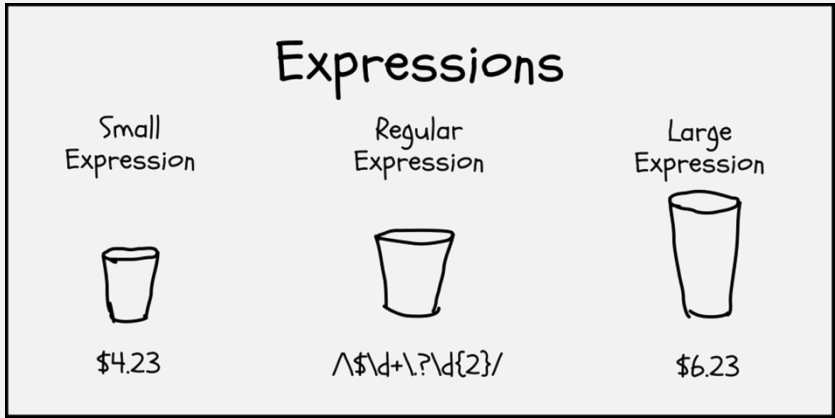
CS 420 / CS 620

Regular Expressions

Monday, October 6, 2025

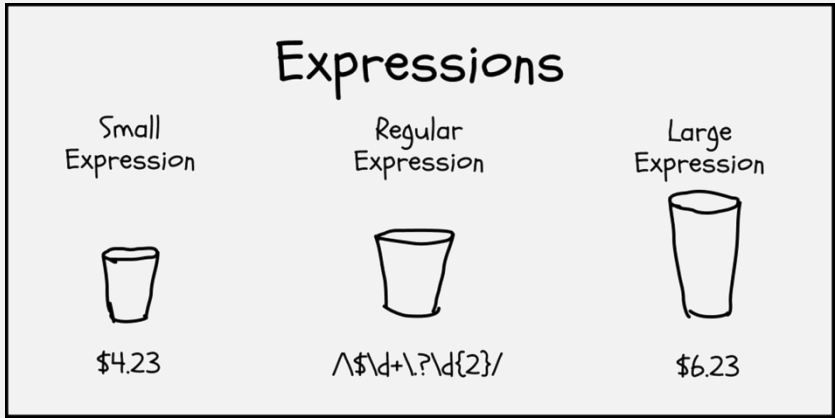
UMass Boston Computer Science



Announcements

- HW 4
  - ~~Due: Mon 10/6 12pm (noon)~~
- HW 5
  - Out: Mon 10/6 12pm (noon)
  - Due (unofficial): Mon 10/13 12pm (noon)
  - Due (up to): Wed 10/15 12pm (noon)
- HW 6 (most likely)
  - Out: Mon 10/13 12pm (noon)
  - Due: Wed 10/20 12pm (noon)
- No class: next Mon 10/13 (Indigenous Peoples)

(stay on schedule!)



In-class question (in gradescope) preview

When used as a string, the epsilon symbol (ε) is equivalent to which of the following?

When used as the empty transition, the epsilon symbol (ε) is equivalent to which of the following?

When used as a regular expression, the epsilon symbol (ε) is equivalent to which of the following?

When used as an input to an NFA's single-step δ function, the epsilon symbol (ε) is which of the following?

When used as an input to an NFA's multi-step $\hat{\delta}$ function, the epsilon symbol (ε) is which of the following?

When used as a transition label in a GNFA, the epsilon symbol (ε) is which of the following?

ast Time

List of Closed Ops for Reg Langs (so far)

- Union
- Concatentation
- Kleene Star (repetition) ?

Star: $A^* = \{x_1 x_2 \ldots x_k | \; k \geq 0 \text{ and each } x_i \in A\}$

Kleene Star Example

Let the alphabet Σ be the standard 26 letters $\{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\}$.

If $A = \{\mathtt{good}, \mathtt{bad}\}$

"repeat" zero

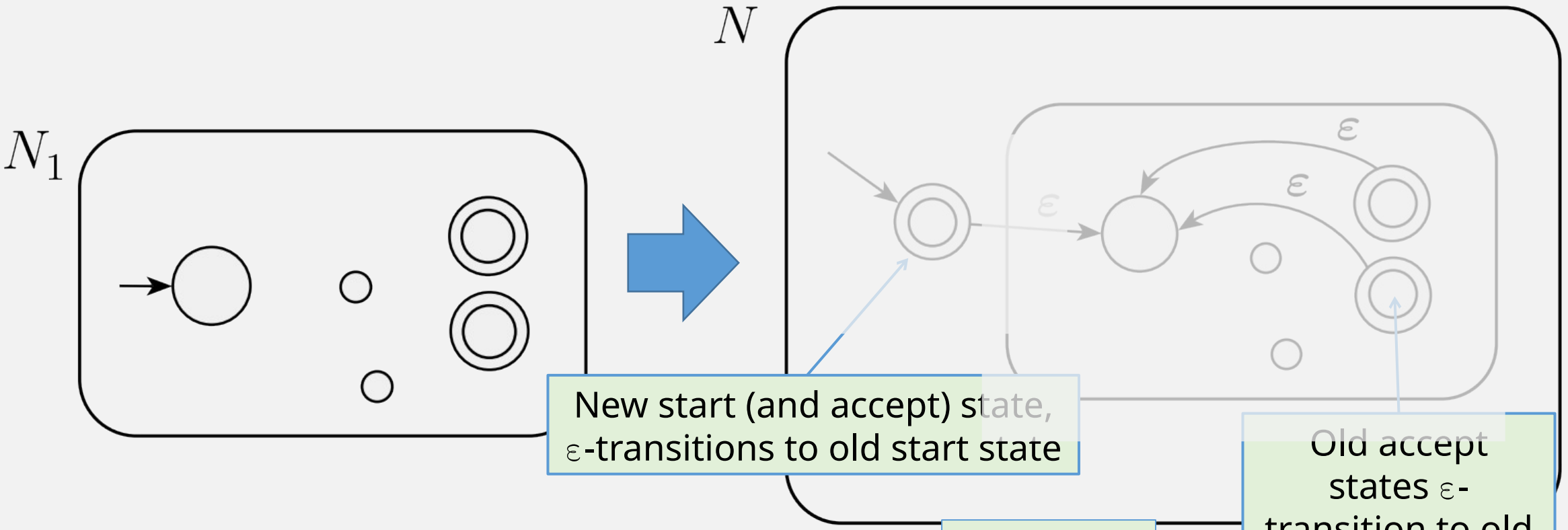
$$A^* = \{\varepsilon, \mathtt{good}, \mathtt{bad}, \mathtt{goodgood}, \mathtt{goodbad}, \mathtt{badgood}, \mathtt{badbad}, \mathtt{goodgoodgood}, \mathtt{goodgoodbad}, \mathtt{goodbadgood}, \mathtt{goodbadbad}, \ldots\}$$

Note: repeat strings in A zero or more times

(this is an infinite language!)

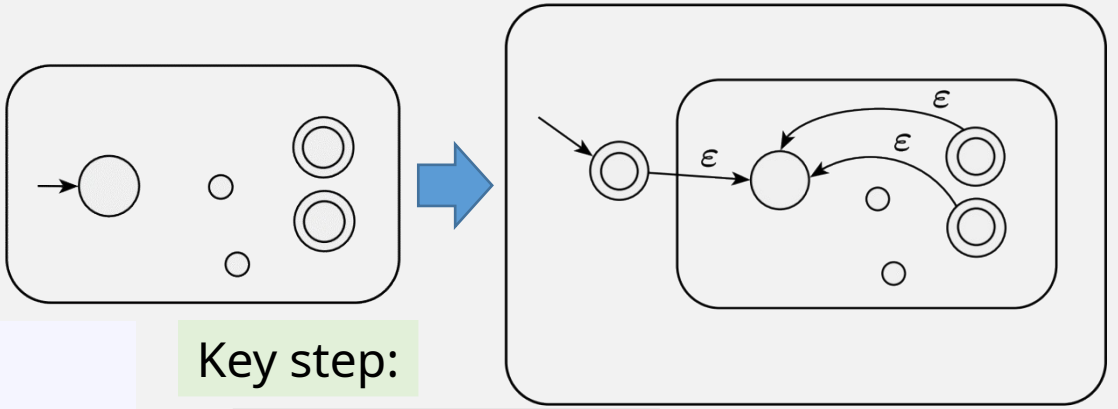
Star: $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$

Kleene Star is Closed for Regular Langs?



-class exercise

Kleene Star is Closed for Regular Langs



THEOREM

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

Key step:

STAR_{NFA} : NFA $\rightarrow$ NFA

Where:

$N = \text{STAR}_{\text{NFA}}(N_1)$

$L(N) = L(N_1)^*$

-class solution

Kleene Star is Closed for Regular Langs

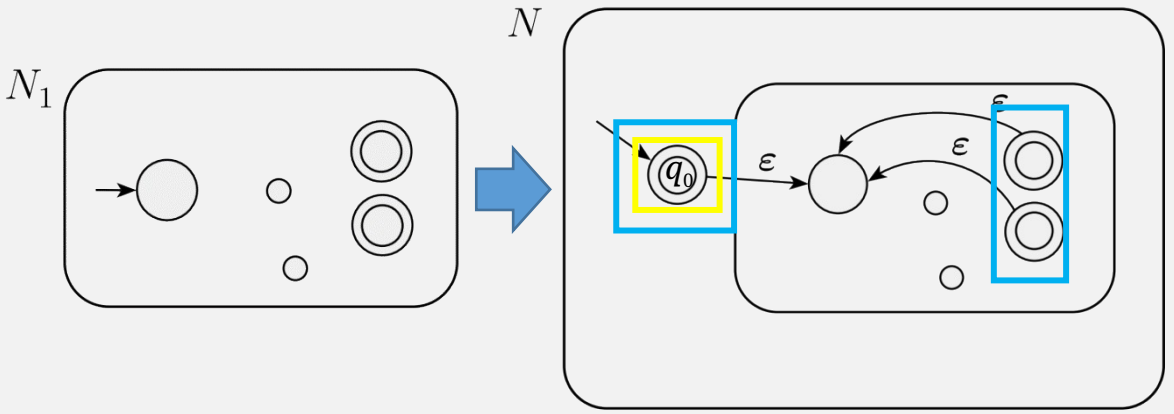
(part of)

$N = \text{STAR}_{\text{NFA}}(N_1)$

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

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

1. $Q = \{q_0\} \cup Q_1$
2. The state q_0 is the new start state.
3. $F = \{q_0\} \cup F_1$



Kleene star of a language must accept the empty string!

-class solution

Sipser Thm 1.49

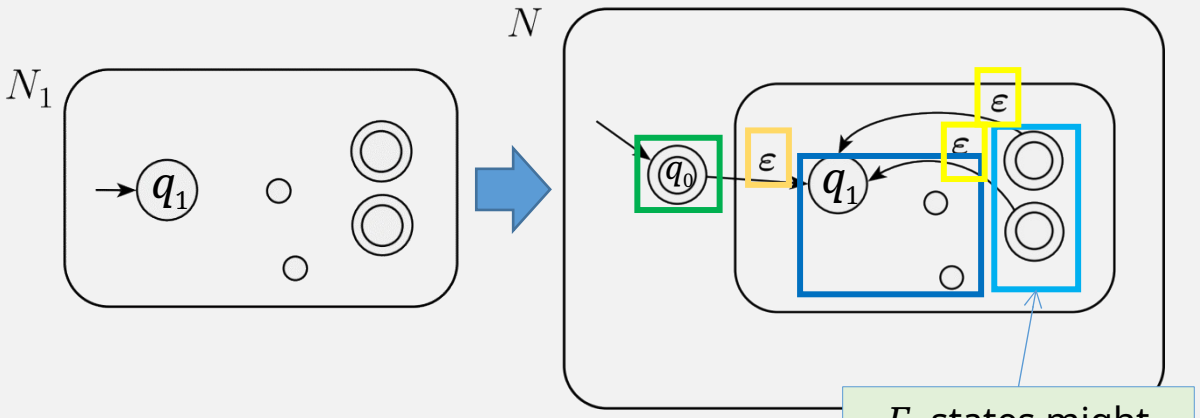
Kleene Star is Closed for Regular Langs

(part of)

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

$N = \text{STAR}(N_1)$ $N = (Q, \Sigma, \delta, q_0, F)$ to recognize A_1^*.

NFA



F_1 states might previously have empty transitions!

1. $Q = \{q_0\} \cup Q_1$
2. The state q_0 is the new start state.
3. $F = \{q_0\} \cup F_1$
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

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

Old accept states ε-transition to old start state q_1

New start state ε-transitions to old start state q_1

New start state has only ε-transitions

F_1 states might previously have empty transitions!

Must add transitions for any added states

Why These (Closed) Operations?

- Union
- Concatenation
- Kleene star (repetition)

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

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

$$A^* = \{x_1 x_2 \ldots x_k | \, k \geq 0 \text{ and each } x_i \in A\}$$

All regular languages can be constructed from:

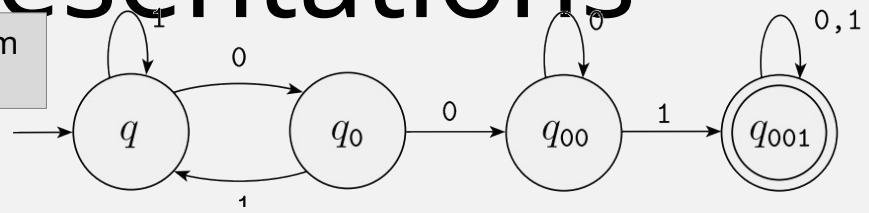
- (language of) single-char strings (from some alphabet), …
- And these three closed operations!

e.g., lang {“a”} , lang {“b”} , …

So Far: Regular Language Representations

State diagram (NFA/DFA)

1.



Formal description

2.

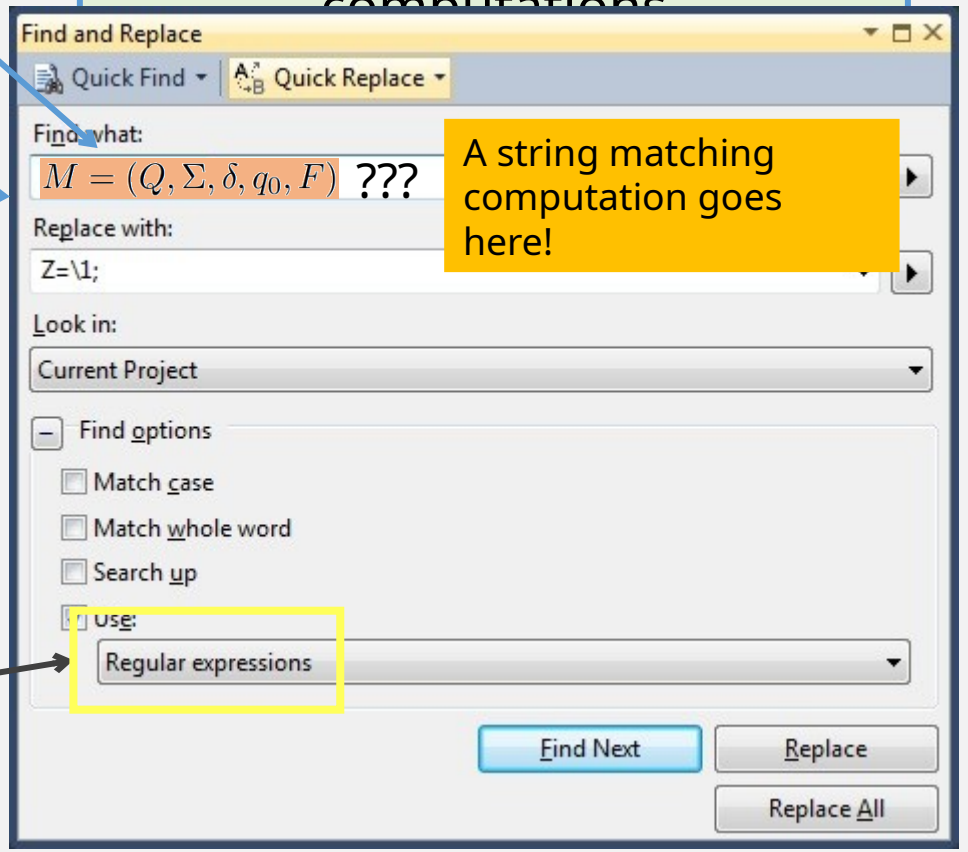
1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

4. q_1 is the start state
5. $F = \{q_2\}$

(hard to write)

Actually, it's a real programming language, for text search / string matching computations



Our Running Analogy:
- Set of all regular languages ~ a "programming language"
- One regular language (or any equiv representation) ~ a "program"

? 3. $\Sigma^*001\Sigma^*$

Need a more concise (textual) notation??

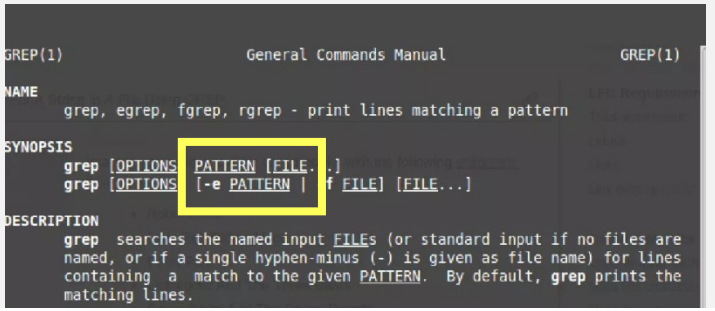
Regular Expressions:
A Widely Used Programming Language
(usually within tools / languages)

- Unix / Linux
- Java
- Python
- Web APIs

java.util.regex

Class Pattern

java.lang.Object
    java.util.regex.Pattern



Python » English 3.8.6rc1 Documentation » The Python Standard Library » Text Processing Services »

— Regular expression operations

ce code: Lib/re.py

module provides regular expression matching operations similar to those found in Perl.

About regular expressions (regex)

Analytics supports regular expressions so you can create more flexible definitions for things like view filters, goals, segments, audiences, content groups, and channel groupings.

This article covers regular expressions in both Universal Analytics and Google Analytics 4.

In the context of Analytics, regular expressions are specific sequences of characters that broadly or narrowly match patterns in your Analytics data.

For example, if you wanted to create a view filter to exclude site data generated by your own employees, you could use a regular expression to exclude any data from the entire range of IP addresses that serve your employees. Let's say those IP addresses range from 198.51.100.1 - 198.51.100.25. Rather than enter 25 different IP addresses, you could create a regular expression like **198\.51\.100\.\d*** that matches the entire range of addresses.

Why These (Closed) Operations?

- Union
- Concatenation
- Kleene star (repetition)

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

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

$$A^* = \{x_1 x_2 \ldots x_k | \, k \geq 0 \text{ and each } x_i \in A\}$$

All regular languages can be constructed from:
- (language of) single-char strings (from some alphabet), and
- these three closed operations!

They are used to define regular expressions!

Regular Expressions: Formal Definition

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.

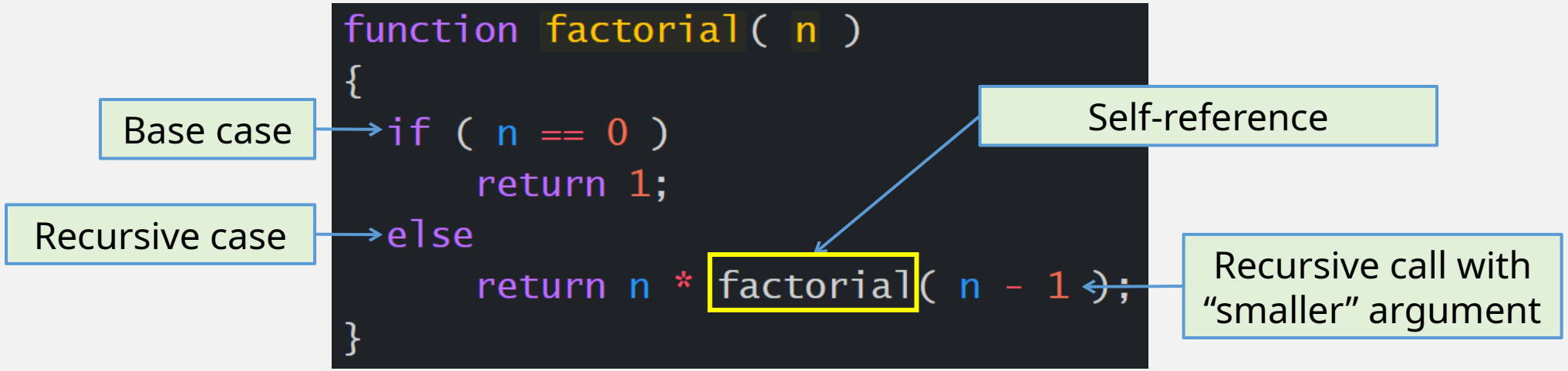
This is a recursive definition

Flashback: Recursive Definitions

Recursive definitions are definitions with a self-reference

A valid recursive definition must have:
- base case and
- recursive case (with a “smaller” self-reference)

Flashback: Recursive Definitions



Flashback: Recursive Definitions

A Natural Number is either:

- Zero, or
- the Successor of a Natural Number

Base case

Recursive case

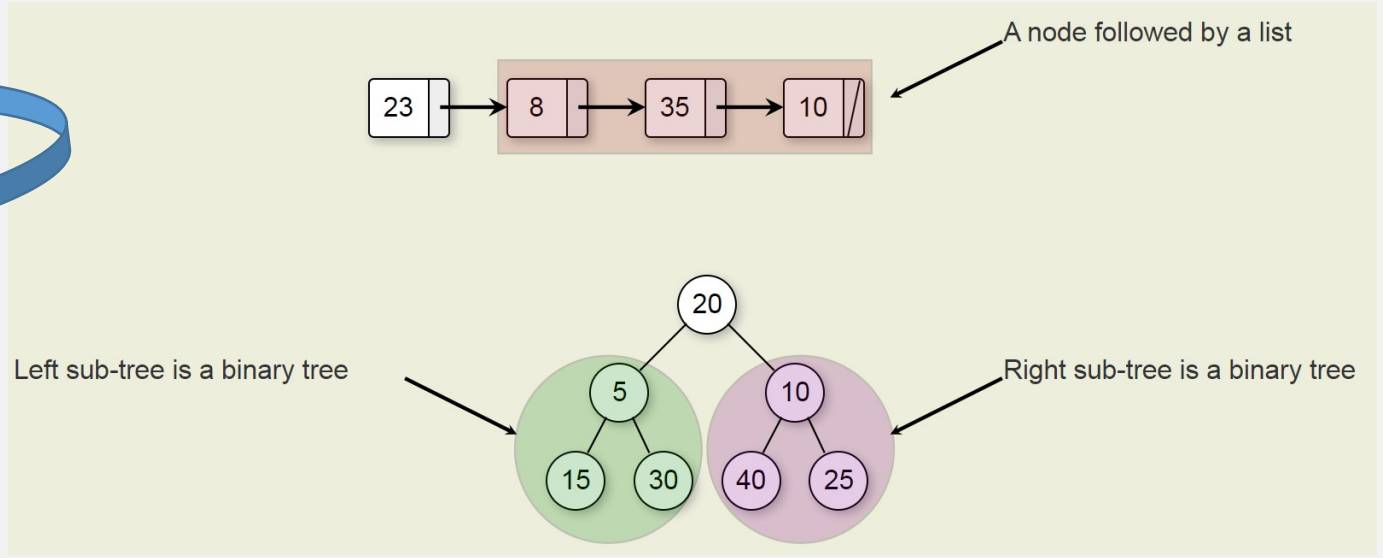
Self-reference

"smaller" argument

Flashback: Recursive Definitions

```
/* Linked list Node*/
class Node {
    int data;
    Node next;
}
```

Smaller self-reference



Q: Where's the base case??

> I call it my billion-dollar mistake. It was the invention of the null reference in 1965.
>
> — Tony Hoare —

Data structures are commonly defined recursively

Regular Expressions: Formal Definition

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.

3 Base Cases

(A lang containing a) length-1 string

(A lang containing) the empty string

(This is the 3rd use of the ε symbol!)

The empty set (i.e., a lang containing no strings)

unio

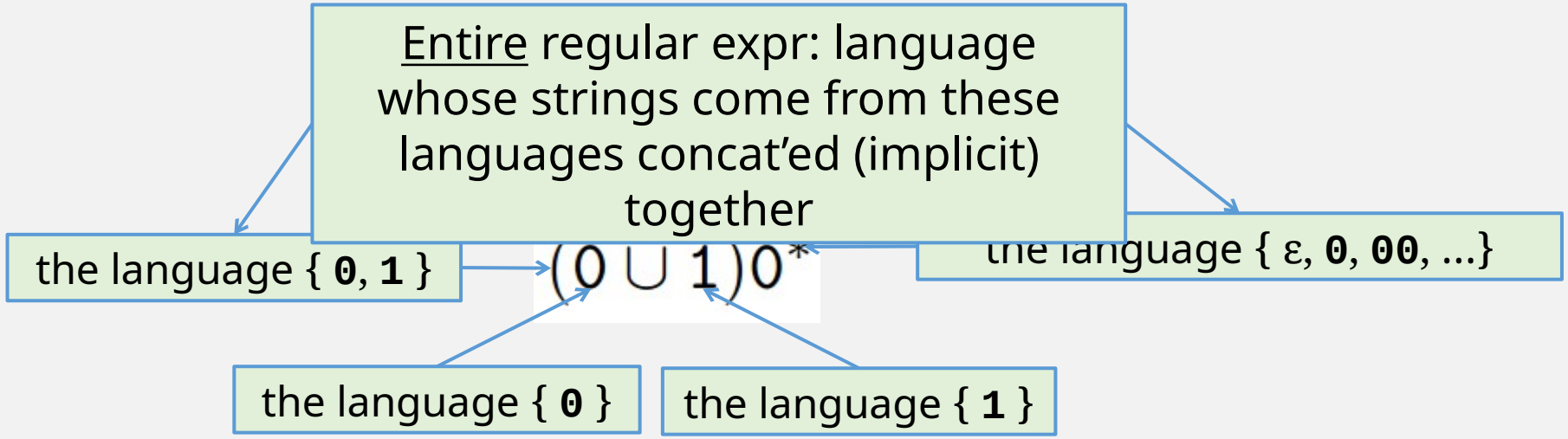
conca

star

3 Recursive Cases

Note:
- A regular expression represents a language
- The *set of all* regular expressions represents a *set of* languages

Regular Expression: Concrete Example



- Operator Precedence:
  - Parentheses
  - Kleene Star
  - Concat (sometimes use ∘, sometimes implic
  - Union

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.

alphabet Σ is $\{0,1\}$

Regular Expression: More Examples

$0^*10^* = \{w | \ w \text{ contains a single } 1\}$

$\Sigma^*1\Sigma^* = \{w | \ w \text{ has at least one } 1\}$ Σ in regular expression = "any char"

$1^*(01^+)^* = \{w | \ \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$ let R^+ be shorthand for RR^*

$(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$ $0 \cup \varepsilon$ describes the language $\{0, \varepsilon\}$

$1^*\emptyset = \emptyset$ $A \circ B = \{xy | \ x \in A \text{ and } y \in B\}$

nothing in B = nothing in $A \circ B$

$\emptyset^* = \{\varepsilon\}$ Star of any lang has ε

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.

Regular Expressions = Regular Langs?

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.

We would like to say:

- A regular expression represents a regular language
- The *set of all* regular expressions represents the *set of all* regular languages

(But we have to prove it)

Thm: A Lang is Regular iff Some Reg Expr Describes It

⇒ If a language is regular, then it's described by a reg expression

⇐ If a language is described by a reg expression, then it's regular (Easier)

- Key step: convert reg expr $\rightarrow$ equivalent NFA!
- (Hint: we mostly did this already when discussing closed ops)

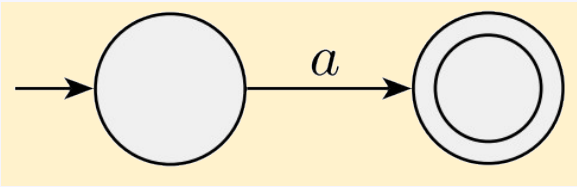
How to show that a language is regular?

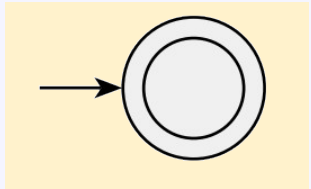
Construct a DFA *or* NFA!

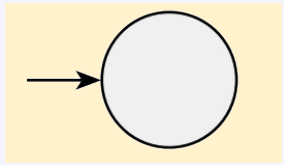
RegExpr→NFA

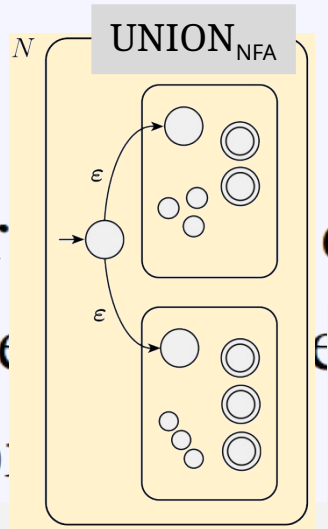
R is a ***regular expression*** if R is

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 ar… expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are…
6. (R_1^*), where R_1 is a regular exp…

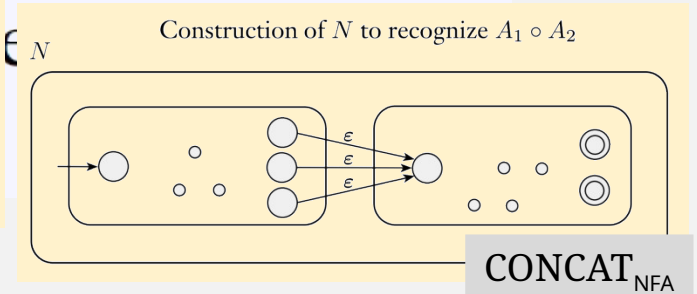




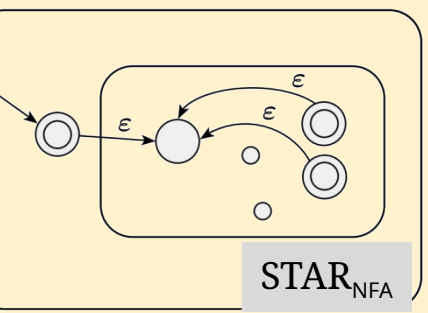




UNION$_{NFA}$



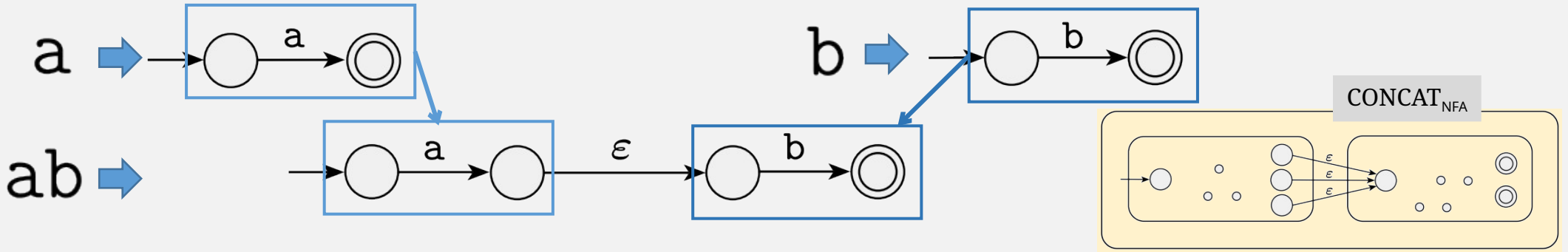
CONCAT$_{NFA}$



STAR$_{NFA}$

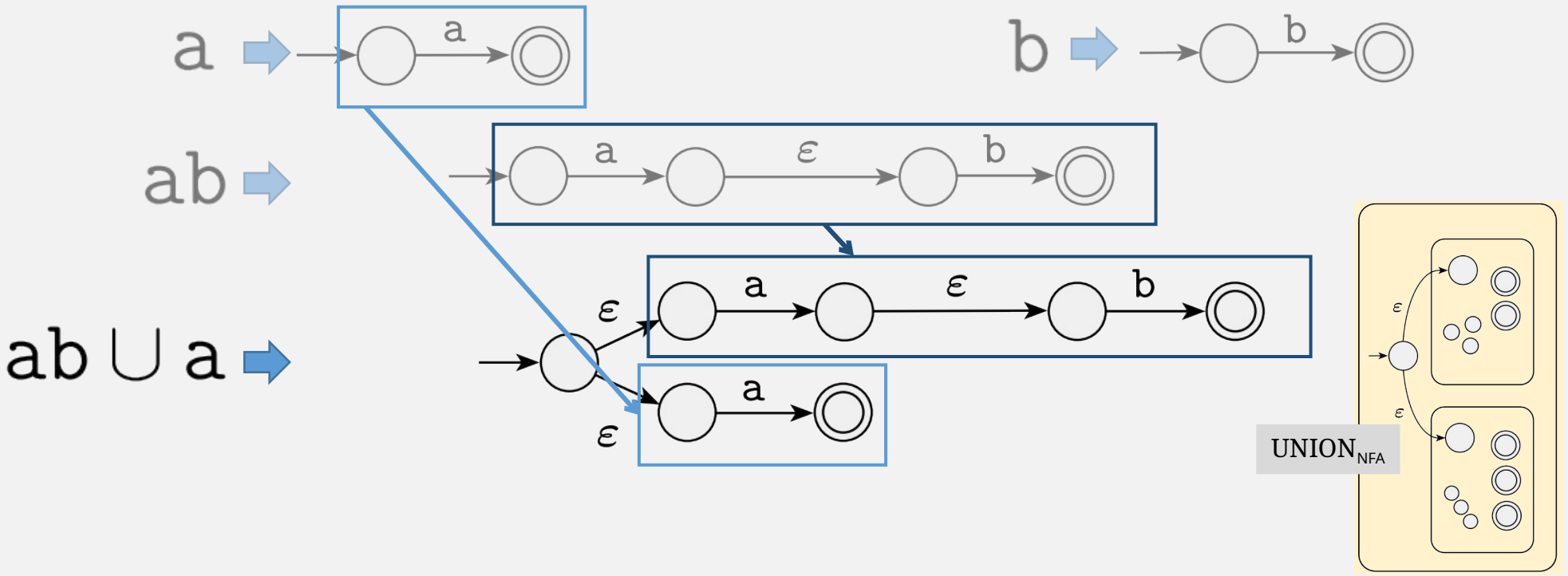
RegExpr→NFA: Example

convert the regular expression $(\mathtt{ab} \cup \mathtt{a})^*$ to an NFA ... step by step



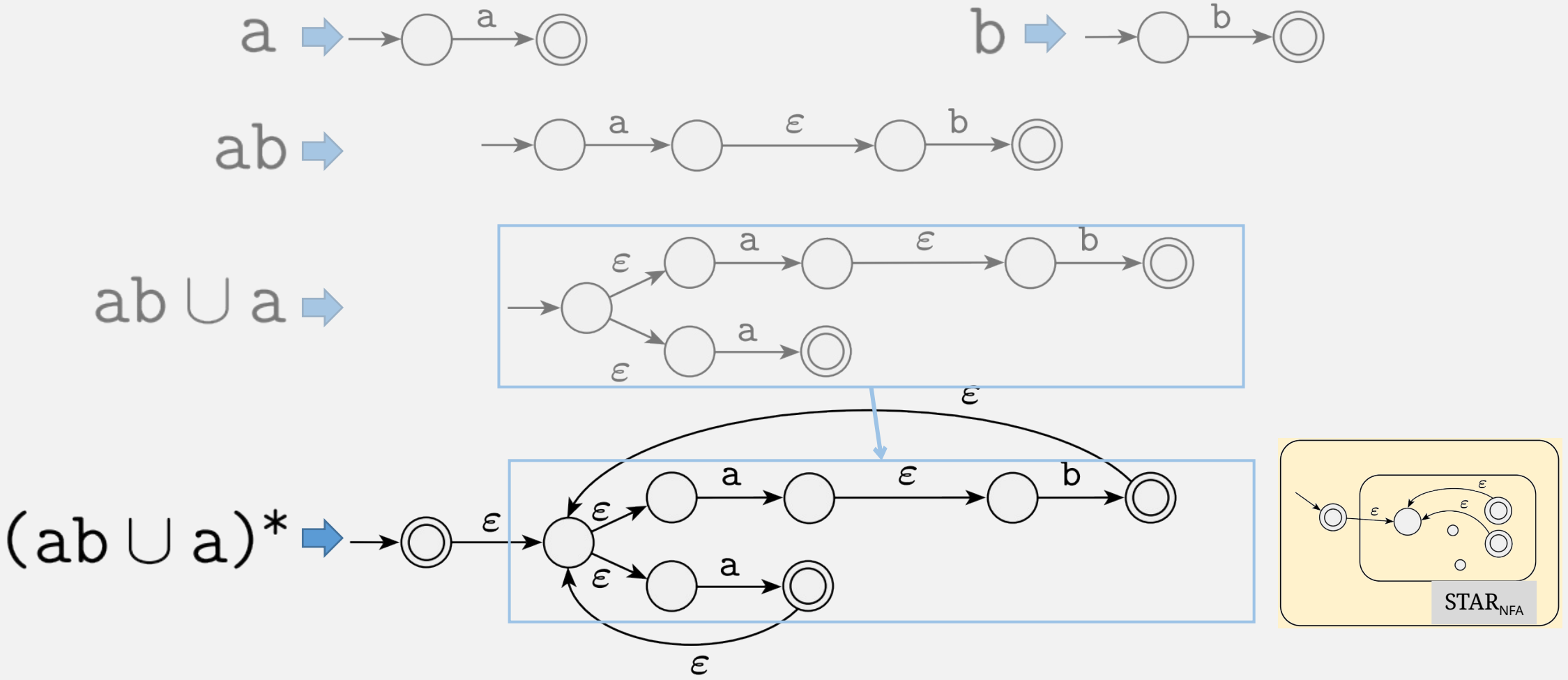
RegExpr→NFA: Example

convert the regular expression $(ab \cup a)^*$ to an NFA



RegExpr→NFA: Example

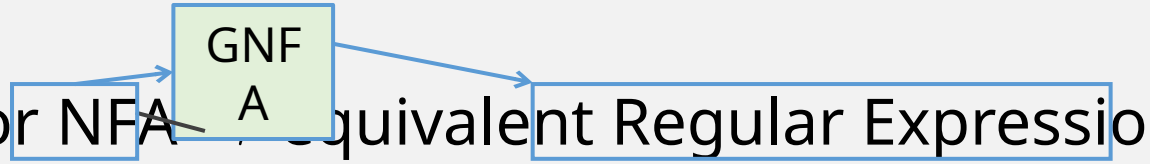
convert the regular expression $(ab \cup a)^*$ to an NFA



Thm: A Lang is Regular iff Some Reg Expr Describes It

⇒ If a language is regular, then it's described by a reg expression (Harder)

- Key step: Convert an ~~DFA or NFA~~ GNFA to an equivalent Regular Expression



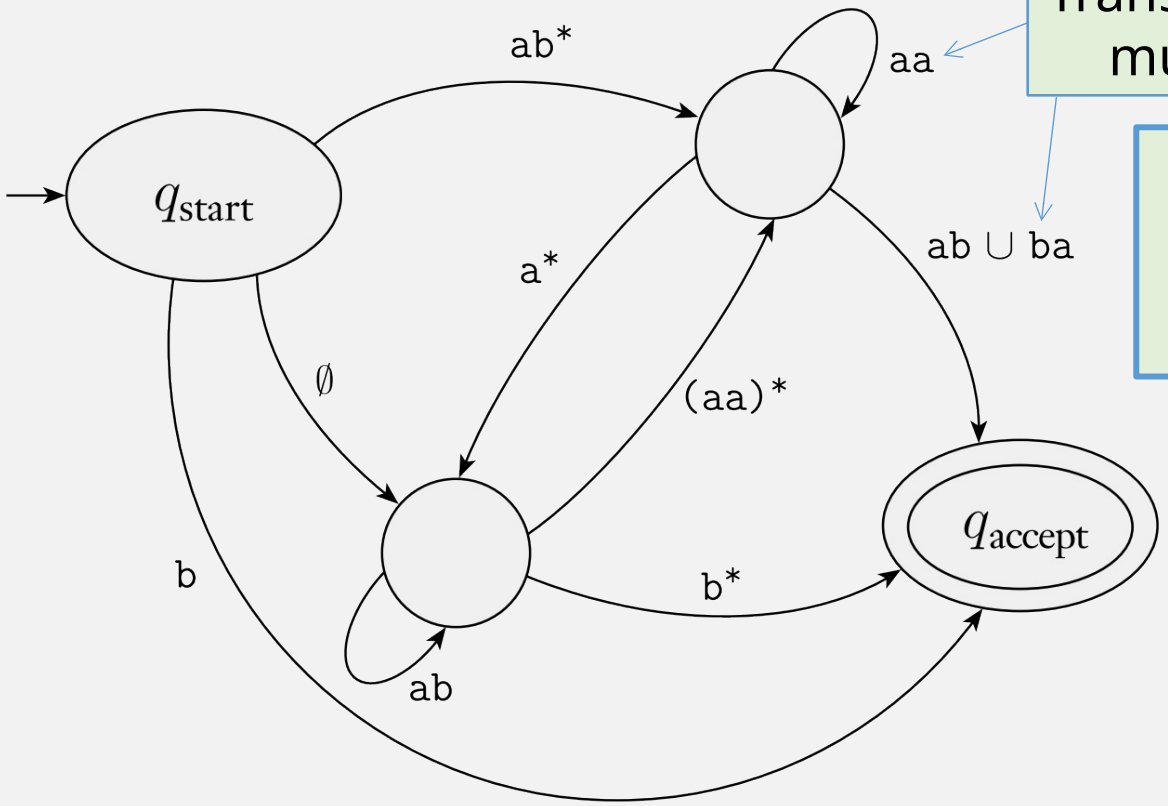
- First, we need another kind of finite automata: a GNFA

⇐ If a language is described by a reg expression, then it's regular (Easier)

- Key step: Convert the regular expression → an equivalent NFA!

(full proof requires writing Statements and Justifications, and creating an "Equivalence" Table)

Generalized NFAs (GNFAs)



Transition can read multiple chars

plain NFA
= GNFA with single char regular expr transitions

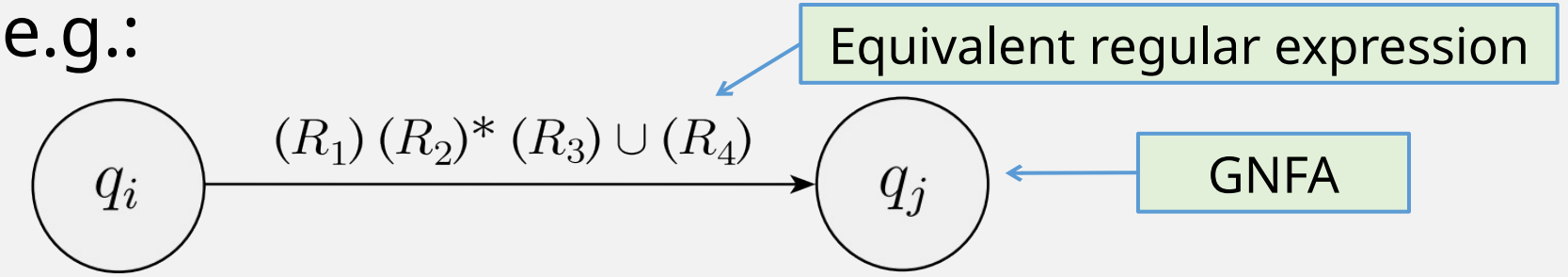
Goal: convert GNFAs to equivalent Regular Exprs

- GNFA = NFA with regular expression transitions

GNFA→RegExpr function :

On GNFA input G:

- If G has 2 states, return the regular expression (on the transition), e.g.:

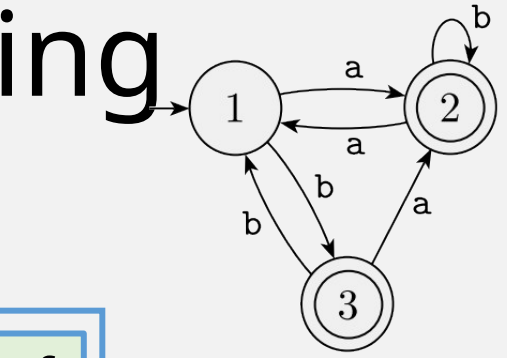


Could there be less than 2 states?

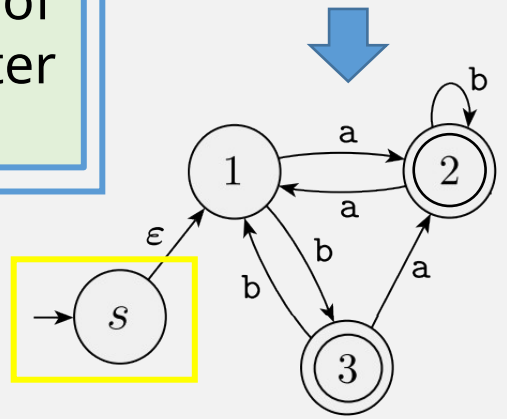
GNFA→RegExpr Preprocessing

- Modify input machine to have:

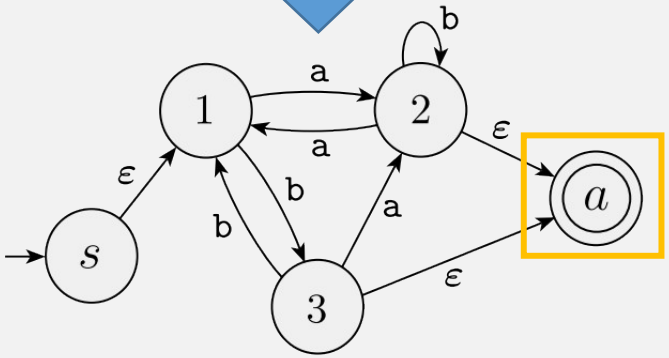
Does this change the language of the machine? i.e., are before/after machines equivalent?



- New start state:
  - No incoming transitions
  - ε transition to old start state



- New, single accept state:
  - With ε transitions from old accept states



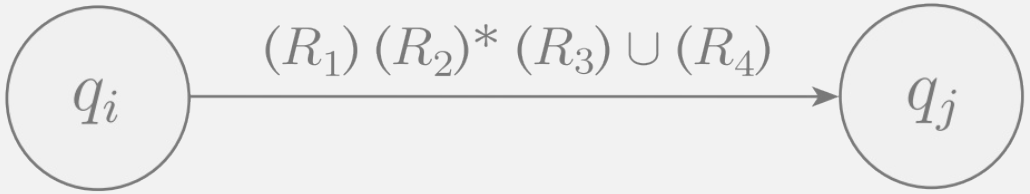
Modified machine always has 2+ states:
- New start state
- New accept state

GNFA→RegExpr function (recursive)

On GNFA input G:

Base Case

- If G has 2 states, return the regular expression (from transition), e.g.:



Recursive Case

- Else:
  - "Rip out" one state
  - "Repair" the machine to get an equivalent GNFA G'
  - Recursively call GNFA→RegExpr(G')

Recursive definitions have:
- base case and
- recursive case
  (with "smaller" self-reference)