Chapter 7: Kleene's Theorem

Regular expressions, Finite Automata, transition graphs are all the same!!

- Method of proof

  Let A,B,C be sets such that $A \subseteq B$, $B \subseteq C$, $C \subseteq A$.
  Then A=B=C.

  Remark: Regular expressions, finite automata, and transition graphs each define a set of languages.

Kleene's Theorem

Any language that can be defined by a regular expression, or finite automaton, or transition graph can be defined by all three methods.

Proof of Kleen's theorem: It is enough to prove each of the lemmas below.

Lemma 1: Every language that can be defined by a finite automaton can also be defined by a transition graph.

Lemma 2: Every language that can be defined by a transition graph can also be defined by a regular expression.

Lemma 3: Every language that can be defined by a regular expression can also be defined by a finite automaton.

$$FA \subset TG \subset RE \subset FA$$

Lemma 1: Every language that can be defined by a finite automaton can also be defined by a transition graph.

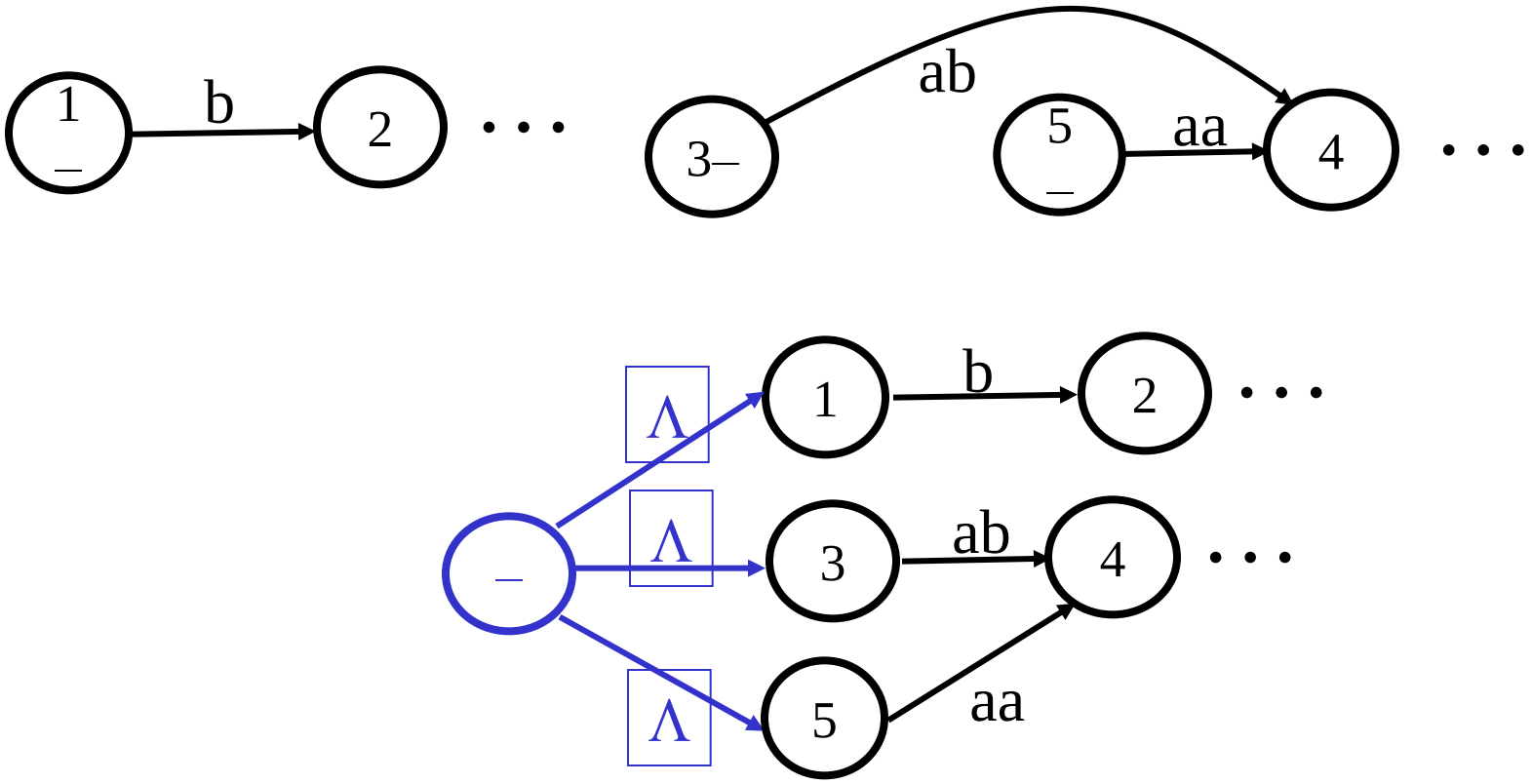
Proof: By definition, every finite automaton is a transition graph.

Lemma 2: Every language that can be defined by a transition graph can also be defined by a regular expression.

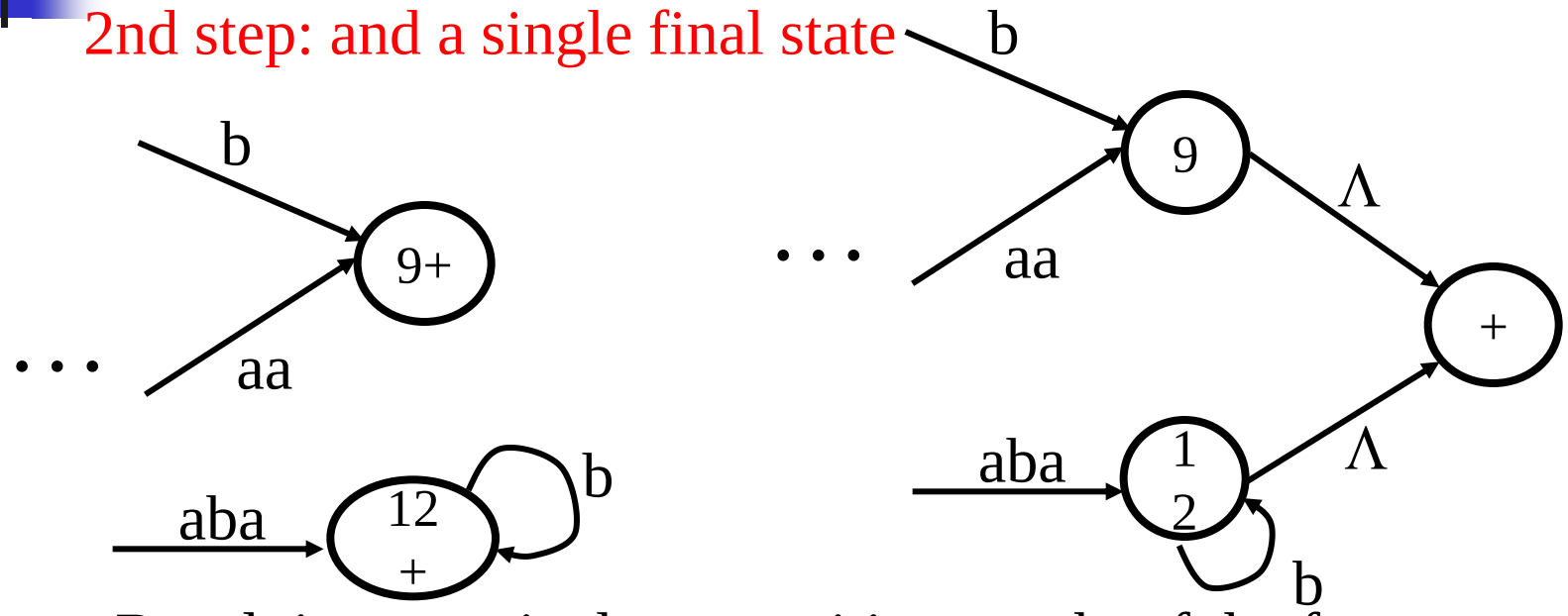
Proof: By constructive algorithm, that works

1. for every transition graph
2. in a finite number of steps

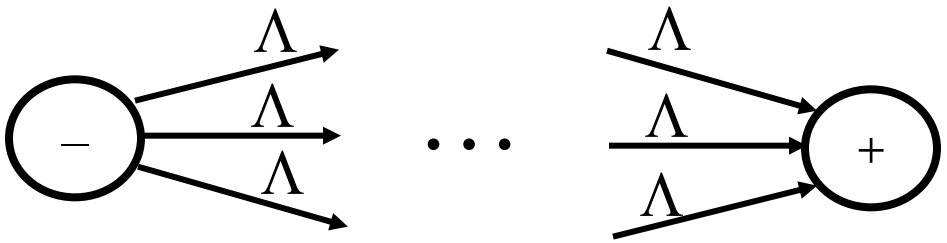
1st step: transform to a transition graph with a single start state.



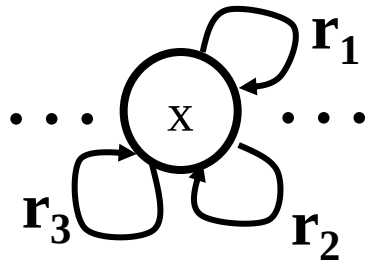
2nd step: and a single final state

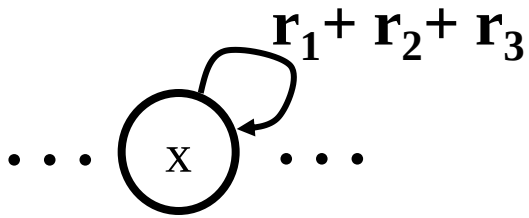


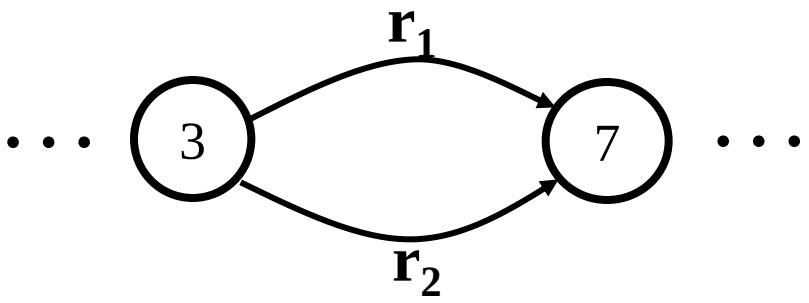
Result is an equivalent transition graph, of the form:

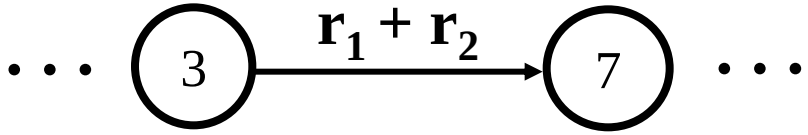


3a. Combine edges that have the same starting and ending state.



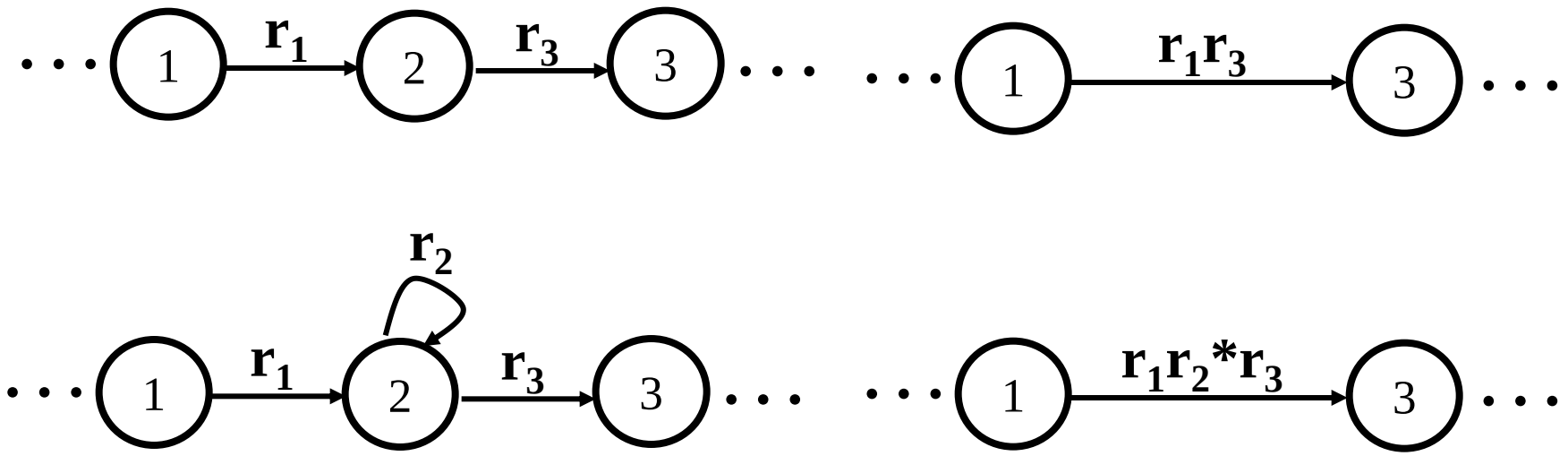




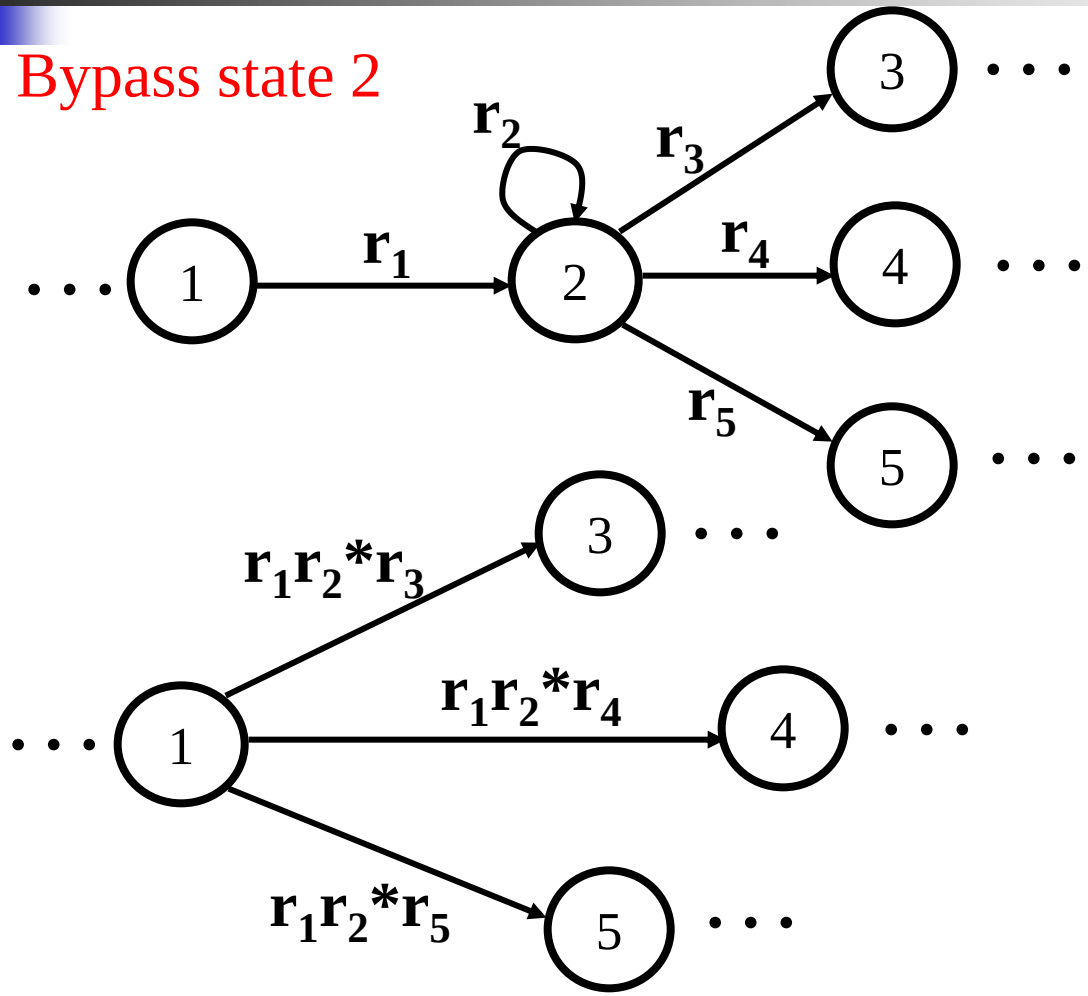


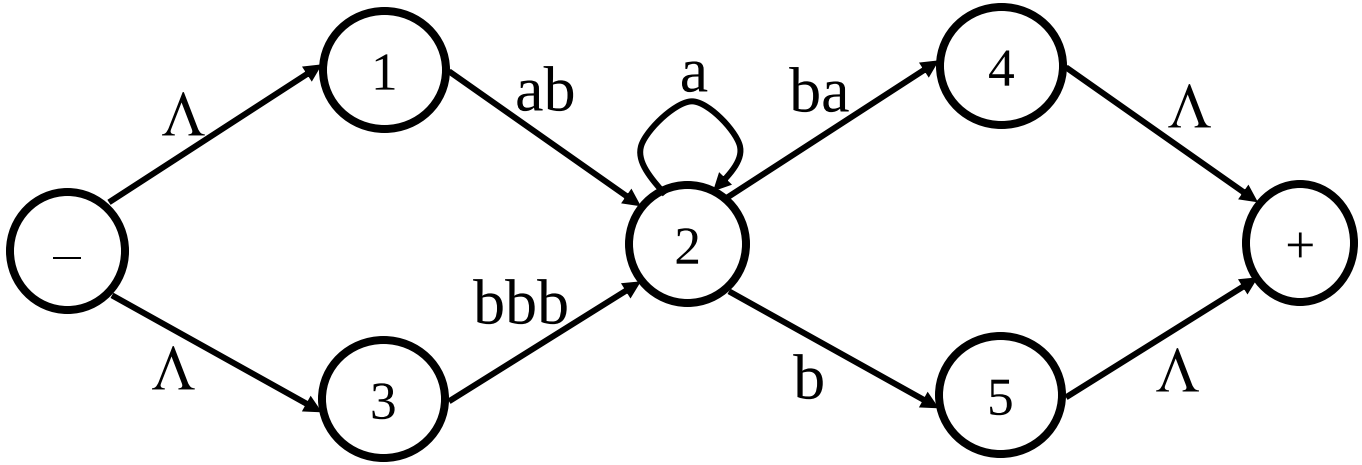
3b. Eliminate states one by one:

bypass and state elimination operation



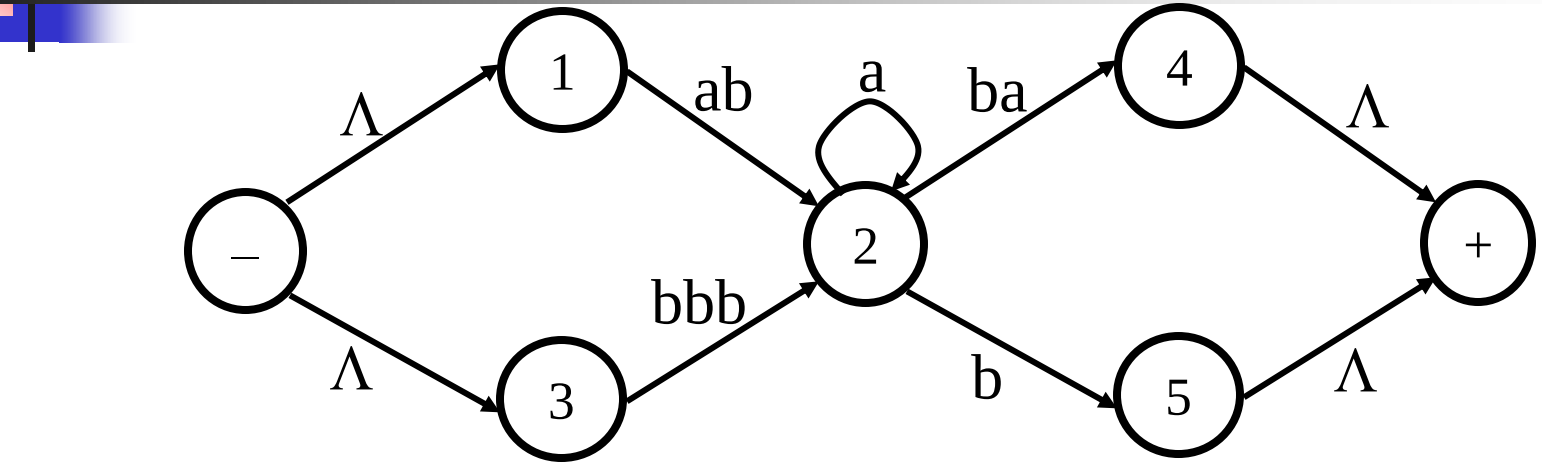
Bypass state 2

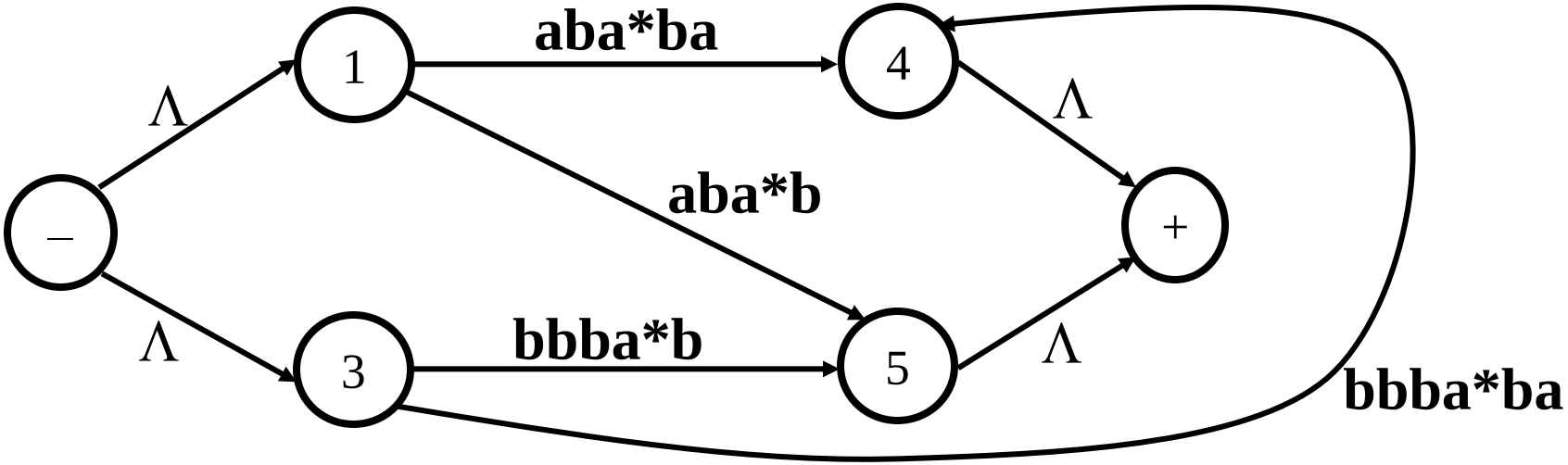


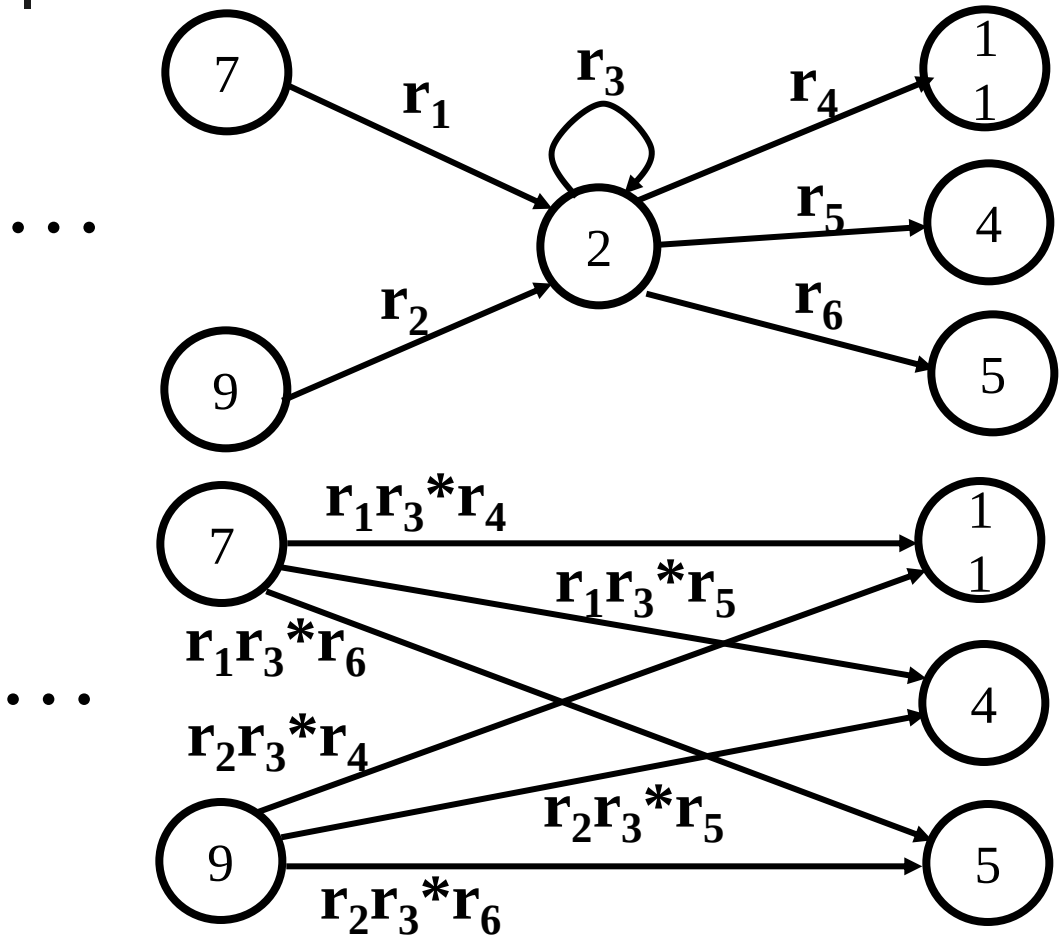


bypass state 2,

paths through state 2: 1→ 2→4, 1→2→5, 3→2→4, 3→2→5

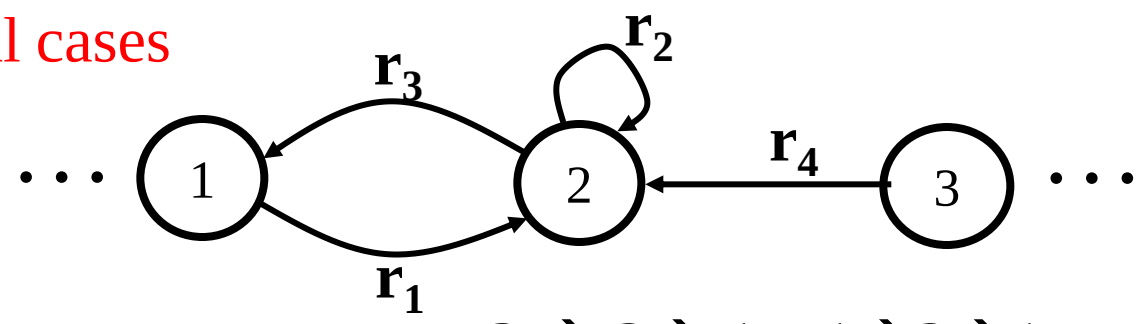

1
ab
a
ba
4
Λ
Λ
–
2
+
bbb
Λ
3
b
5
Λ


aba*ba
1
4
Λ
Λ
aba*b
–
+
Λ
3
bbba*b
5
Λ
bbba*ba

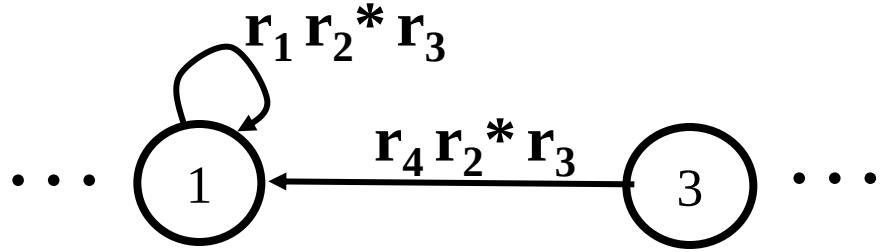


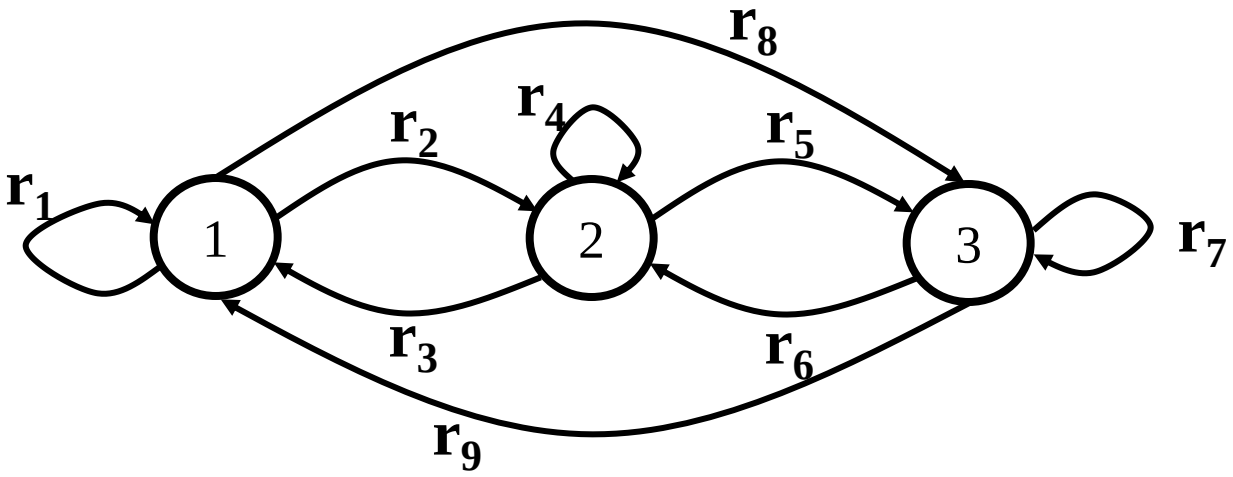
Even with many paths through state 2, always an equivalent GTG.

Special cases

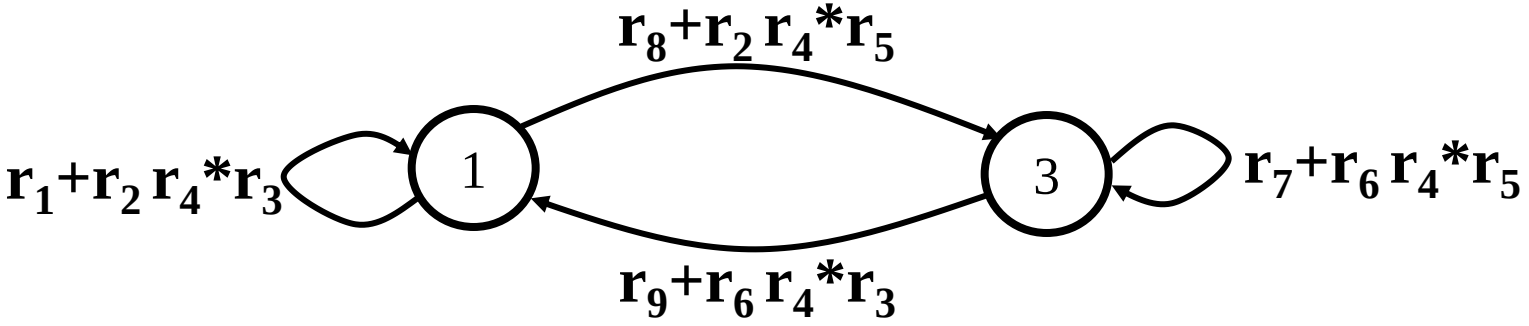


3 → 2→ 1, 1→2→1

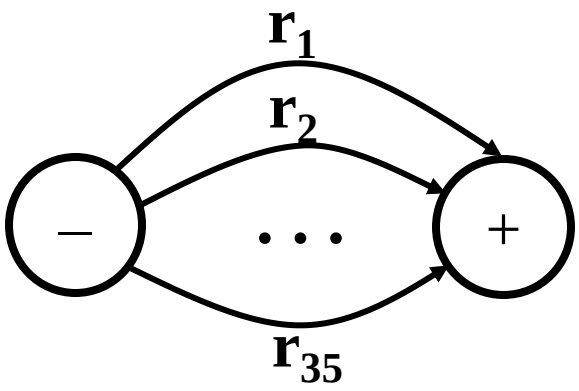


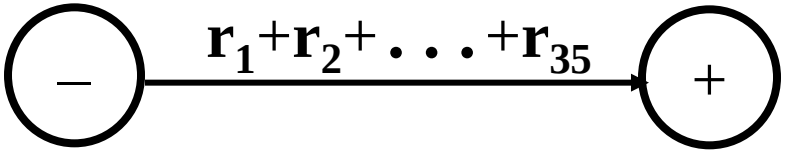


3 →2→1,
3→2→3,
1 →2→1,
1→2→3

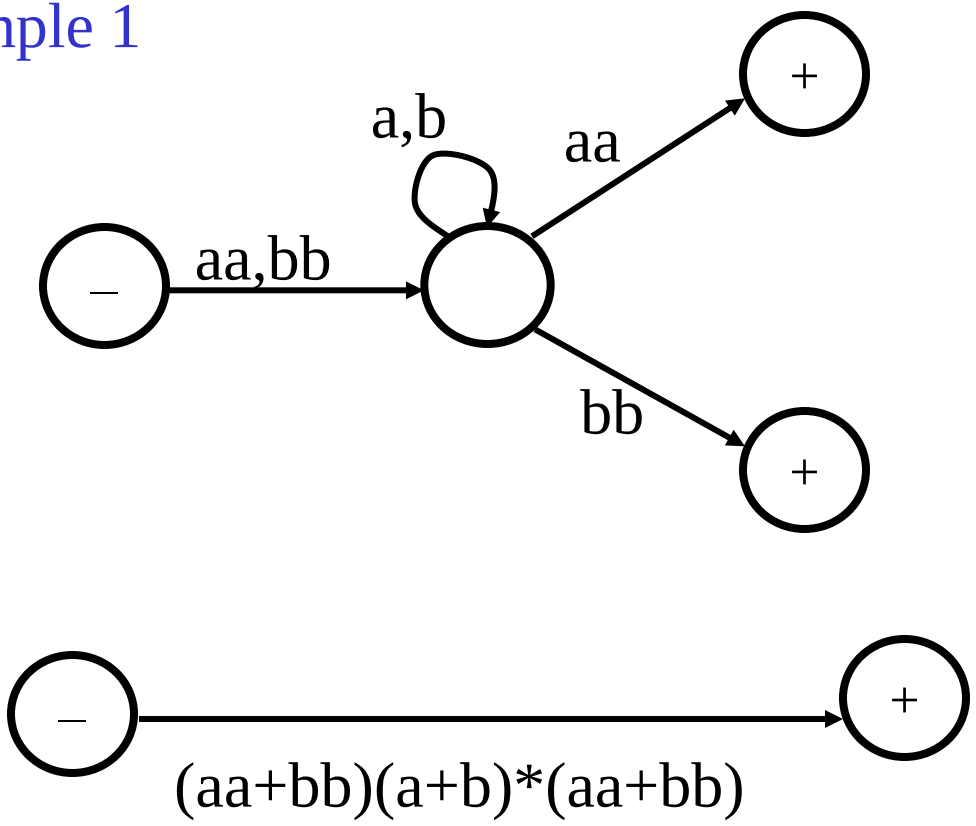


3a. Combine edges

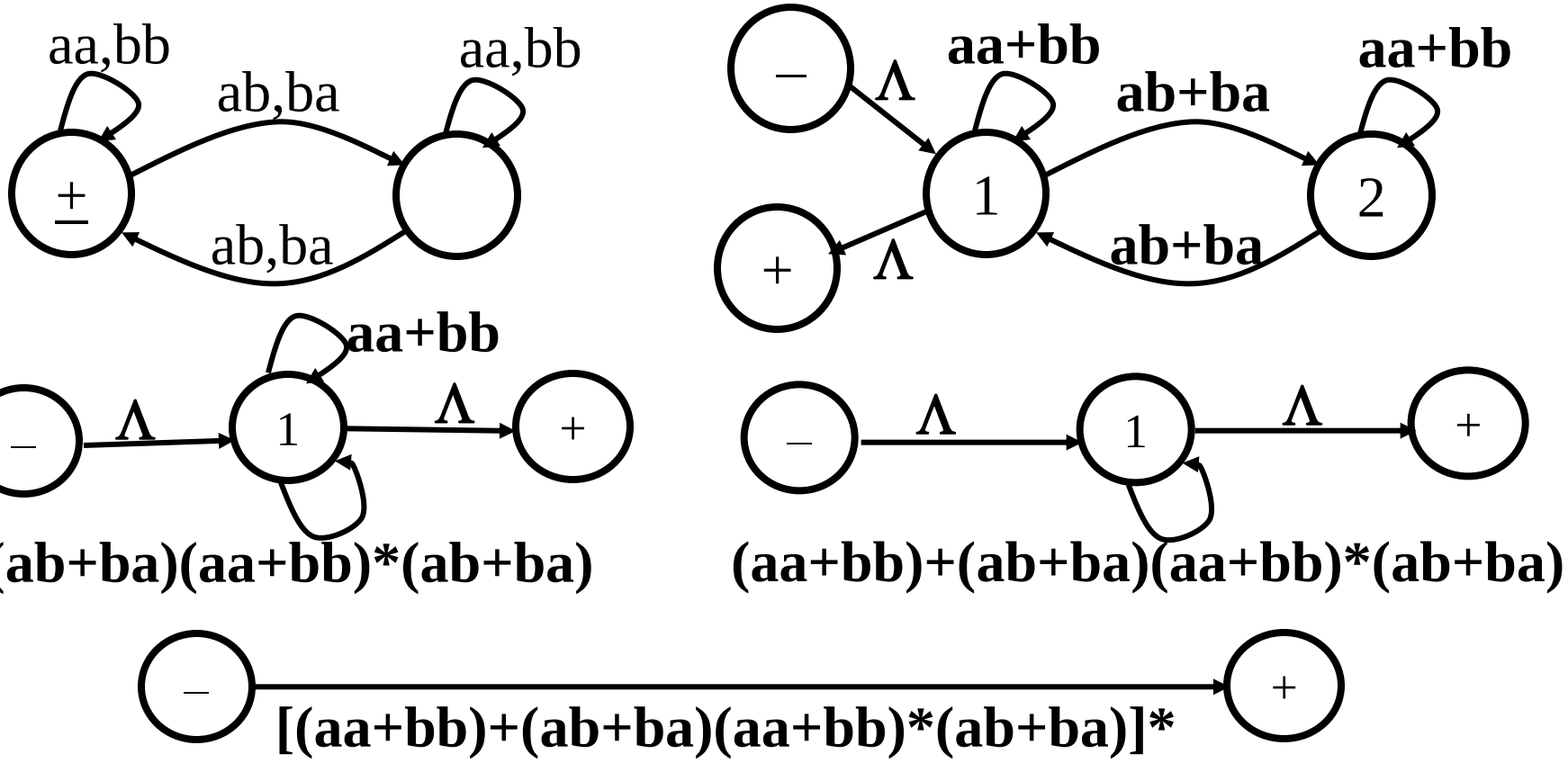




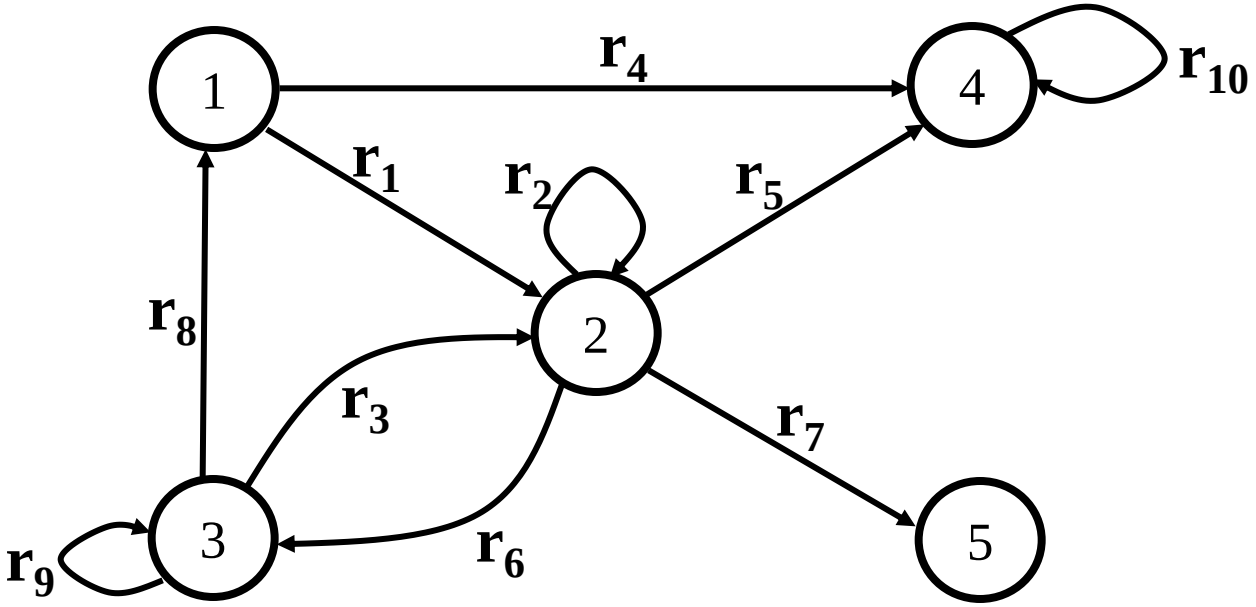
Example 1



EXAMPLE2: EVEN-EVEN



Example 3



Transition Graph → Regular Expression

- Algorithm (and proof)

1. Add (if necessary) a unique start state without incoming edges and a unique final state without outgoing edges.

For each state that is not a start state or a final state, repeat steps 2 and 3.

2. Do a bypass and state elimination operation.
3. Combine edges that have the same starting and ending state.
4. Combine the edges between the start state and the final state. The label on the only remaining edge is the regular expression result. If there is none, the regular expression is ϕ.

Lemma 3: Every language that can be defined by a regular expression can also be defined by a finite automaton.

Proof: By constructive algorithm starting from the recursive definition of regular expressions.

- Remember: Given an alphabet Σ, the set of regular expressions is defined by the following rules.
  1. For every letter in Σ, the letter written in bold is a regular expression. **Λ** is a regular expression.
  2. If $\mathbf{r_1}$ and $\mathbf{r_2}$ are regular expressions, so is $\mathbf{r_1}+\mathbf{r_2}$.
  3. If $\mathbf{r_1}$ and $\mathbf{r_2}$ are regular expressions, so is $\mathbf{r_1}\,\mathbf{r_2}$.
  4. If $\mathbf{r_1}$ is a regular expression, so is $\mathbf{r_1}^*$.
  5. Nothing else is a regular expression.

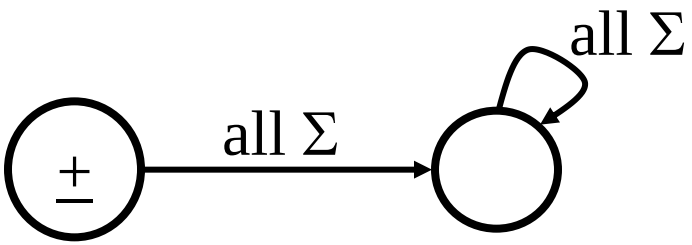
Chapter 7: Kleene's Theorem

Build a finite automaton that accepts the language: **(a+b)*(aa+bb)(a+b)***

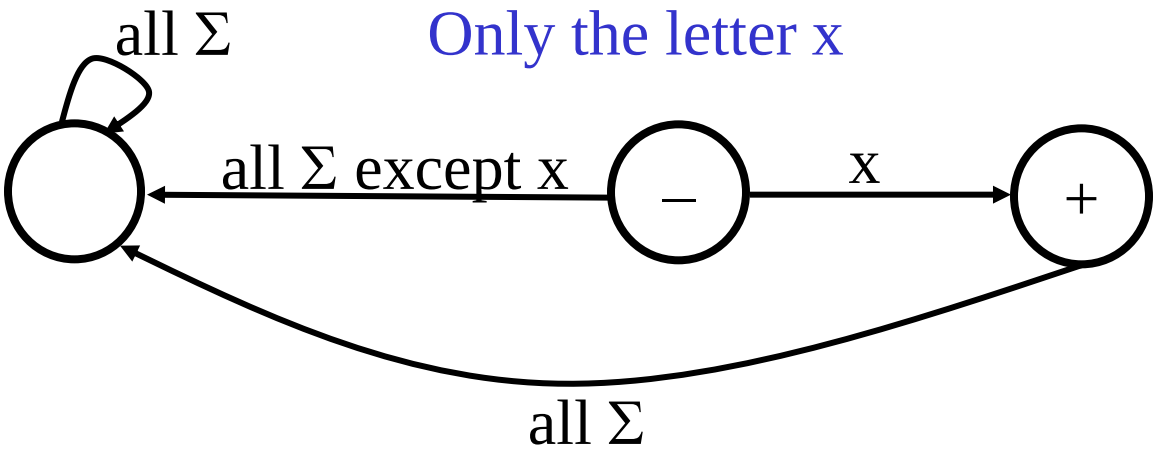
Rule:

1. The letter **a** 1
2. The letter **b** 1
3. The word **aa** (using 1) 3
4. The word **bb** (using 2) 3
5. The expression **aa+bb** (using 3 and 4) 2
6. The expression **a+b** (using 1 and 2) 2
7. The expression **(a+b)*** (using 6) 4
8. The expression **(a+b)*(aa+bb)** (using 7 and 5) 3
9. The expression **(a+b)*(aa+bb)(a+b)*** (using 8 and 7) 3

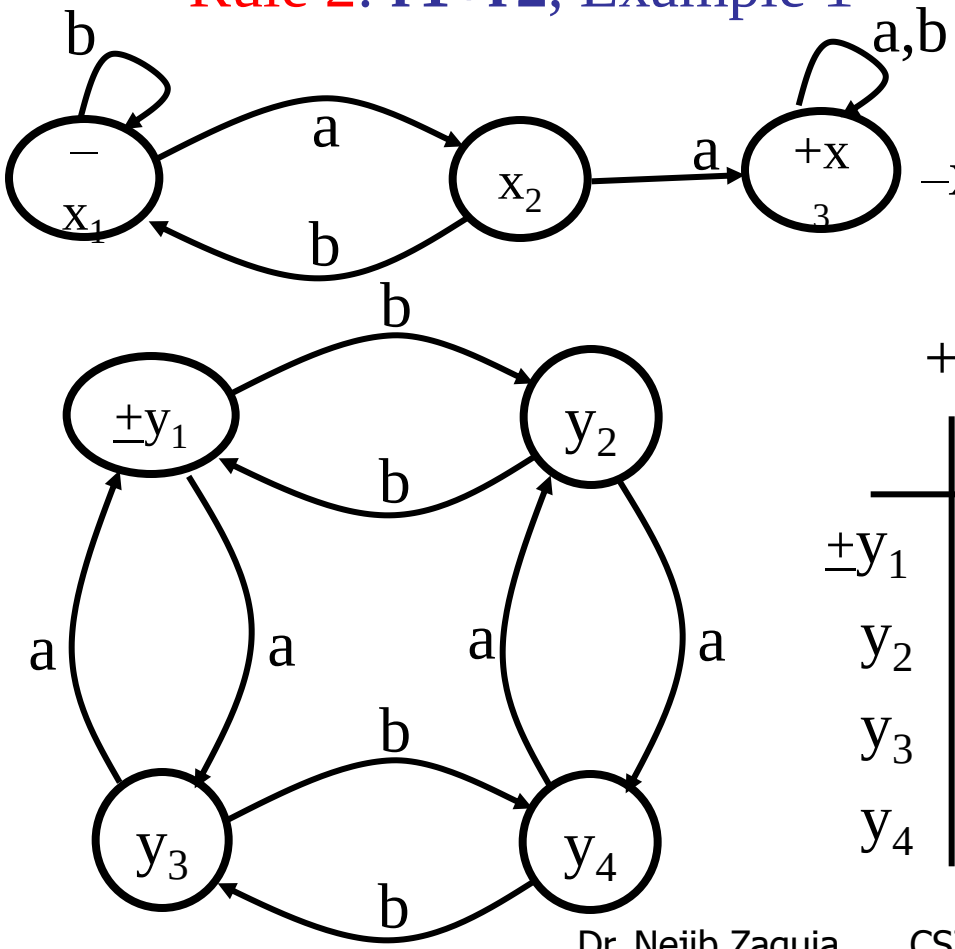
Rule 1



The language Λ



Rule 2: **r1+r2**, Example 1

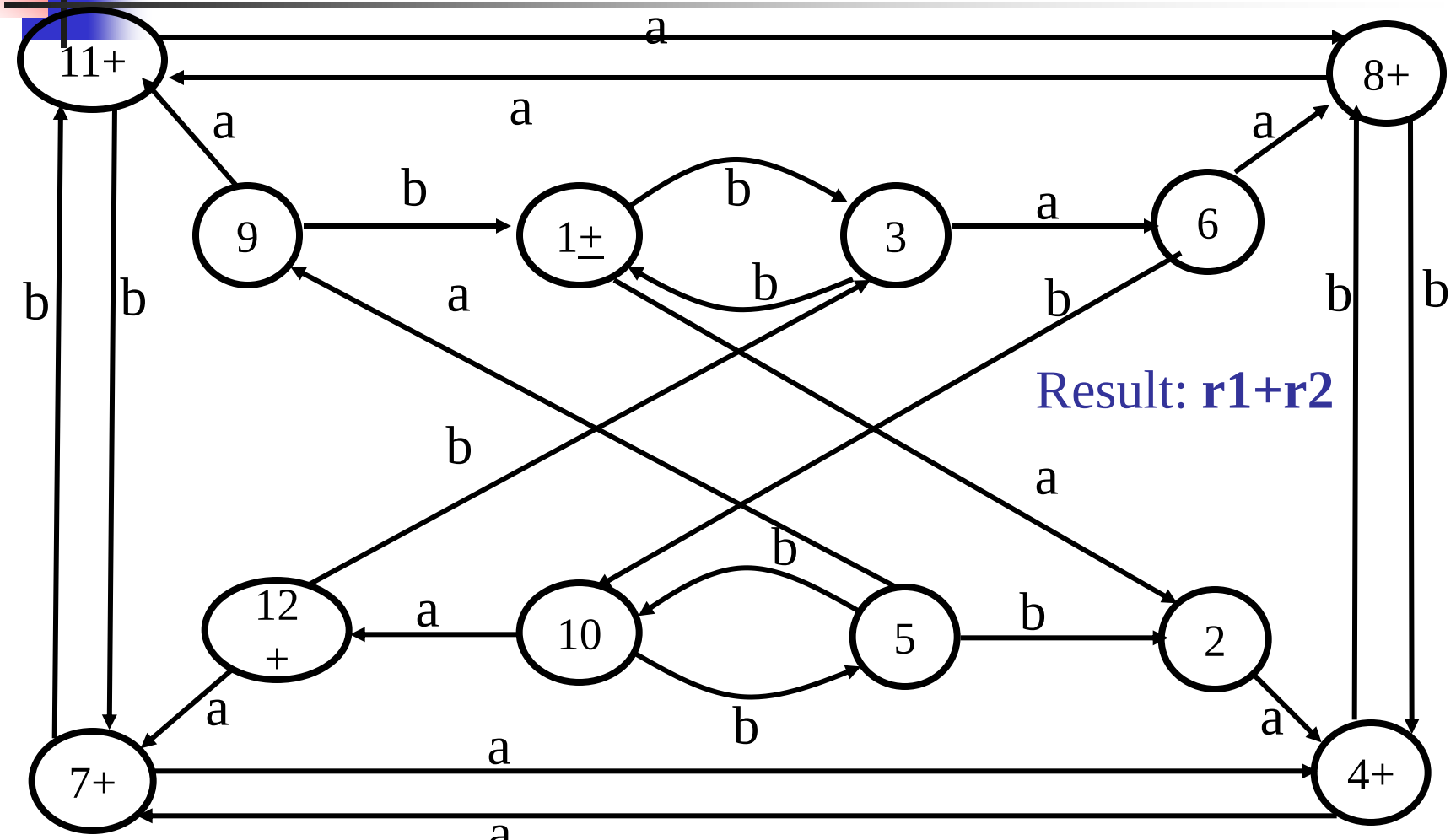


All word containing aa

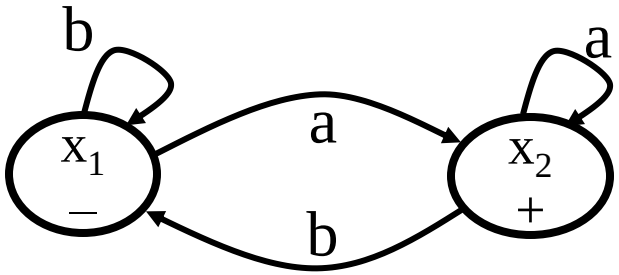
	a	b
$-x_1$	x_2	x_1
x_2	x_3	x_1
$+x_3$	x_3	x_3

	a	b
$\pm y_1$	y_3	y_2
y_2	y_4	y_1
y_3	y_1	y_4
y_4	y_2	y_3

EVEN-EVEN

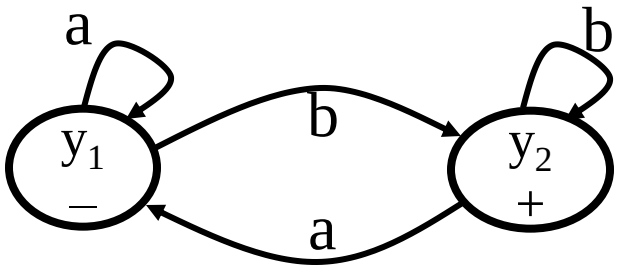

11+
8+
9
1+
3
6
12
+
10
5
2
7+
4+
a
a
a
a
b
b
b
b
b
b
b
b
b
b
a
a
a
a
a
a
a
Result: r1+r2

Example 2.



(words ending in a)

	a	b
$-x_1$	x_2	x_1
$+x_2$	x_2	x_1



(words ending in b)

	a	b
$-y_1$	y_1	y_2
$+y_2$	y_1	y_2

Lemma 3: Every language that can be defined by a regular expression can also be defined by a finite automaton.

Proof: By constructive algorithm starting from the recursive definition of regular expressions

1. There is an FA that accepts only the empty word (Λ) and an FA that accepts only a single letter.
2. If there is an FA that accepts the language defined by $\mathbf{r_1}$ and an FA that accepts the language defined by $\mathbf{r_2}$, then there is an FA that accepts the language $\mathbf{r_1+r_2}$.
3. If there is an FA that accepts the language defined by $\mathbf{r_1}$ and an FA that accepts the language defined by $\mathbf{r_2}$, then there is an FA that accepts the language defined by their concatenation $\mathbf{r_1r_2}$.
4. If there is an FA that accepts the language defined by $\mathbf{r}$ then there is an FA that accepts the language defined by $\mathbf{r^*}$.

Thus for every regular expression, we can construct a FA.

Algorithm 1: r_1+r_2

Input:

FA 1: alphabet: Σ states: $x_1, x_2, x_3, \ldots$ start state: x_1

FA 2: alphabet: Σ states: $y_1, y_2, y_3, \ldots$ start state: y_1

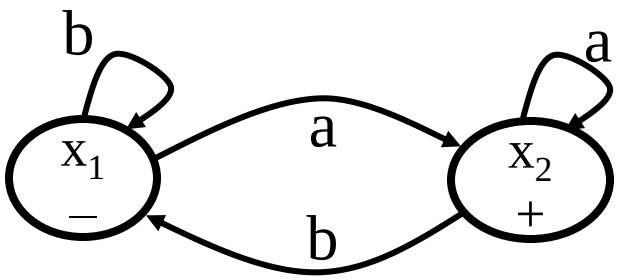
plus final states and transitions

The new FA:

alphabet: Σ states: $z_1, z_2, z_3, \ldots$ start state: x_1 or y_1

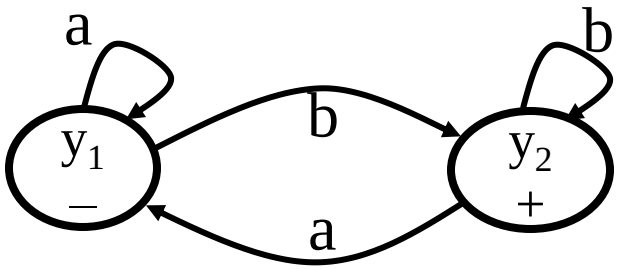
transitions: if $z_i = x_j$ or y_k and $x_j \rightarrow x_{new}$ and $y_k \rightarrow y_{new}$ (for input p) then $z_{new} = (x_{new}$ or $y_{new})$ for input p.

If x_{new} or y_{new} is a final state, then z_{new} is a final state.



(words ending in a)

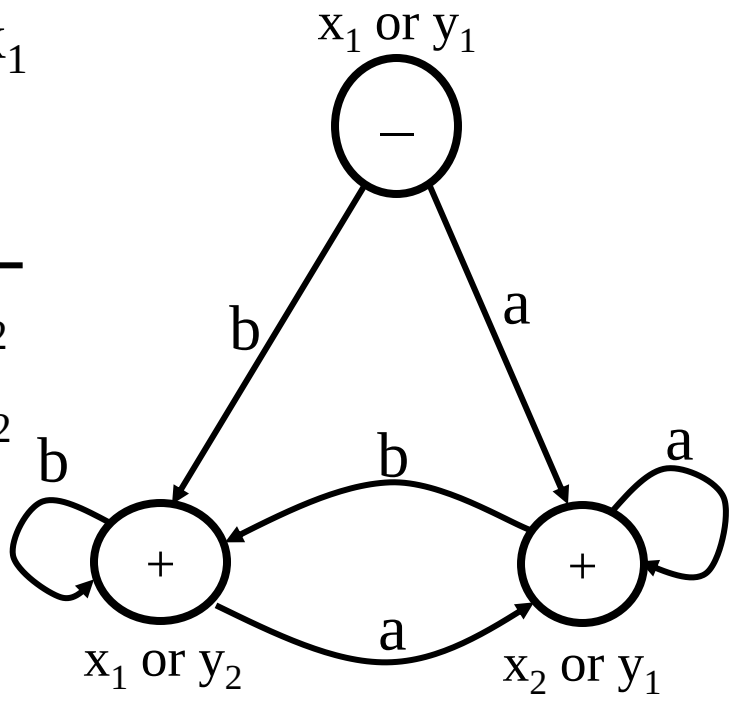
	a	b
$-x_1$	x_2	x_1
$+x_2$	x_2	x_1



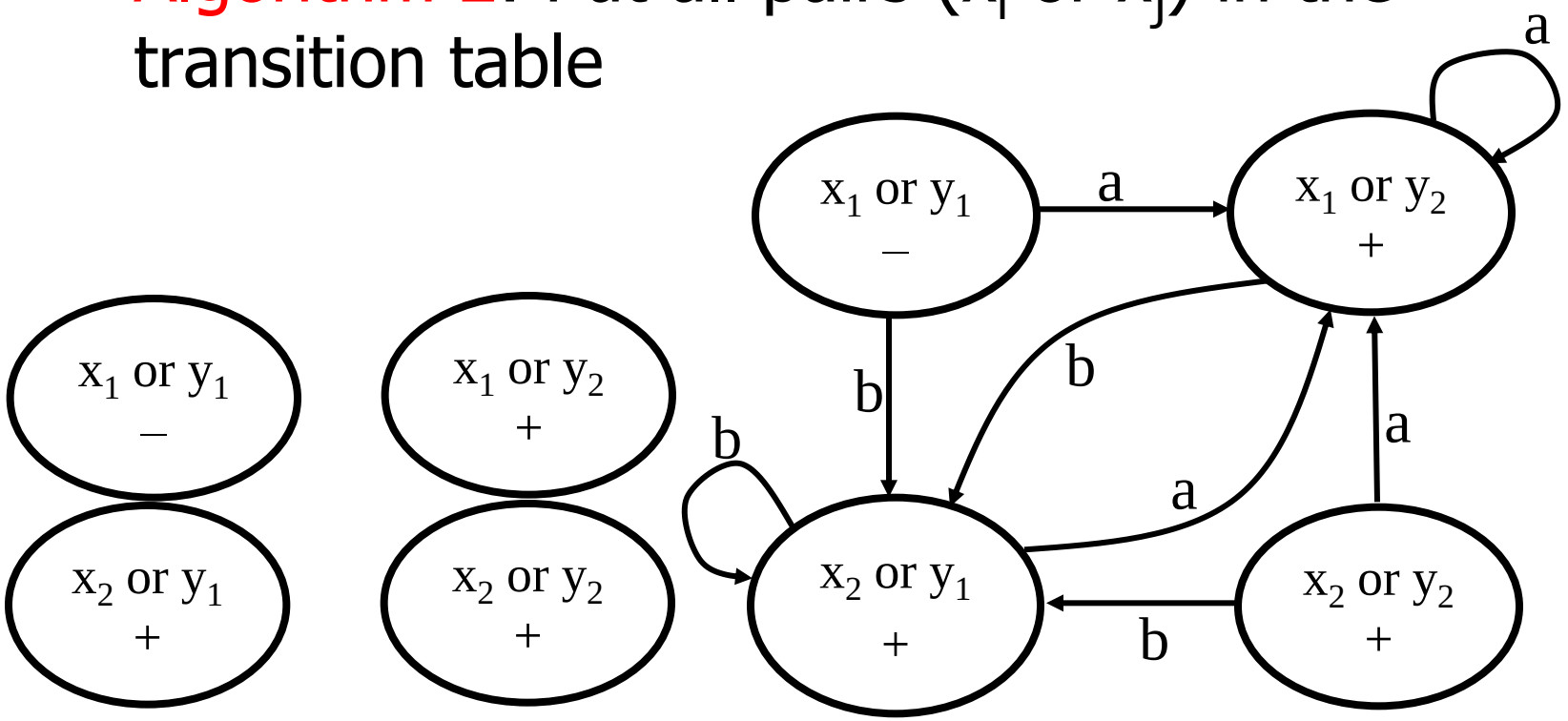
(words ending in b)

	a	b
$-y_1$	y_1	y_2
$+y_2$	y_1	y_2

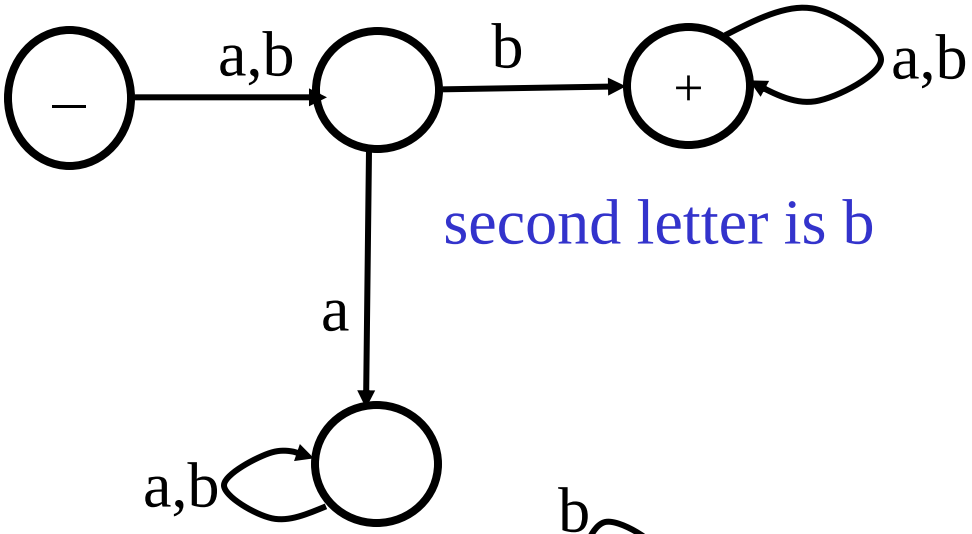
algorithm 1



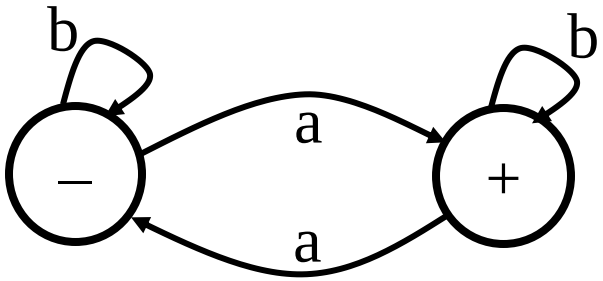
- Algorithm 2: Put all pairs (x_i or x_j) in the transition table



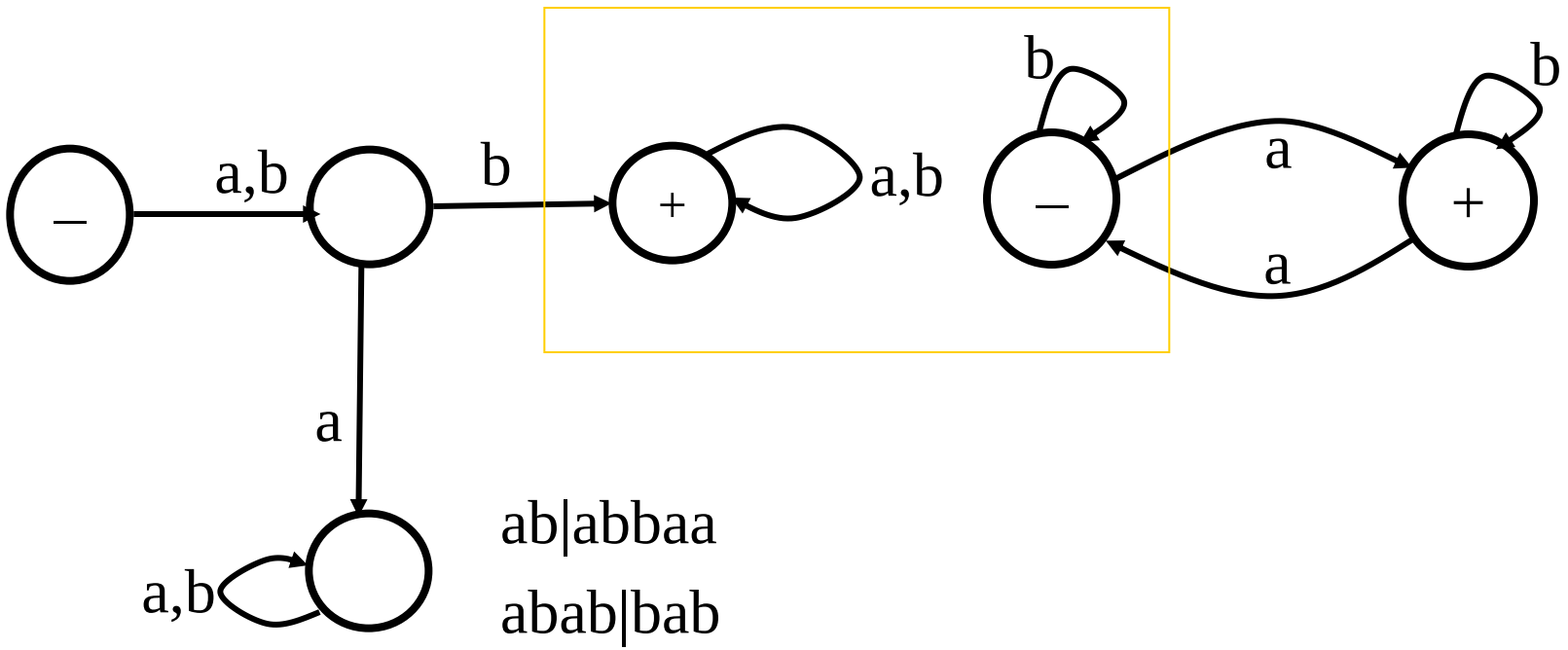
Rule 3: **r1r2**, Example 1



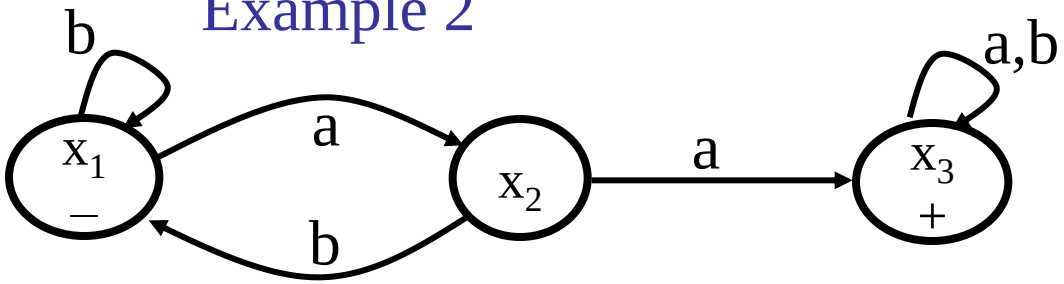
second letter is b



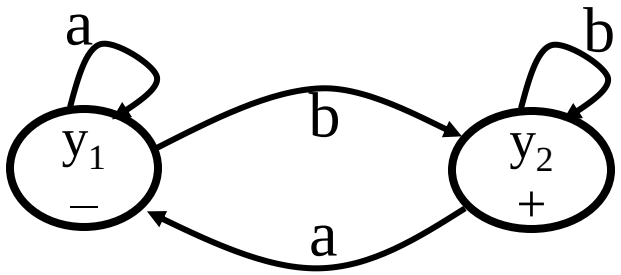
odd number of a's


a,b
b
a,b
b
b
a
a
a
a,b
ab|abbaa
abab|bab

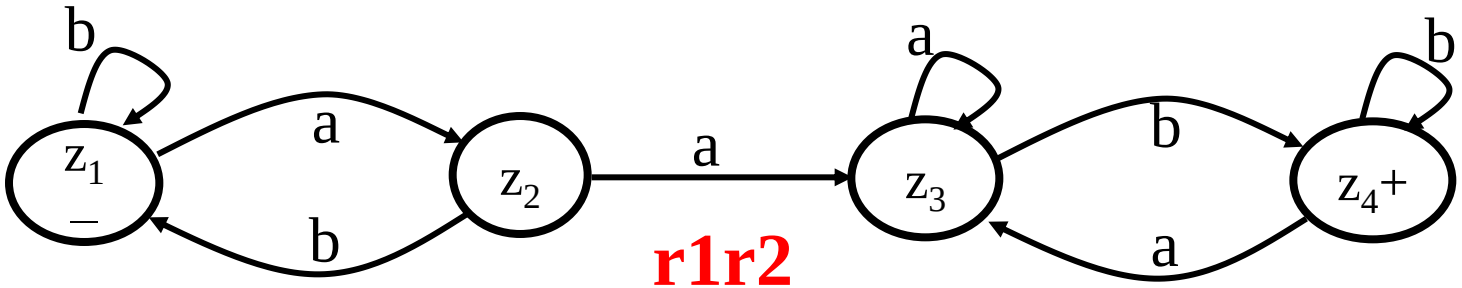
Example 2



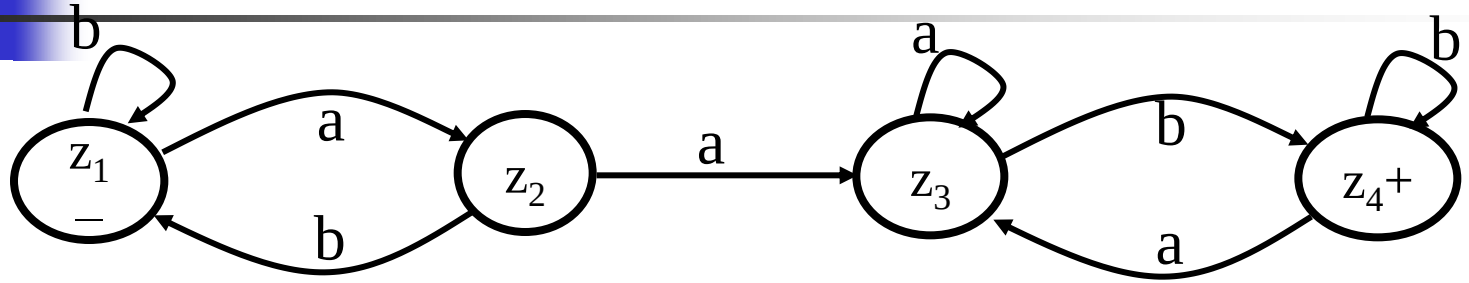
r_1: all with aa



r_2: words ending in b



r1r2



We start by creating the states z1=x1and z2=x2

(z2,a)= x3 "we continue on AF1" OR
y1"we move to AF2, since x3 is a final state in AF1"

(z2, a) = x3 or y1= z3

(z3, a) = x3 or y1= z3
(z3, b) = x3 or y1 or y2 = z4 +

(z4, a)= x3 or y1=z3
(z4, b)=x3 or y1 or y2 = z4 +

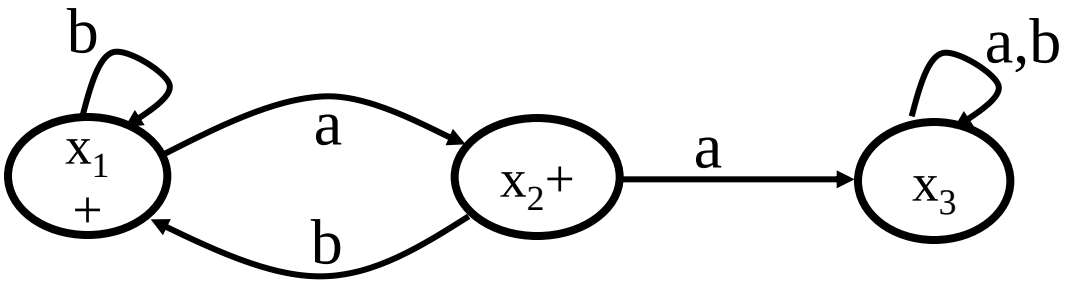
Rule 3: Concatenation: Summary of the Algorithm

1. Add a state z for every state of the first automaton that is possible to go through before arriving at a final state.
2. For each final state x, add a state z = (x or y_1), where y_1 is the start state of the second automaton.
3. Starting from the states added at step 2, add states:

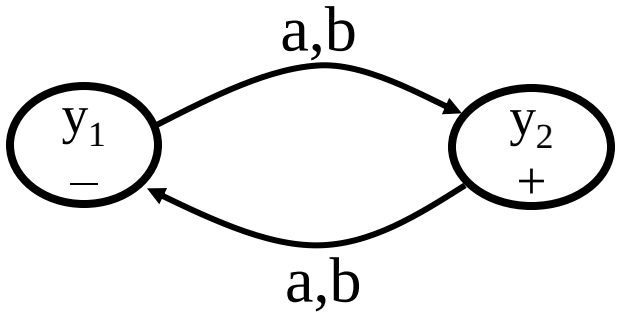
$$z = \begin{cases} x \text{ (state such that execution continues on } 1^{st} \text{ automaton)} \\ \text{OR} \\ y \text{ (state such that execution moves on } 2^{nd} \text{ automaton}\} \end{cases}$$

4. Label every state that contains a final state from the second automaton as a final state.

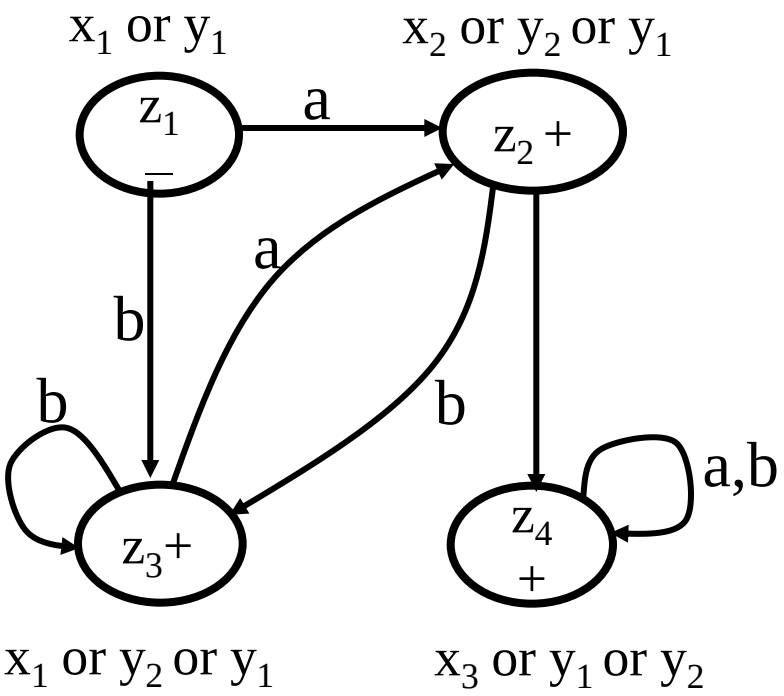
Example 3



r_1: no double a

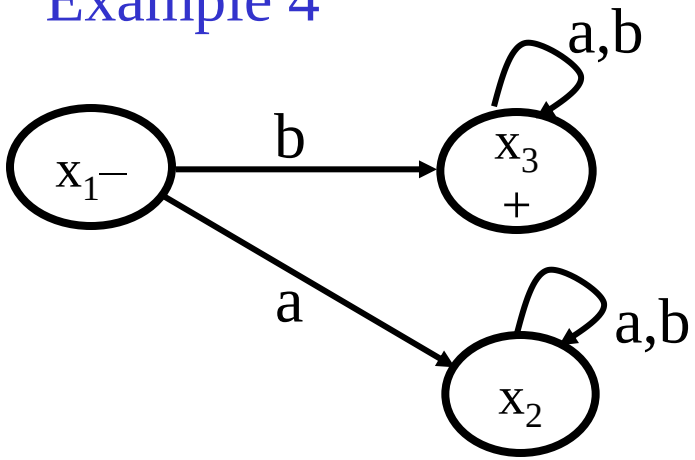


r_2: odd number of letters

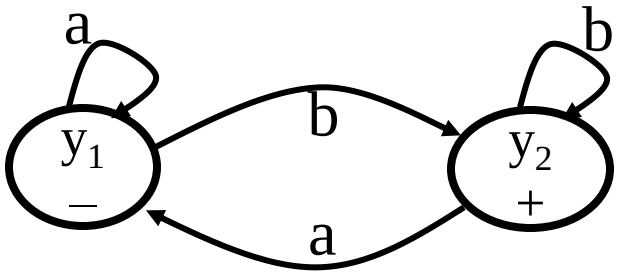


$\mathbf{r_1 r_2}$: all words except Λ

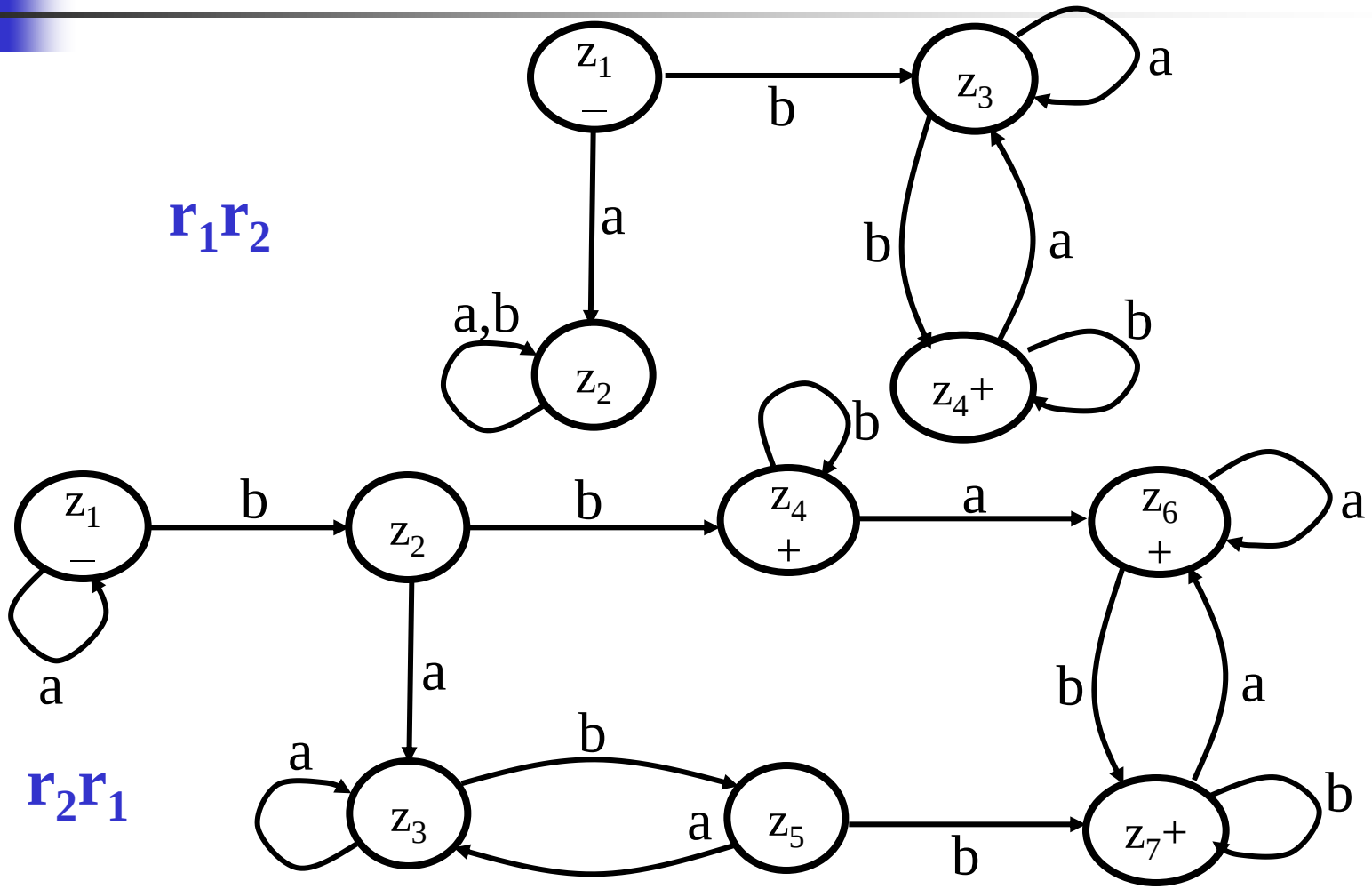
Example 4



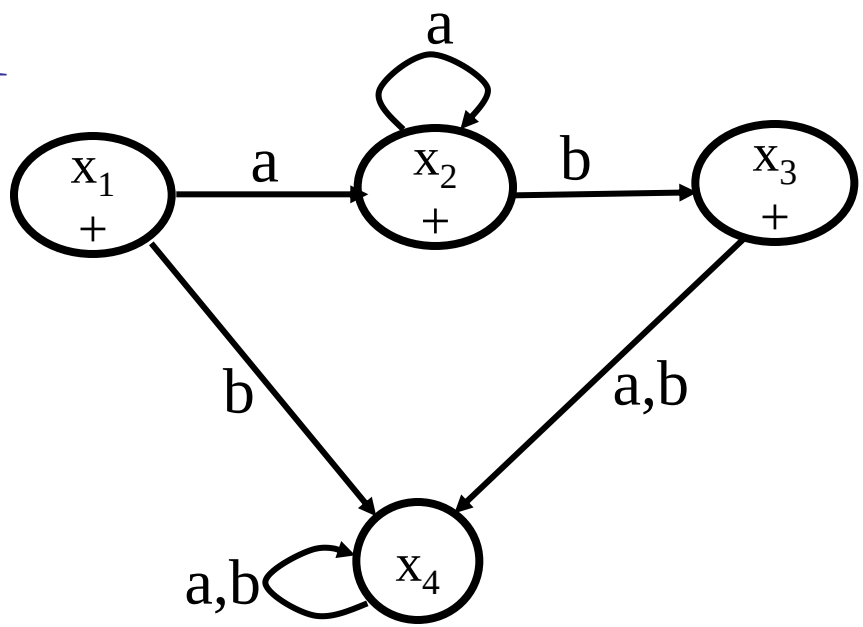
$\mathbf{r_1}$: words starting with b



$\mathbf{r_2}$: words ending in b


r_1r_2
z_1
–
b
z_3
a
a
b
a
b
a,b
z_2
z_4+
b
z_1
–
b
z_2
b
z_4
+
a
z_6
+
a
a
a
b
a
r_2r_1
a
b
z_3
a
z_5
b
z_7+
b

Rule 4: **r***, Example 1



r: **a* + aa*b**

r*: words without double b, and that do not start with b.

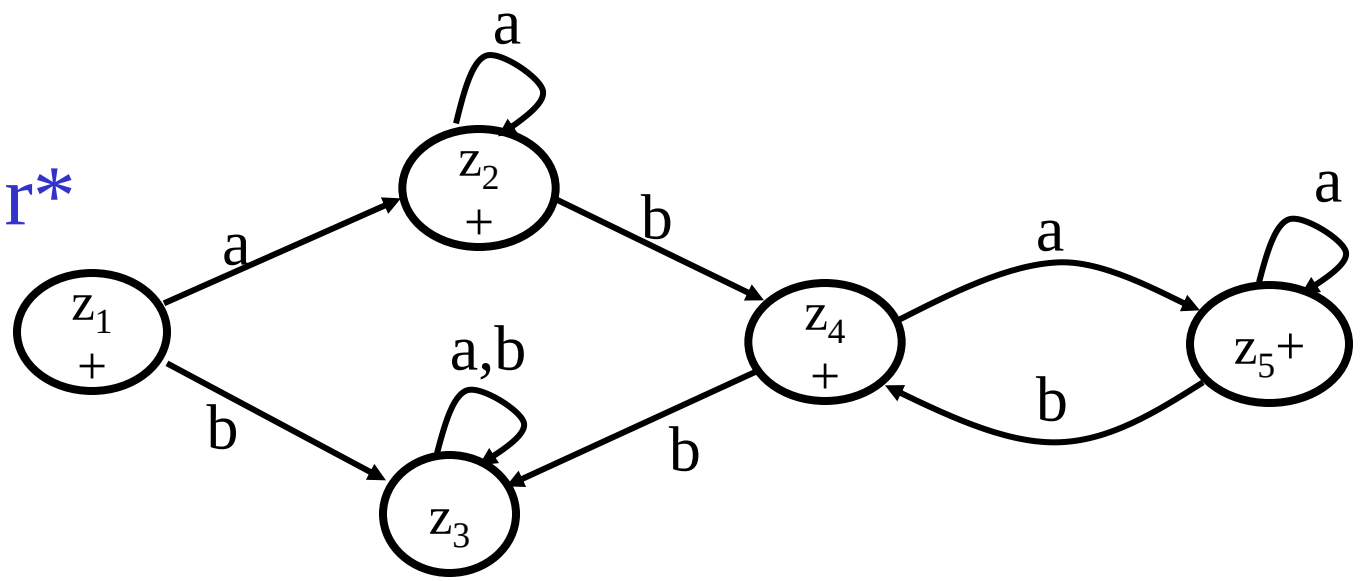
z1=x1 ± (z1, a) = x1 or x2= z2 + (z1,b) = x4= z3

(z2,a)= x1 or x2 =z2 (z2, b)= x1 or x3 or x4= z4 +

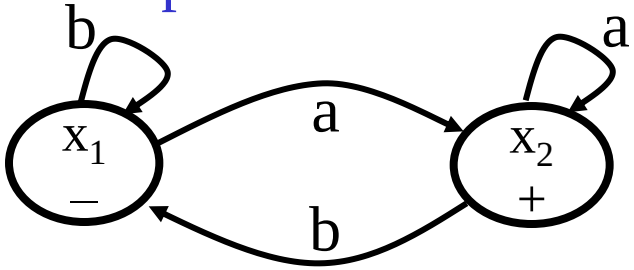
(z3, a)=z3 (z3,b) = z3

(z4,a)= x1 or x2 or x4= z5 + (z4, b)= x4 = z3

(z5,a)= x1 or x2 or x4= z5 (z5, b)= x1 or x3 or x4= z4

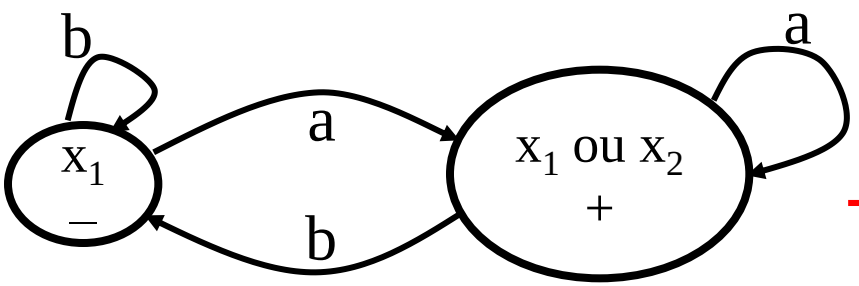


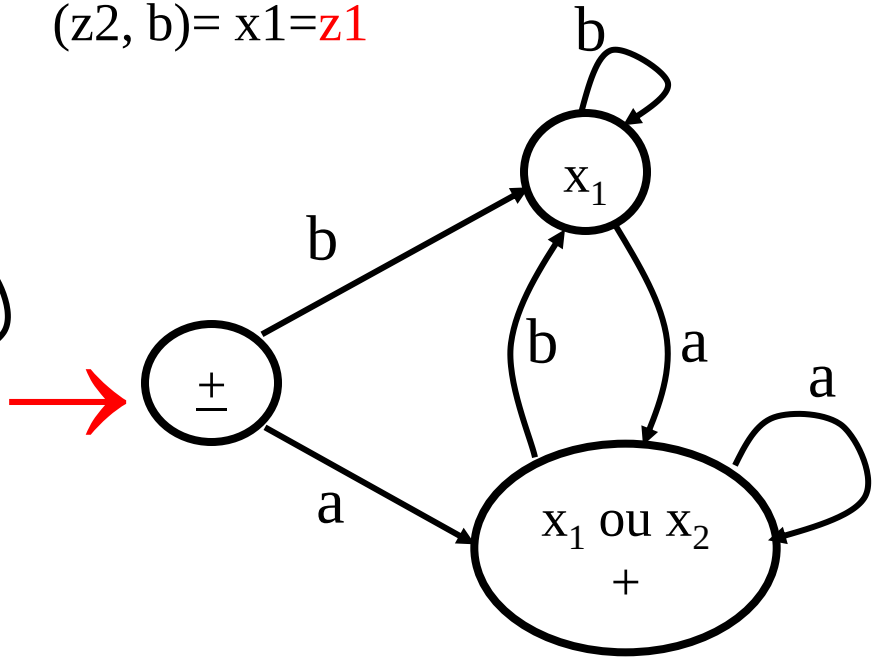
Example 2



z1=x1 –
(z1, a) = x1 ou x2= z2 +
(z1,b) = z1
(z2,a)= x1 ou x2 =z2
(z2, b)= x1=z1

Words ending in a





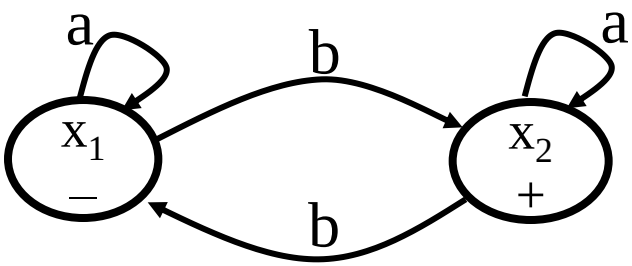
Problem with Λ

Rule 4: Kleene Star: Algorithm

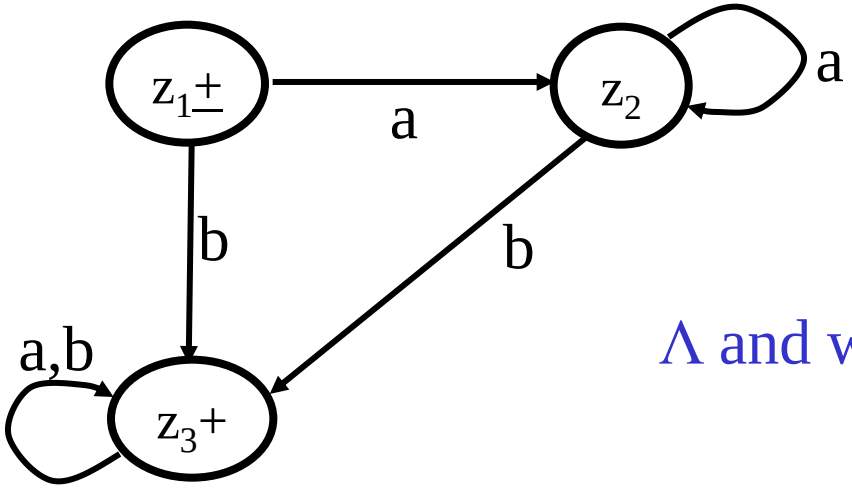
Given: an FA whose states are $\{x_1, x_2, x_3, \ldots\}$

- For every subset of states, create a state of the new FA. Remove any subset that contains a final state but not the start state.
- Make the transition table for all the new states.
- Add a $\pm$ state. Connect it to the same states as the original start state was connected to using the same transitions.
- The final states must be those that contain at least one final state from the original FA.

Example 3



Words with an odd number of b's.

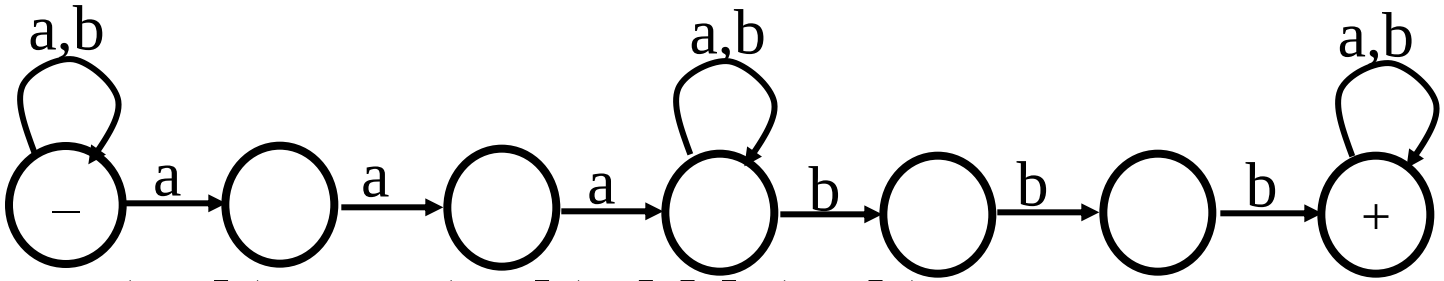


Λ and words with at least one b.

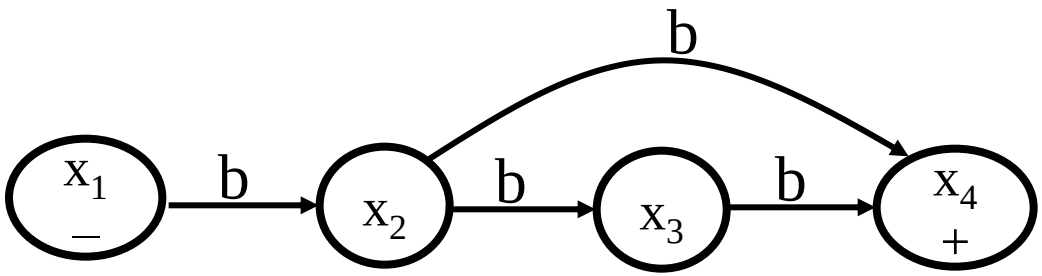
- A nondeterministic finite automaton (NFA) is:
  1. a finite set of states, one of which is designated as the start state, and some (maybe none) of which are designated the final states
  2. an alphabet Σ of input letters
  3. a finite set of transitions that show how to go to a new state, for some pairs of state and letters

Remark: Every finite automaton is a nondeterministic finite automaton. Every nondeterministic finite automaton is a transition graph.

2 Examples of Nondeterministic Finite Automata



(a+b)*aaa(a+b)*bbb(a+b)*



bb+bbb

Theorem: Every language that can be defined by a nondeterministic finite automaton can also be defined by a deterministic finite automaton.

Proof (1): Every nondeterministic finite automaton is a transition graph.

By lemma 2: transition graph → regular expression

By lemma 3: regular expression → finite automaton

Proof (2): By constructive algorithm (See Manuel page 135)

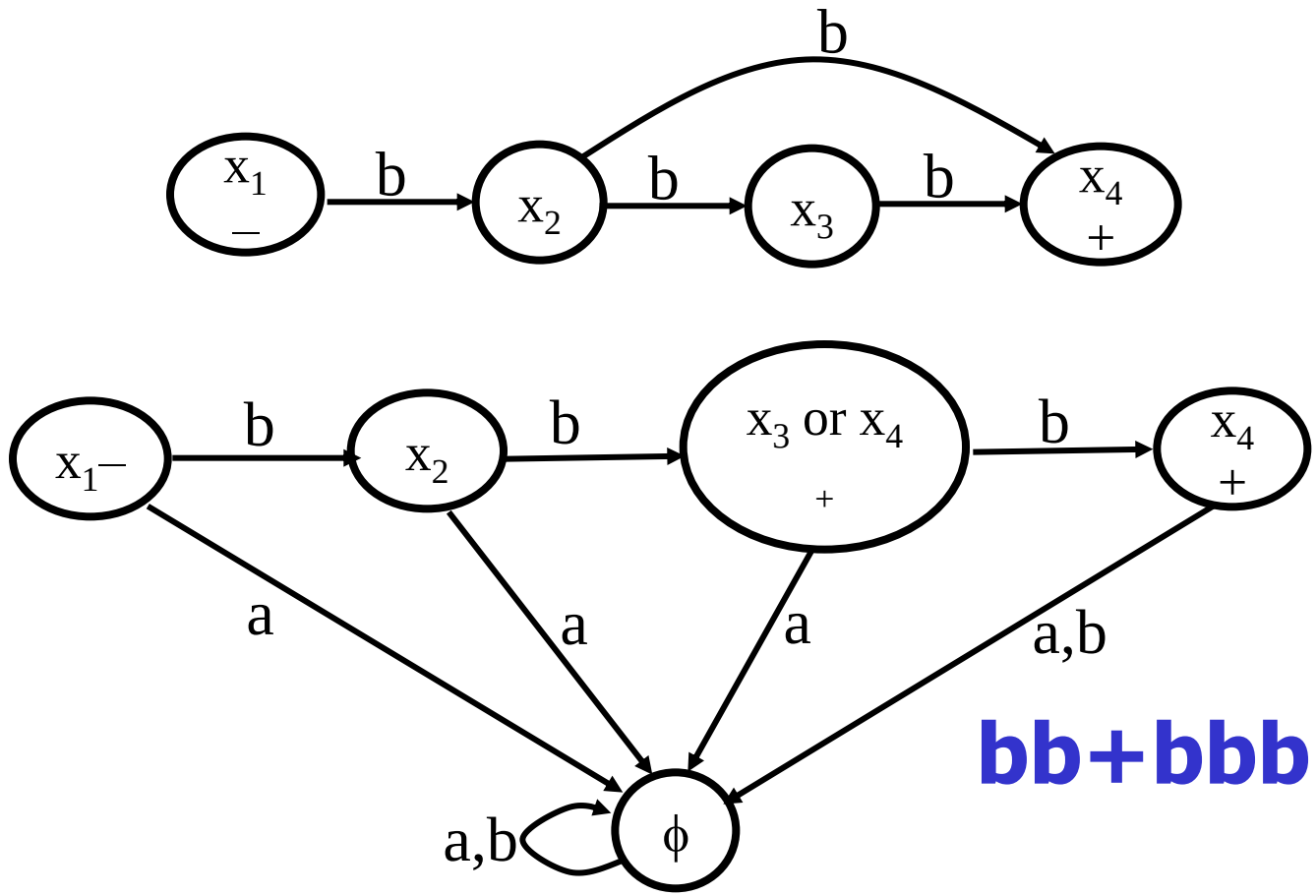
Chapter 7: Kleene's Theorem

Algorithm: nondeterministic automaton → deterministic automaton (FA)

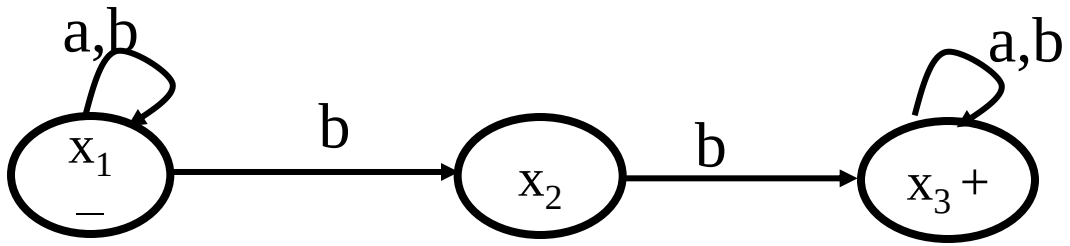
Given: a nondeterministic automaton whose states are

$\{x_1, x_2, x_3, \ldots\}$

1. For every subset of states, create a state of the new FA.
2. Make the transition table for all the new states (or just the new states that can be entered).
3. Add a state ϕ. Add transitions that loop back to itself for all letters of the alphabet. For each new state, if there is no transition for letter p, add one that goes to the ϕ state.
4. The final states must be those that contain at least one final state from the original nondeterministic finite automaton.

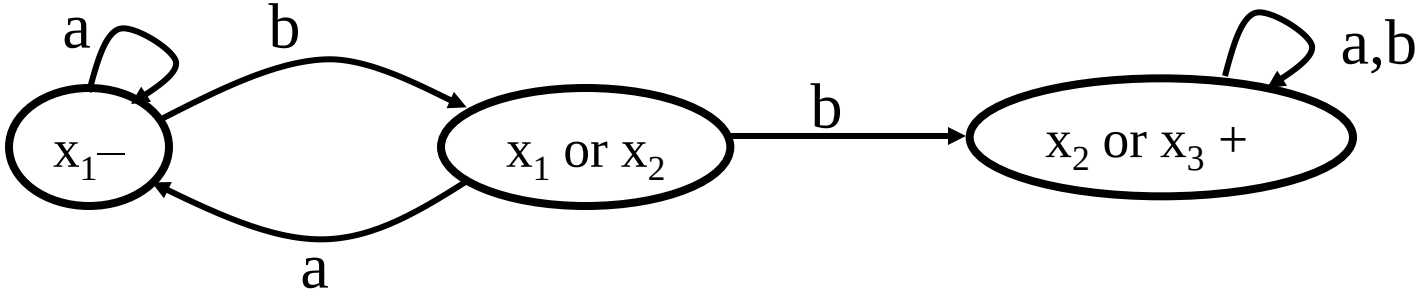


bb+bbb



(x1,a)=x1
(x1 or x2, a)=x1
(x2 or x3,a)=x2 or x3

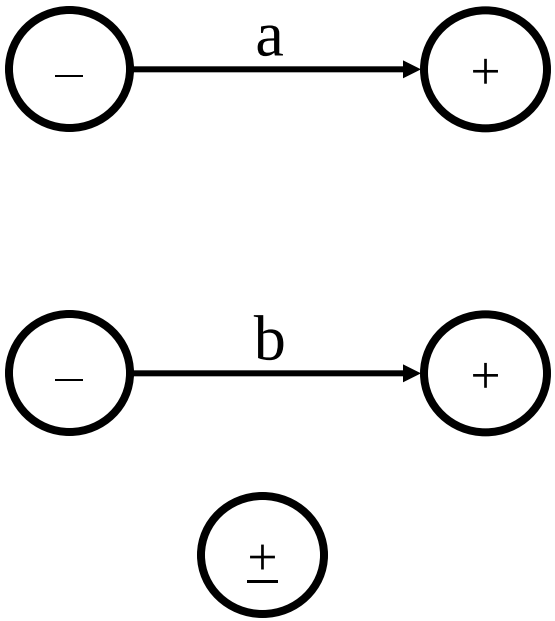
(x1,b)=x1 or x2
(x1 or x2,b)=x2 or x3
(x2 or x3,b)=x2 or x3



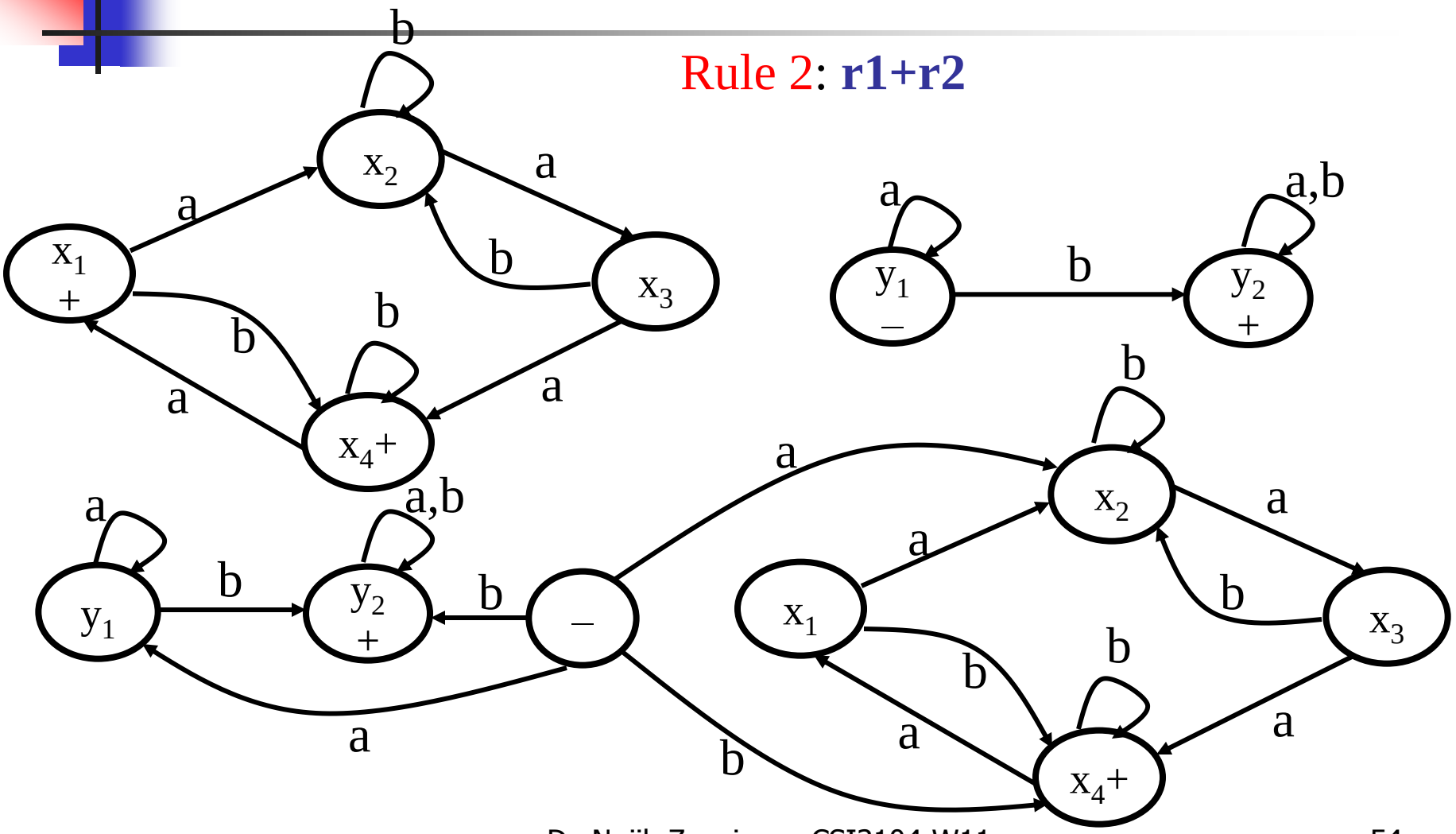
Lemma 3: Every language that can be defined by a regular expression can also be defined by a finite automaton.

Proof: By constructive algorithm starting from the recursive definition of regular expressions, we build a nondeterministic finite automaton. Then, by the most recent theorem, it is possible to then build a finite automaton.

Rule 1: Λ and the letters in Σ



Rule 2: **r1+r2**


b
x2
a
a
x1
+
b
x3
b
b
a
a
x4+
a
a,b
y1
b
y2
+
a
a,b
y1
b
y2
+
b
–
a
b
a
b
x2
a
a
x1
b
x3
b
b
a
a
x4+

Rule 3: **r1r2**