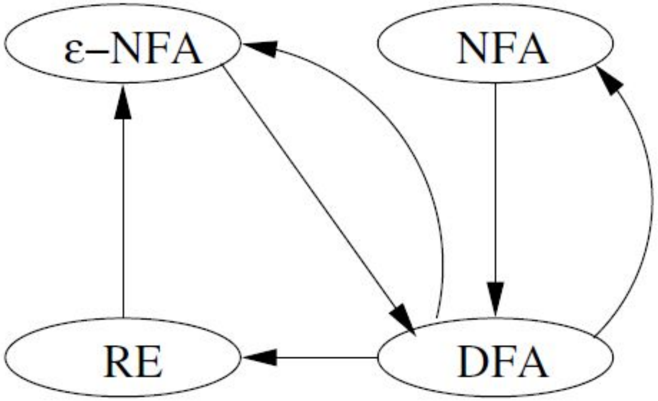
- Parentheses may be used wherever needed to influence the grouping of operators.
- Order of precedence is * (highest), then concatenation, then + (lowest).
- **Example:** 01*+1
  - (0(1*))+1
  - (01)*+1
  - 0(1*+1)

Examples: RE's

- L(**01**) = {01}.
- L(**01**+**0**) = {01, 0}.
- L(**0**(**1**+**0**)) = {01, 00}.
  - Note order of precedence of operators.
- L(**0***) = {ε, 0, 00, 000,… }.
- L((**0**+**10**)*(ε+**1**)) = all strings of 0's and 1's without two consecutive 1's.
- **Example 3.2 (in textbook):** Write a RE for the set of strings that consists of alternating 0's and 1's.

Equivalence of RE's and Automata

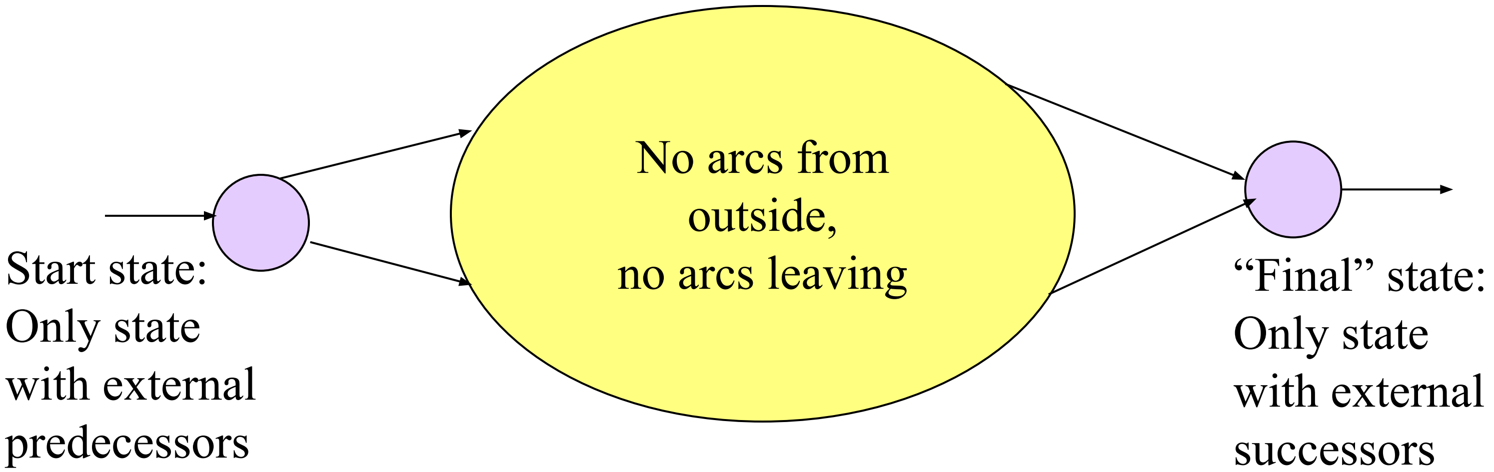
- We need to show that for every RE, there is an automaton that accepts the same language.
  - Pick the most powerful automaton type: the ε-NFA.
- And we need to show that for every automaton, there is a RE defining its language.
  - Pick the most restrictive type: the DFA.



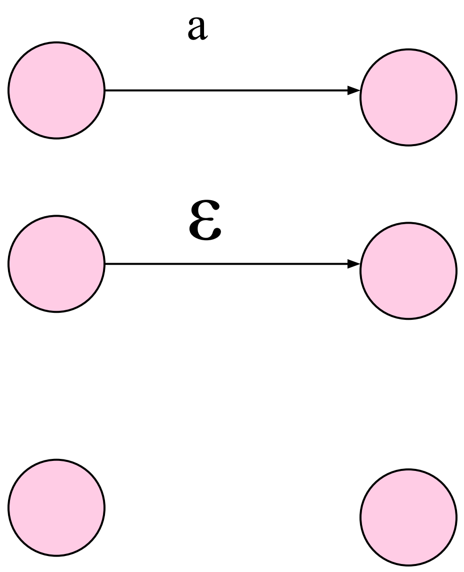
Converting a RE to an ε-NFA

- Proof is an induction on the number of operators (+, concatenation, *) in the RE.
- We always construct an automaton of a special form (next slide).

Form of ε-NFA's Constructed

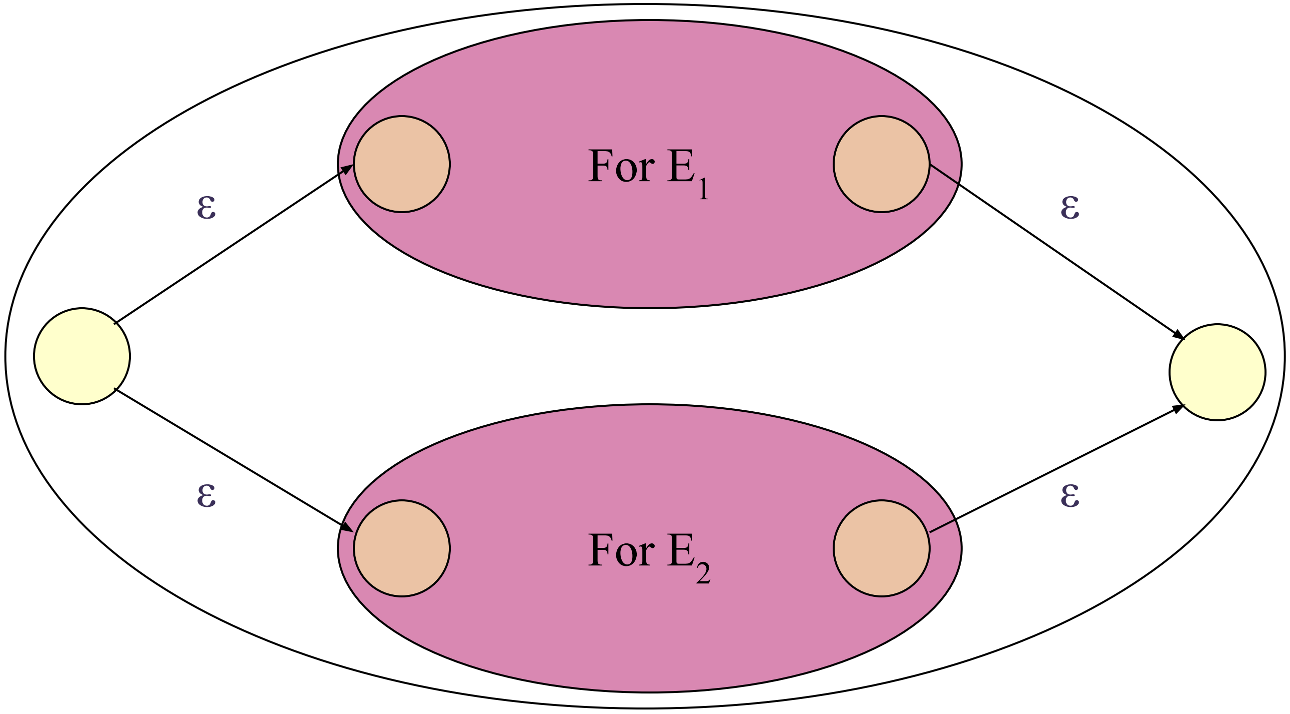


RE to ε-NFA: Basis



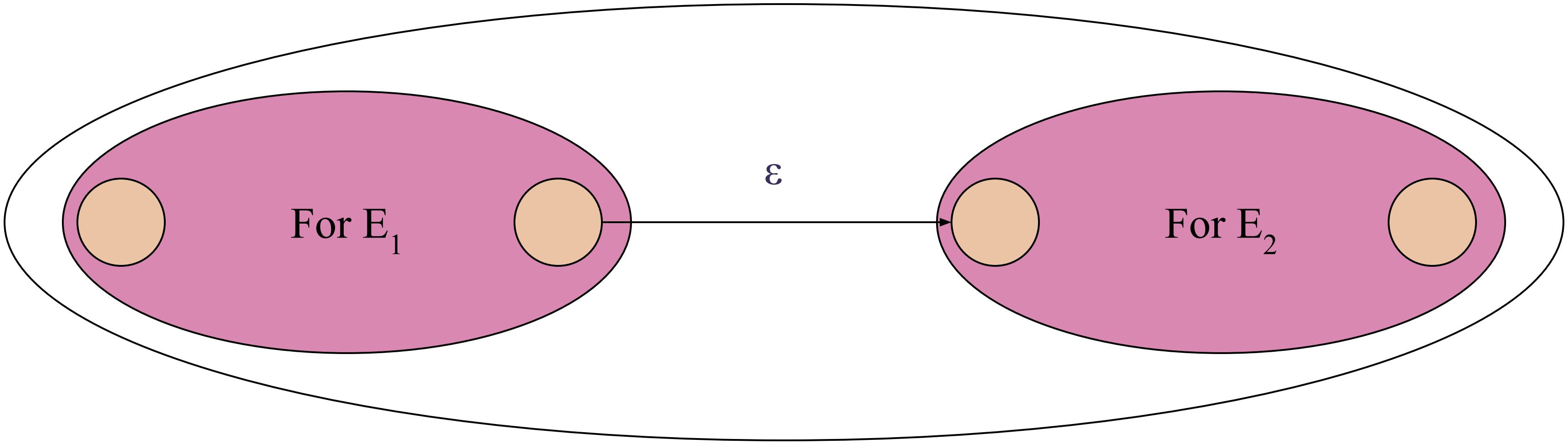
- Symbol **a**:
- ε:
- ∅:

RE to ε-NFA: Induction 1 – Union



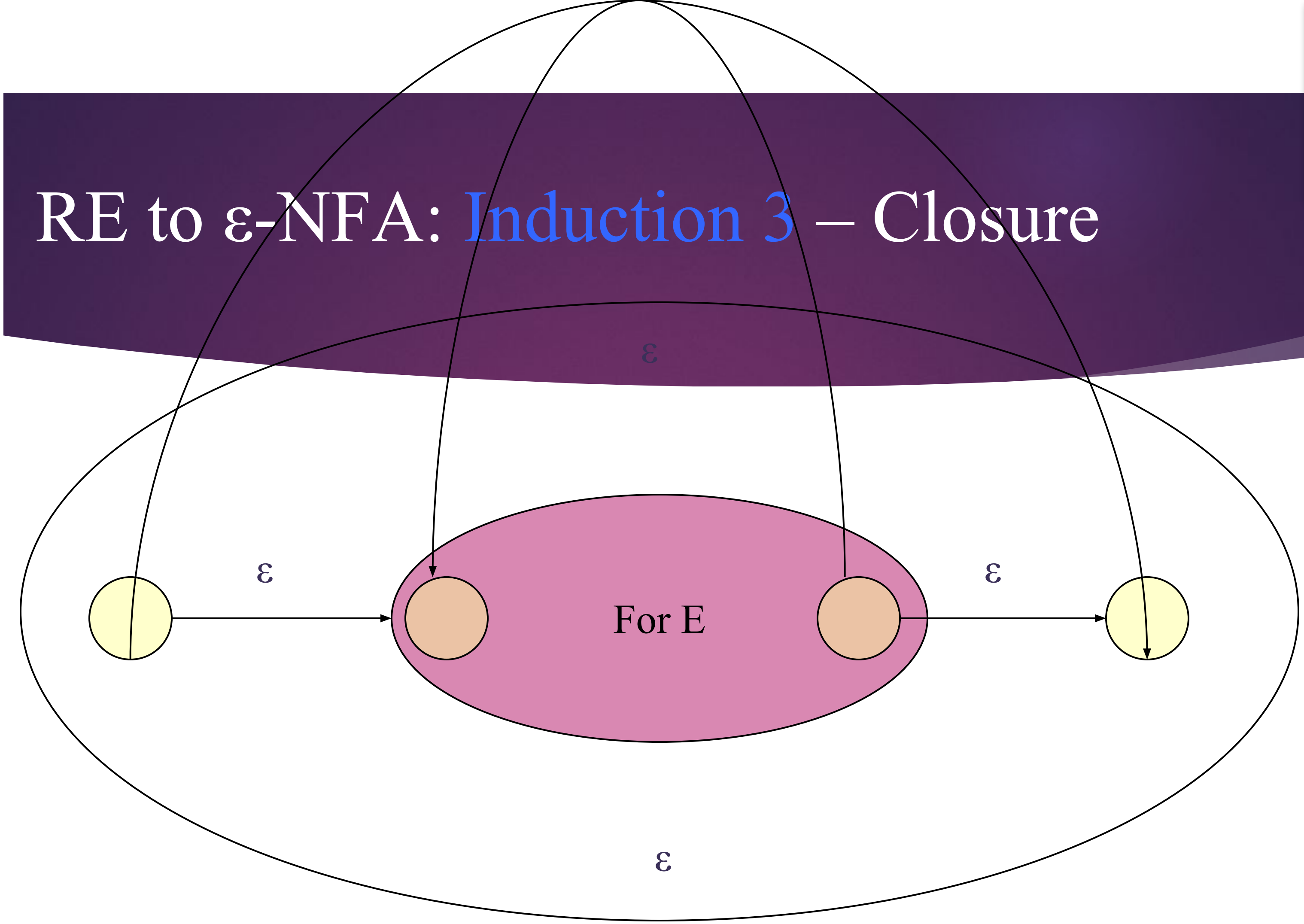
For $E_1 \cup E_2$

RE to ε-NFA: Induction 2 – Concatenation



For E_1E_2

RE to ε-NFA: Induction 3 – Closure

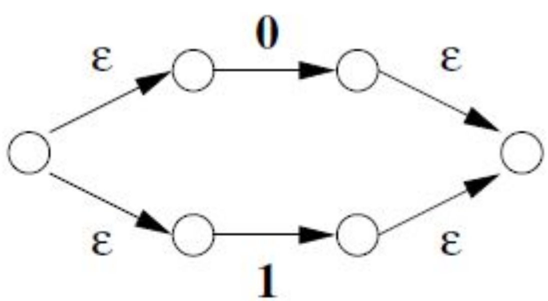


For E*

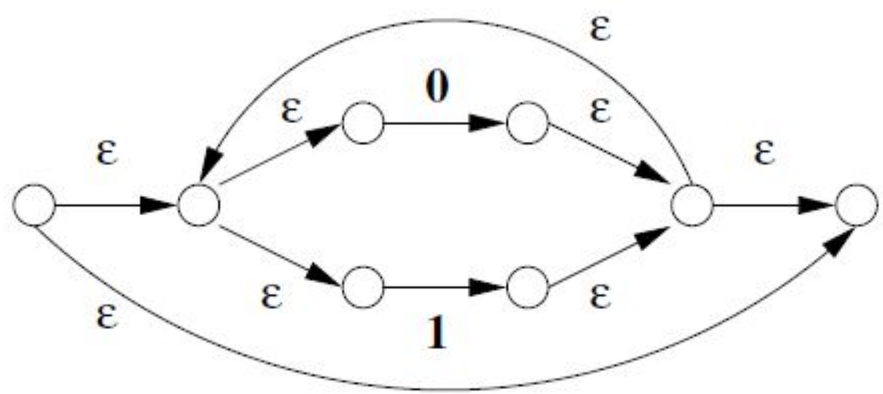
Example

- Convert the following RE to ε-NFA:

  (0+1)*1(0+1)

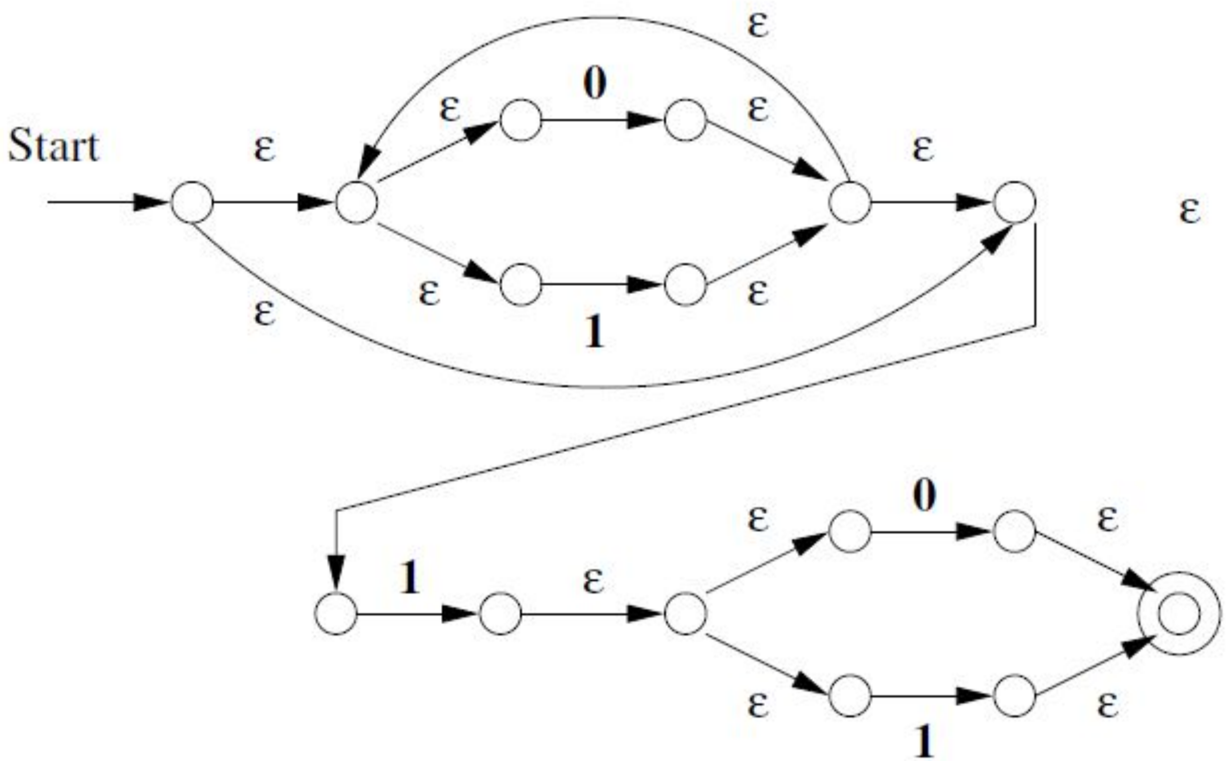


(a)



(b)

Example- Contd…



(c)

Try Yourself

- 01*
- (0+1)10
- 00(0+1)*

THANK YOU!