CS375: Logic and Theory of Computing

Fuhua (Frank) Cheng

Department of Computer Science

University of Kentucky

Table of Contents:

- **Week 1: Preliminaries (set algebra, relations, functions) (read Chapters 1-4)**
- **Weeks 2-5: Regular Languages, Finite Automata (Chapter 11)**
- **Weeks 6-8: Context-Free Languages, Pushdown Automata (Chapters 12)**
- **Weeks 9-11: Turing Machines (Chapter 13)**

Table of Contents (conti):

- **Weeks 12-13: Propositional Logic (Chapter 6), Predicate Logic (Chapter 7), Computational Logic (Chapter 9), Algebraic Structures (Chapter 10)**

6. Regular Languages & Finite Automata

- Regular Language Topics

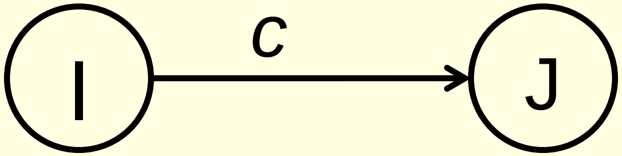
What is the function of a DFA/NFA?

1. A device to recognize a regular language

2. Can also be used as a device to generate a regular language

Why?

For an edge ‘*a’* between two states ‘I‘ and ‘J’ of a given DFA/NFA,



a “path” between I and J through c is represented as follows:

$$I \rightarrow cJ$$

I is called the *start point*, J is called the *end point*. I and J could be the same if edge c is a loop of state I.

For each final state ‘F’ of the DFA/NFA, define an “empty path” as follows:

$$F \rightarrow \Lambda$$

‘F’ is both the *start point* and *end point* of this path.

Two paths can be merged into a single path if start point of the second path is the same as the end point of the first path. So for the following two paths,

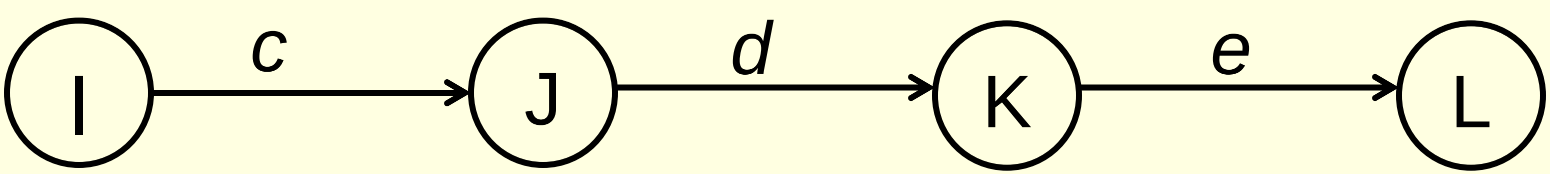
$$I \rightarrow cJ \qquad\qquad J \rightarrow dK$$

after merge, we get a path between 'I' and 'K' through c and d, represented as follows:

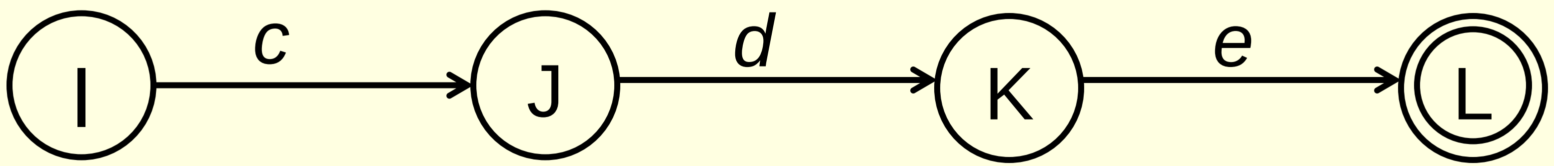
$$I \rightarrow cdK$$

Several paths can be merged to form longer path. The path representation for the following case is

$$\boxed{I \rightarrow cdeL}$$



In the following case if L is a final state,



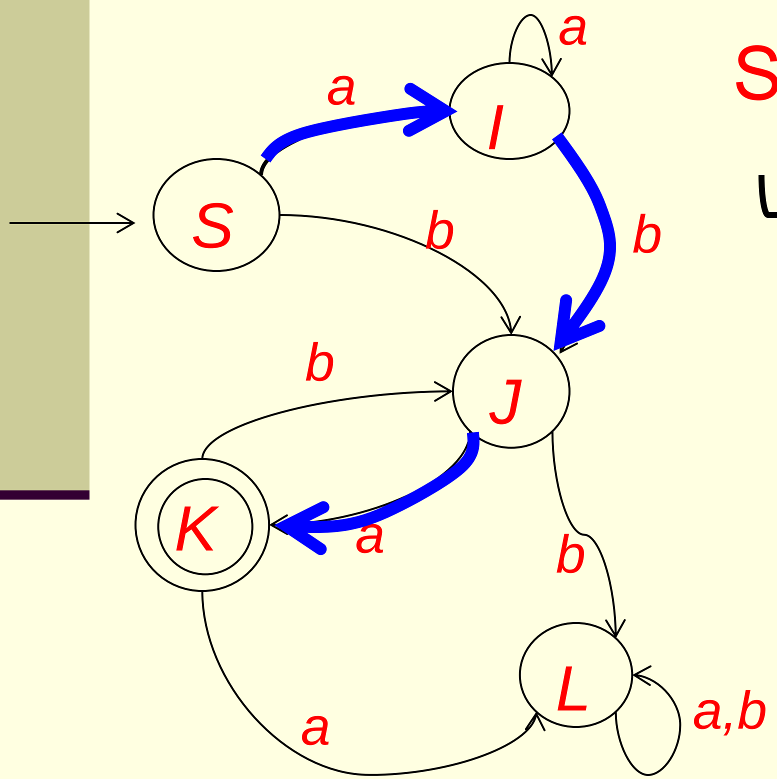
then we can merge the path $I \rightarrow cdeL$ with the empty path $L \rightarrow \Lambda$ to get a path represented as follows:

$$I \rightarrow cde$$

The right hand side of the path representation is a string only.

A string can be generated by a DFA/NFA if there is a path from the start state of the FA to a final state such that the right hand side of the path representation is that string only.

For the following DFA, the string 'aba' can be generated by this DFA because aba is the right hand side of the path representation $S \rightarrow aba$ constructed as follows:



$S \rightarrow aI$; $I \rightarrow bJ$; $J \rightarrow aK$; $K \rightarrow \Lambda$

$S \rightarrow abJ$

$S \rightarrow abaK$

$S \rightarrow aba$

6. Regular Languages & Finite Automata

- Regular Language Topics

How should this function of a DFA/NFA be characterized ?

Use a ***regular grammar***

6. Regular Language… & Fi… A…

- Regular Language Topic…

> *A grammar is like a life creating and growing mechanism*

Regular languages can be charact… d by regular expressions, by DFAs, by NFAs, as well as b… ***regular grammars.***

RECALL: what is a grammar?

a set of rules used to define the structure of the strings in a language, usually represented as a 4-tuple

$G = (N, T, S, P)$

N : alphabet of nonterminals (uppercase letters)

T : alphabet of terninals (lowercase letters)

S : start symbol (nonterminal)

P : set of productions

6. Regular Languages & Finite Automata

- Regular Language Topics

Example.

$N = \{S\}$,   T = {a, b, c},   S: start symbol,

$$\boxed{\begin{aligned} P:\ & S \to \Lambda \\ & S \to aS \\ & S \to bS \\ & S \to cS \end{aligned}}$$

then the grammar can be represented by the 4-tuple

$G = (\{S\}, \{a, b, c\} , S, P)$

6. Regular Languages & Finite Automata

- Regular Language Topics

Back to regular grammars -

A regular grammar is a grammar whose productions take the following form:

$$\boxed{A \rightarrow wB} \qquad \text{or} \qquad \boxed{A \rightarrow w}$$

where w is a string of terminals and A and B are nonterminals.

So we can have productions like:

$A \rightarrow \Lambda$

$A \rightarrow w$ (w: not empty)

$A \rightarrow B$ (A and B can be the same)

$A \rightarrow wB$ (w: not empty)

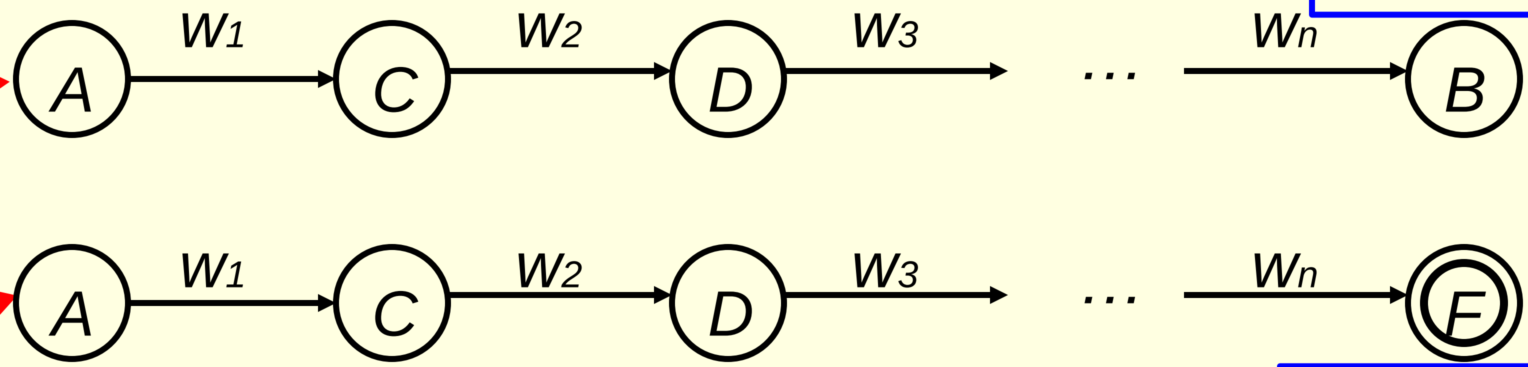
Relationship between an RG and an NFA:

A regular grammar is a grammar whose productions take the following form:

$A \rightarrow wB$ or $A \rightarrow w$

where w is a string of terminals.

$w = w_1w_2w_3\ldots w_n$



B doesn't have to be a final state

F nust be a final state

A regular grammar is a grammar whose productions take the following form:

$$\boxed{A \to wB} \quad \text{or} \quad \boxed{A \to w}$$

where w is a string of terminals.

'Is a regular language context dependent?

'Regular' in what sense?

Why do not need a rule of the forms: $A \to Bw$ *?*

Because for a string $w=w_1w_2w_3$ *generated by the above rule:*

$$B \to Bw_3 \to Bw_2w_3 \to Bw_1w_2w_3 \to \Lambda w_1w_2w_3 \to w_1w_2w_3$$

one can get it by:

$$A \to w_1A \to w_1w_2A \to w_1w_2w_3A \to w_1w_2w_3\Lambda \to w_1w_2w_3$$

6. Regular Languages & Finite Automata

- Regular Language Top

a *and b should have separate/ independent breeding lines*

Example. *Construct a* regular gram the language of

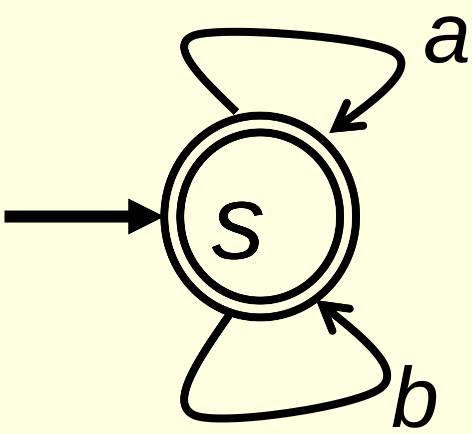
{ Λ, a, b, aa, bb, aaa, b … , a^n, b^n, ...}

-

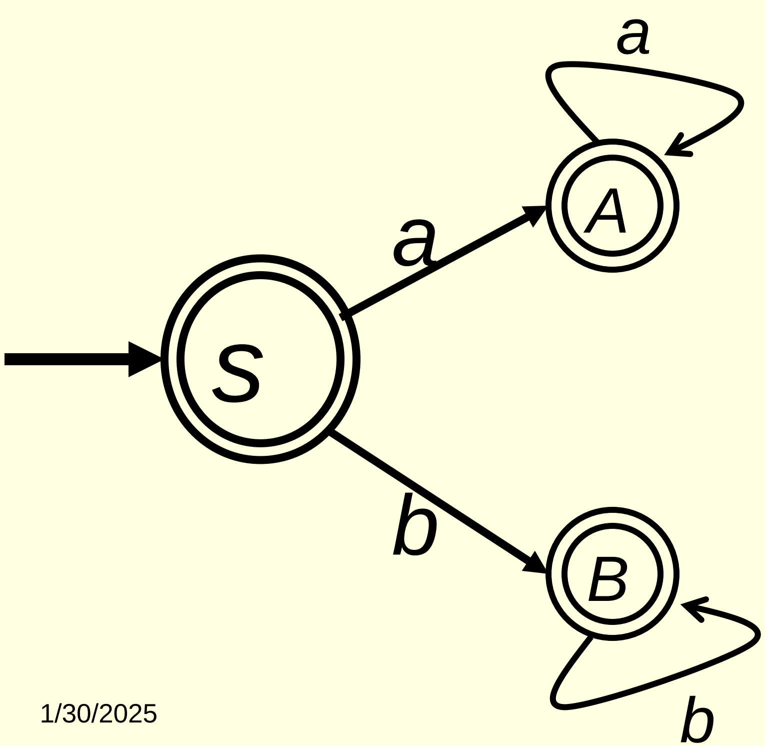
Would $S \rightarrow \Lambda \mid aS \mid bS$ *work?*

$S \rightarrow \Lambda$; $S \rightarrow aS$; $S \rightarrow bS$ No

$S \rightarrow \Lambda | aA | bB$; $A \rightarrow \Lambda | aA$; $B \rightarrow \Lambda | bB$ Yes



$$S \to \Lambda\,;$$
$$S \to aS\,; \quad S \to bS$$



$$S \to \Lambda|aA|bB\,;$$
$$A \to \Lambda|aA\,; \;\; B \to \Lambda|bB$$

6. Regular Languages & Finite Automata

- Regular Language Topics

Example. Write a regular grammar for

{ab, acb, accb, acccb, accccb, …}.

Solution: A regular expression for the language is *ac*b.*

A regular grammar is

$S \to aT$

$T \to b \mid cT.$

Note that 'a' and 'b' only appear once

True?

$\boxed{ab:\ S \Rightarrow aT \Rightarrow ab}$

$\boxed{ac^2b:\ S \Rightarrow aT \Rightarrow acT \Rightarrow accT \Rightarrow accb \Rightarrow ac^2b}$

Would $S \to aS \mid b \mid cS$ work? **NO**

6. Regular Languages & Finite Automata

- Regular Language Topics

- If a letter, say c, is supposed to appear only once in a string, then we should not have a production of the following form:

$$S \rightarrow cS$$

- If a is not supposed to get involved in the growth of b or b is not supposed to get involved in the growth of a then we should not have productions of the following form:

$$S \rightarrow aS \mid bS$$

Transforming an NFA to a Regular Grammar

1. State names become the nonterminals. **Why?**

2. Edge symbols of the NFA become the terminals **Why?**

3. The start state becomes the start symbol of the grammar.

4. For each state transition from I to J labeled with x construct a production $I \to xJ$.

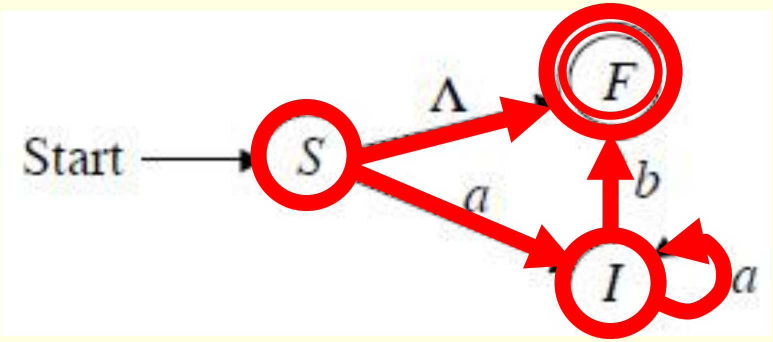
5. For each final state F construct a production $F \to \Lambda$.

if $x = \Lambda$ then simply $I \to J$

6. Regular Languages & Finite Automata

- Regular Language Topics

Example. Transform the following NFA to a regular grammar.



Non-terminals: S, F, I

Terminals: a, b

Start symbol: S

Productions : $S \rightarrow aI \mid F$

$I \rightarrow aI \mid bF$

$F \rightarrow \Lambda$

6. Regular Languages & Finite Automata

- Regular Language Topics

NFA ⇔ Regular Grammar

Regular Expression ⇔ NFA ⇔ Regular Grammar

Regular Expression ⇒ NFA ⇒ Regular Grammar

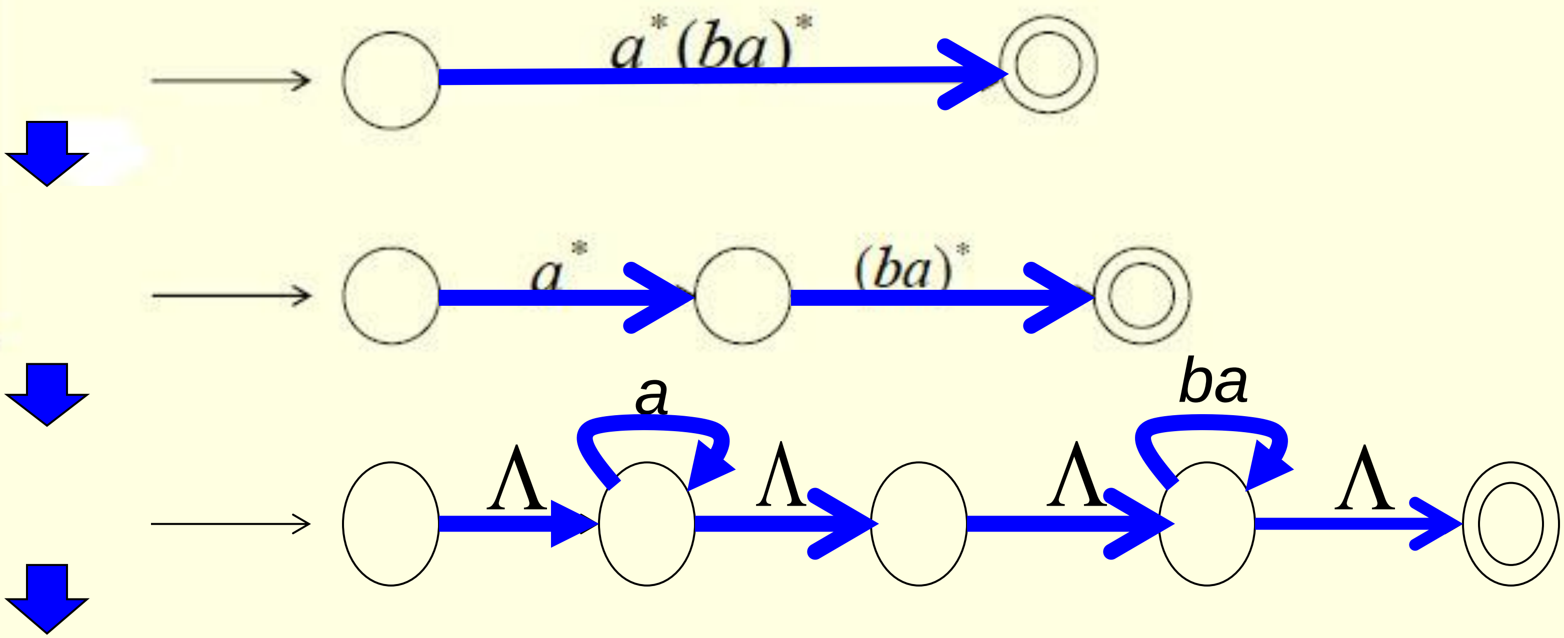
6. Regular Languages & ata

Compact regular grammar

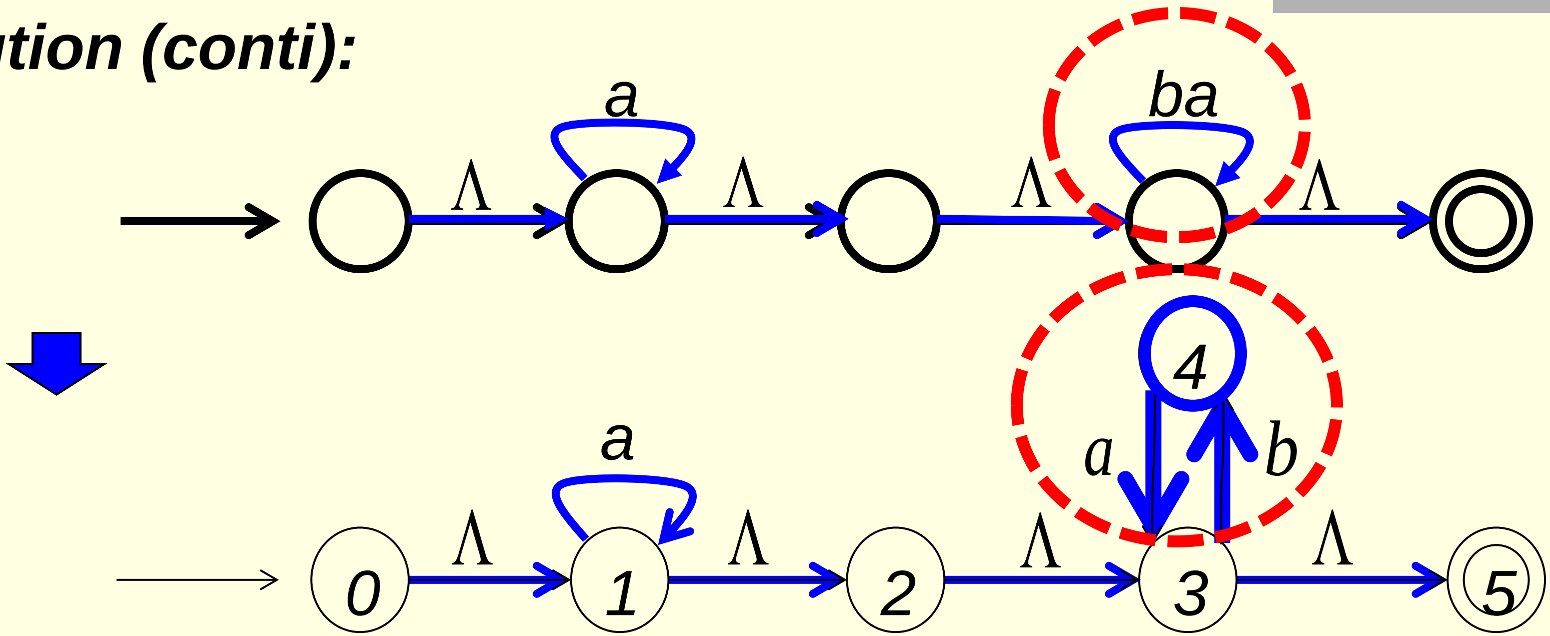
- Regular Language Topics

Example. Transform the regular expression $a^*(ba)^*$ into a regular grammar.

Solution: Draw an NFA and then use the algorithm.



Solution (conti):



Hence,

$$\Lambda(0) = \{0, 1, 2, 3, 5\} \qquad \Lambda(1) = \{1, 2, 3, 5\}$$

$$\Lambda(2) = \{2, 3, 5\} \qquad \Lambda(3) = \{3, 5\}$$

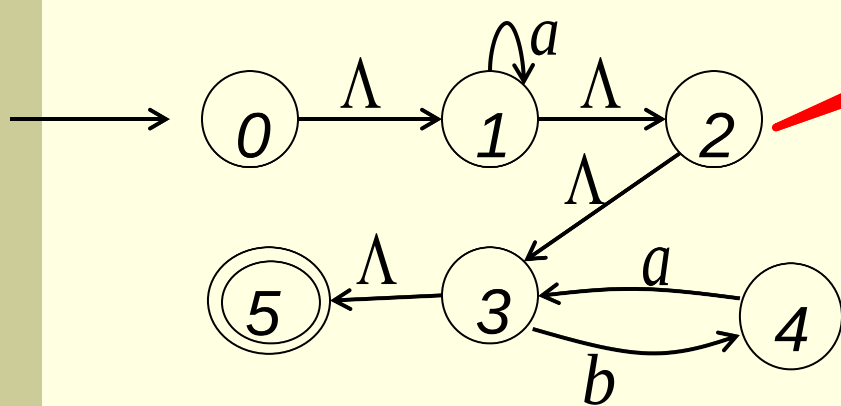
$$\Lambda(4) = \{4\} \qquad \Lambda(5) = \{5\}$$

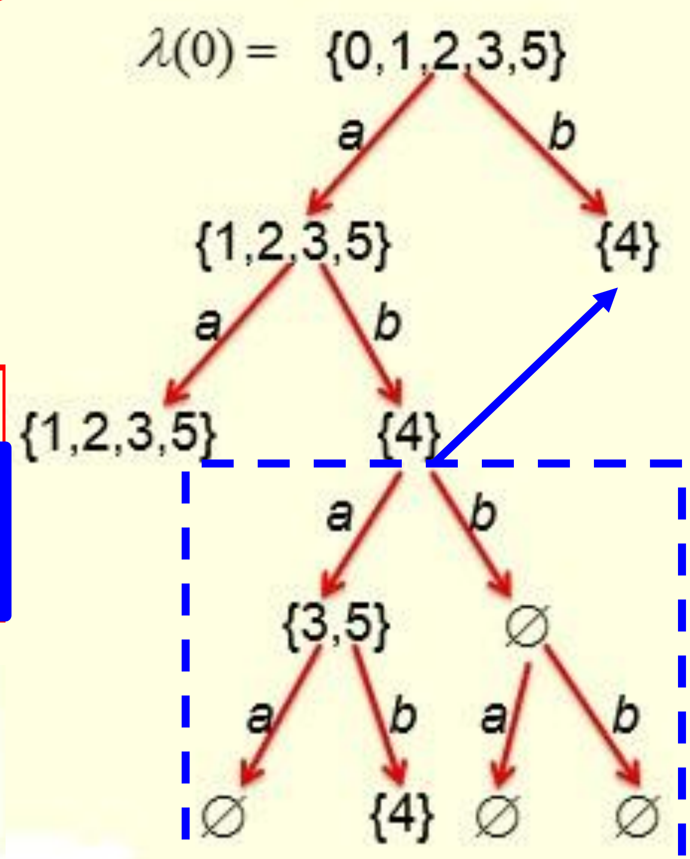
6. Regular Languages & Finite Automata

- Regular Language Topics

Can we convert this NFA to a regular grammar directly?

Solution: Draw an NFA and then use the algorithm.





Then,

1. build the tree on the right
2. identify all distinct nodes

{0,1,2,3,5}, {1,2,3,5}, {3,5}, ∅ , {4}

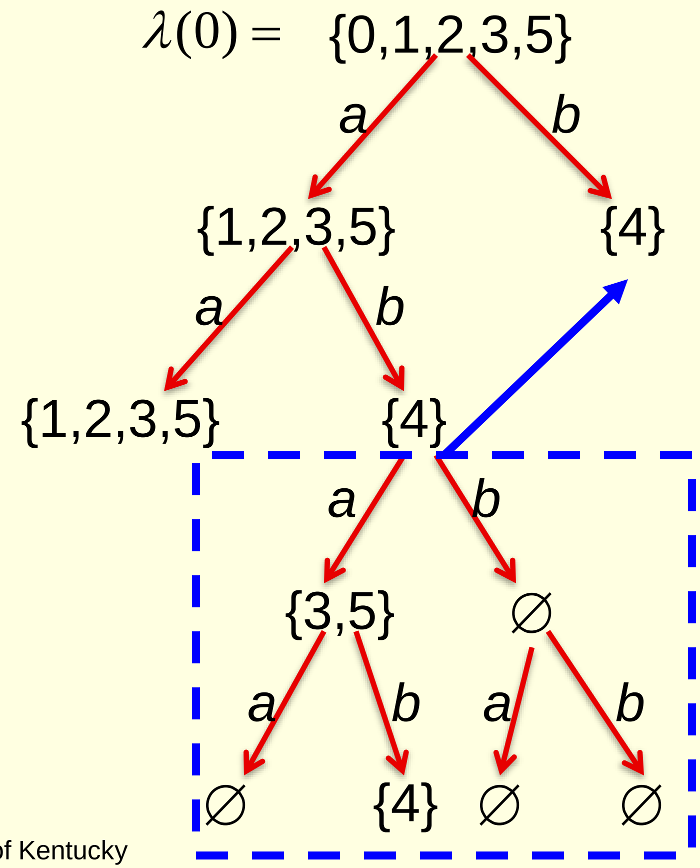
6. Regular Languages & Finite Automata

- Regular Language Topics

Solution: Draw an NFA and then use the algorithm.

Then,
1. build the transition table
2. and write it in simplified form

	T	a	b
S F	{0,1,2,3,5}	{1,2,3,5}	{4}
F	{1,2,3,5}	{1,2,3,5}	{4}
	{4}	{3,5}	∅
F	{3,5}	∅	{4}
	∅	∅	∅



6. Regular Languages & Finite Automata

- Regular Language Topics

Solution: Draw an NFA and then use the algorithm.

Then,

1. build the transition table
2. and write it in simplified form

	T	*a*	*b*
S F	{0,1,2,3,5}	{1,2,3,5}	{4}
F	{1,2,3,5}	{1,2,3,5}	{4}
	{4}	{3,5}	∅
F	{3,5}	∅	{4}
	∅	∅	∅

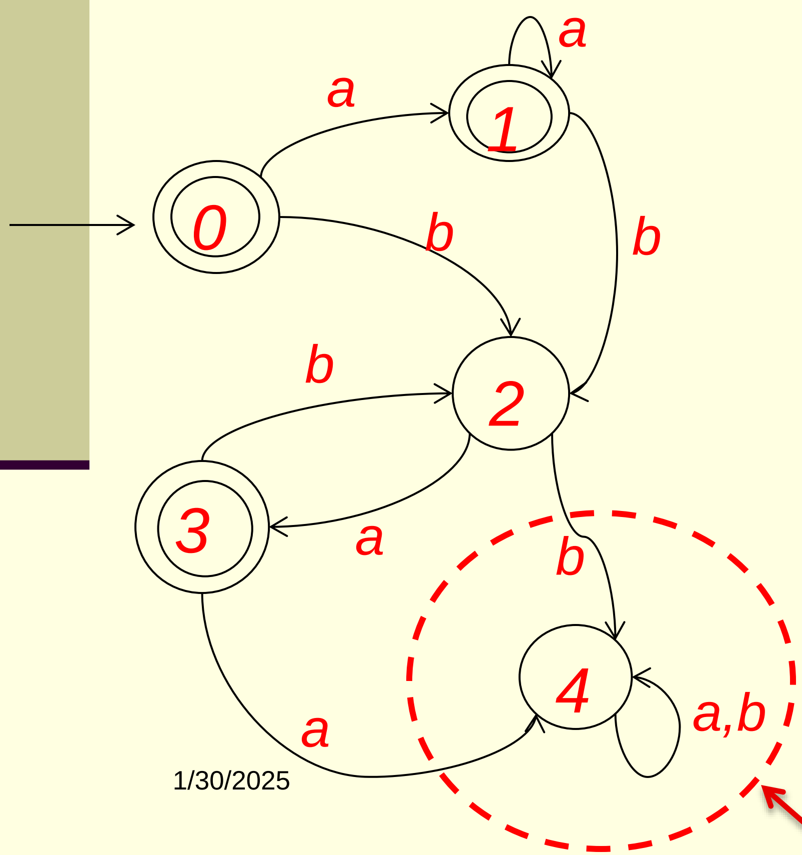
	T	*a*	*b*
S F	0	1	2
F	1	1	2
	2	3	4
F	3	4	2
	4	4	4

6. Regular Languages & Finite Automata

- Regular Language Topics

Solution: Draw an NFA and then use the algorithm.

Then, draw an DFA



	T	a	b
S F	0	1	2
F	1	1	2
	2	3	4
F	3	4	2
	4	4	4

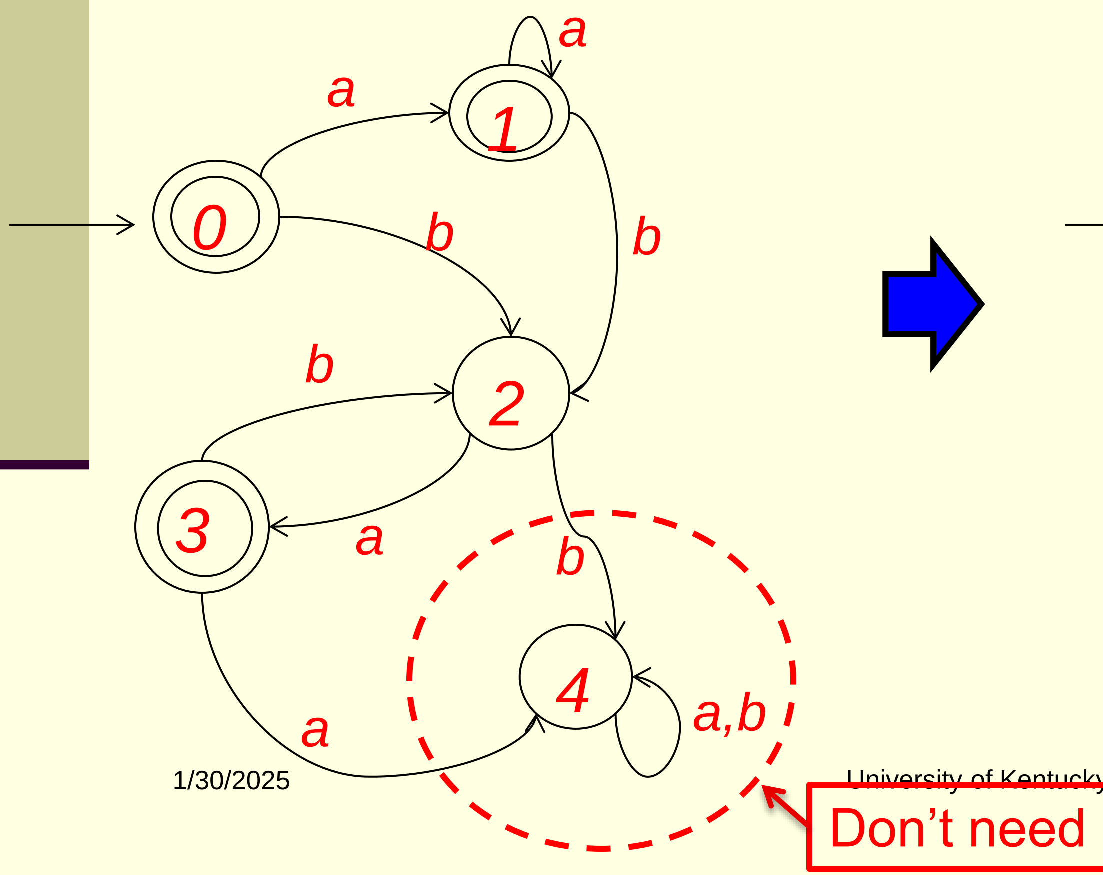
Don't need

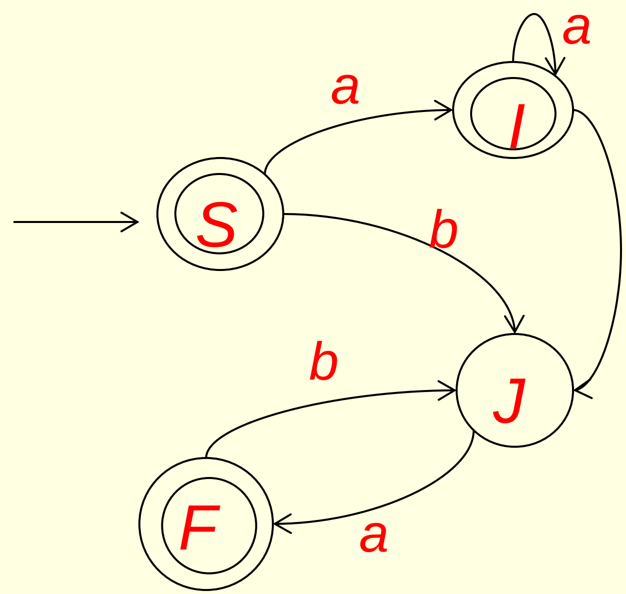
6. Regular Languages & Finite Automata

- Regular Language Topics

Solution: Draw an NFA and then use the algorithm.

Then, draw an DFA





6. Regular Languages & Finite Automata

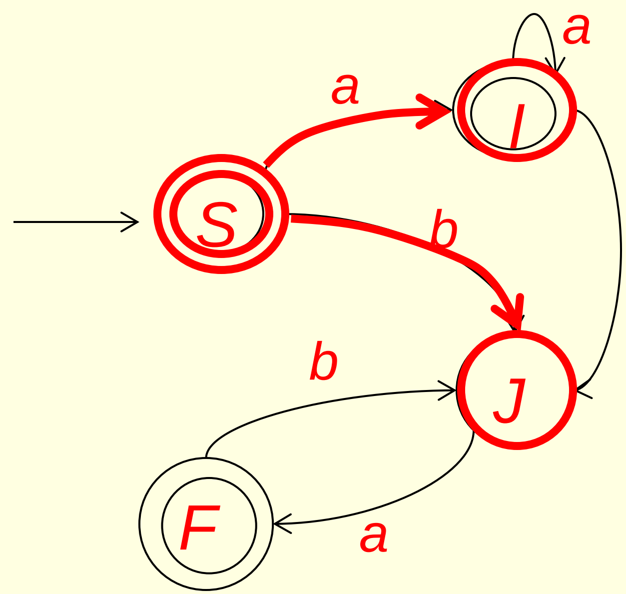
- Regular Language Topics

Solution: Draw an NFA and then use the algorithm.

Then, construct a regular grammar

Resulting grammar:

$$S \to aI \mid bJ \mid \Lambda$$
$$I \to aI \mid bJ \mid \Lambda$$
$$J \to aF$$
$$F \to bJ \mid \Lambda$$



simplify

$$S \to aI \mid baF \mid \Lambda$$
$$I \to aI \mid baF \mid \Lambda$$
$$F \to baF \mid \Lambda$$

can this be further simplified?

$$S \to aS \mid baF \mid \Lambda$$
$$F \to baF \mid \Lambda$$

6. Regular Languages & Finite Automata

- Regular Language Topics

How is simplification done?

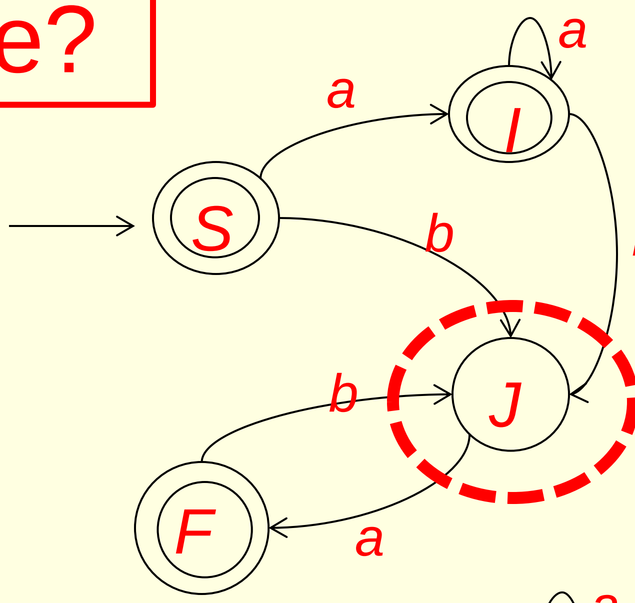
Resulting grammar:

$$S \to aI \mid bJ \mid \Lambda$$

$$I \to aI \mid bJ \mid \Lambda$$

$$J \to aF$$

$$F \to bJ \mid \Lambda$$

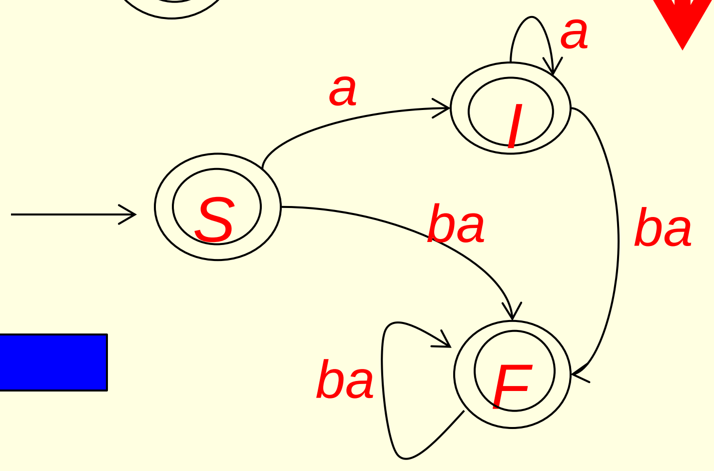


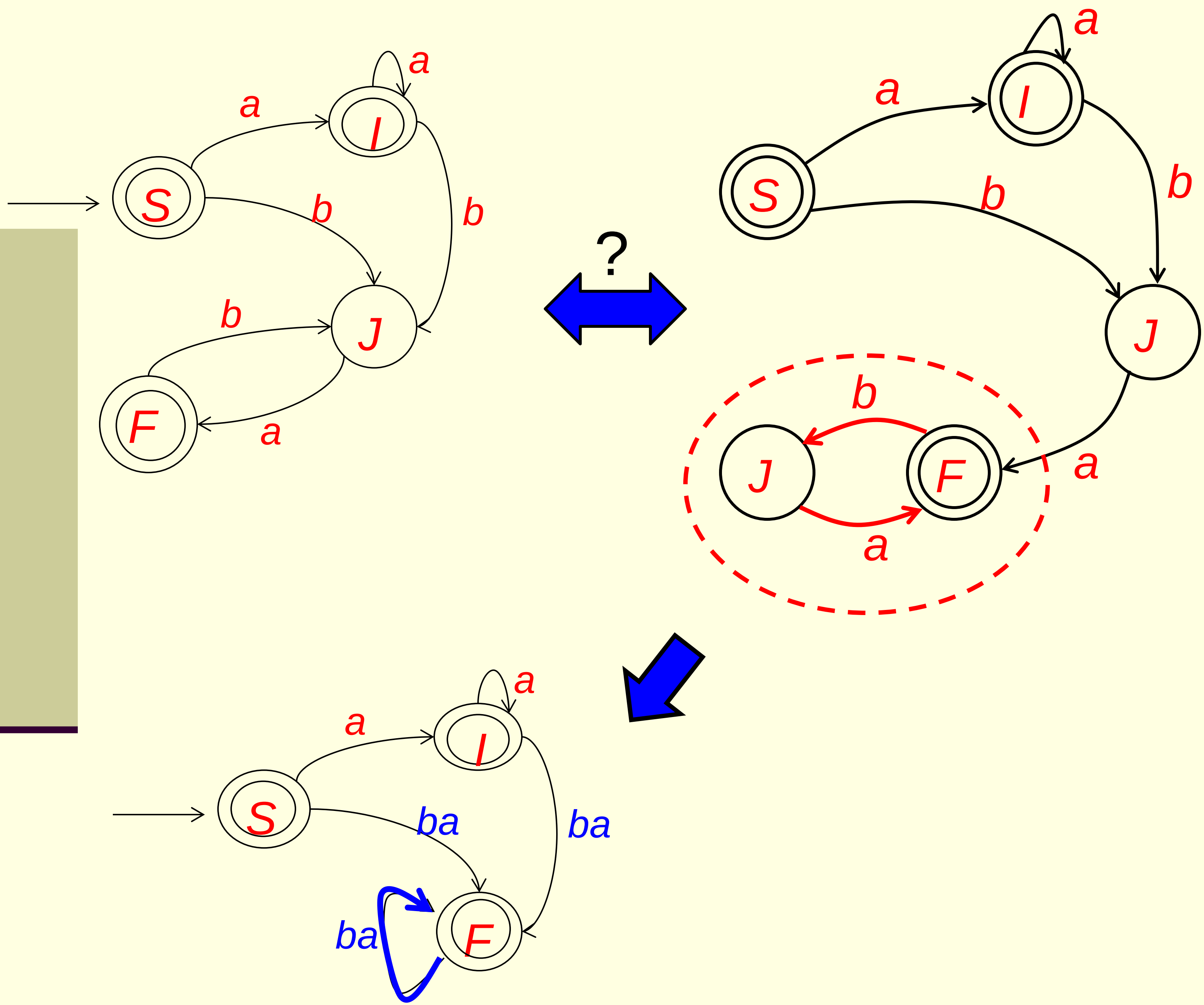
simplify

$$S \to aI \mid baF \mid \Lambda$$

$$I \to aI \mid baF \mid \Lambda$$

$$F \to baF \mid \Lambda$$





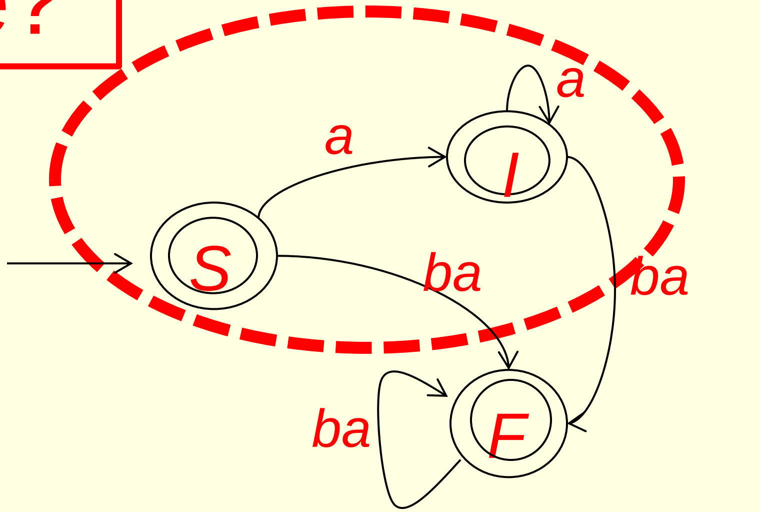
S
I
J
F
a
b
?
ba

6. Regular Languages & Finite Automata

- Regular Language Topics

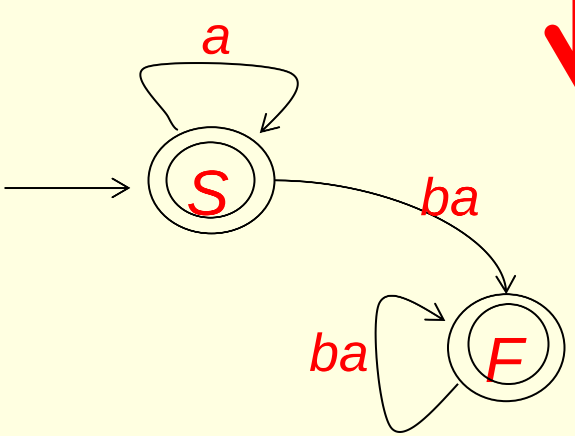
How is simplification done?

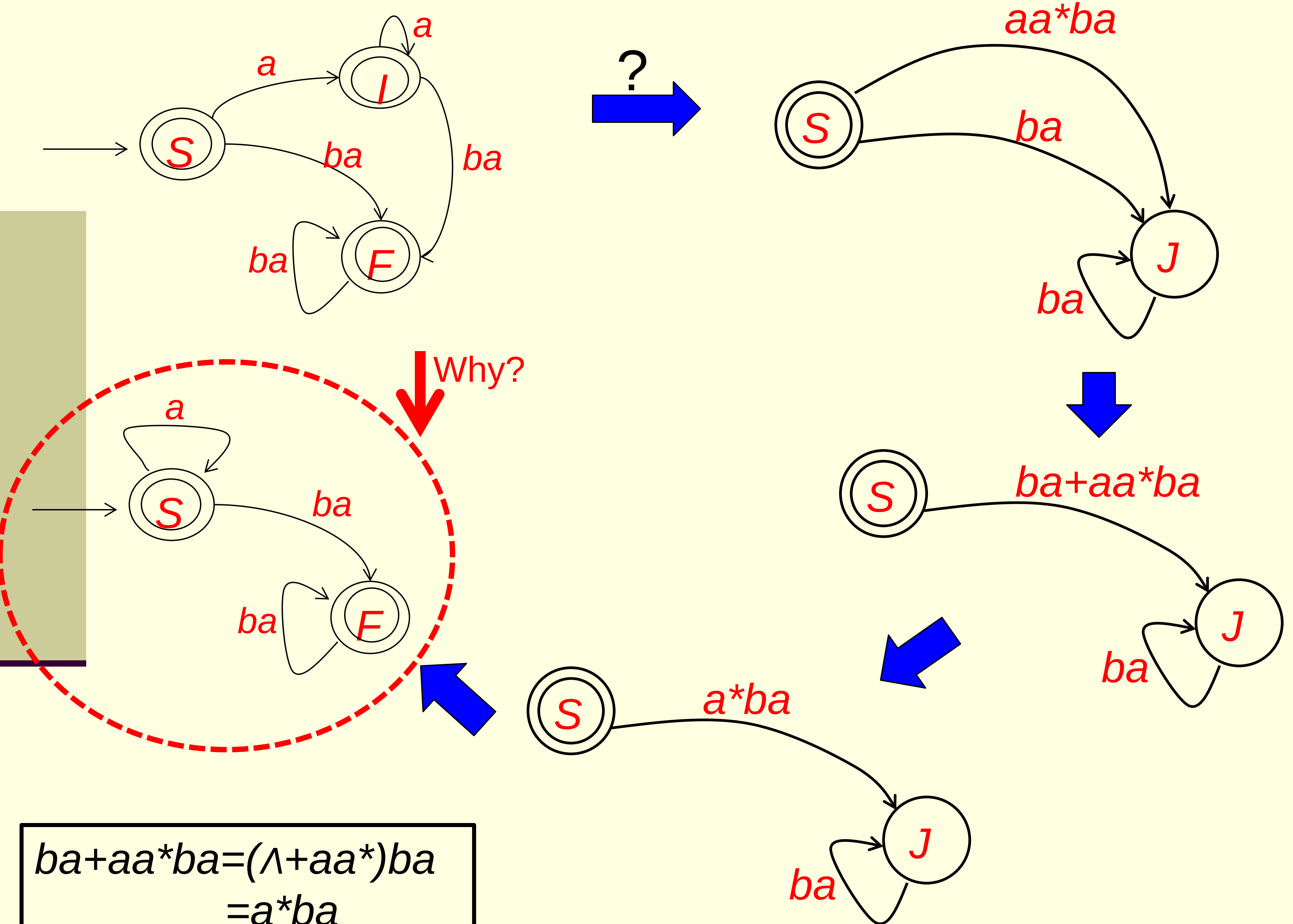
$$S \to aI \mid baF \mid \Lambda$$
$$I \to aI \mid baF \mid \Lambda$$
$$F \to baF \mid \Lambda$$



Further simplification

$$S \to aS \mid baF \mid \Lambda$$
$$F \to baF \mid \Lambda$$



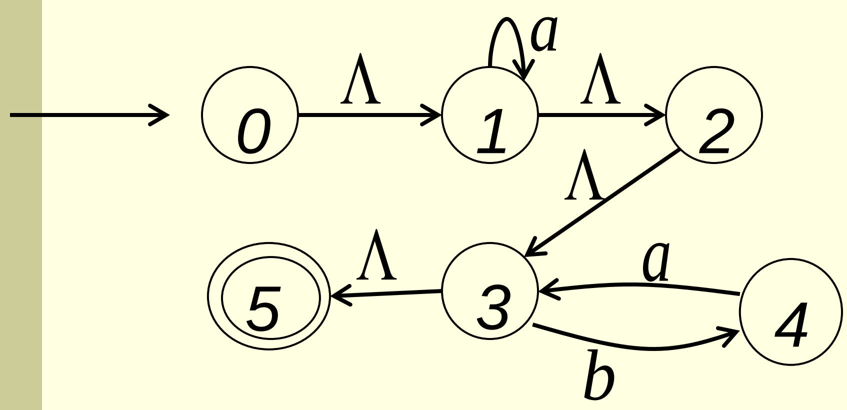


$ba+aa^*ba=(\Lambda+aa^*)ba$

$=a^*ba$

6. Regular Languages & Finite Automata

- Regular Language Topics



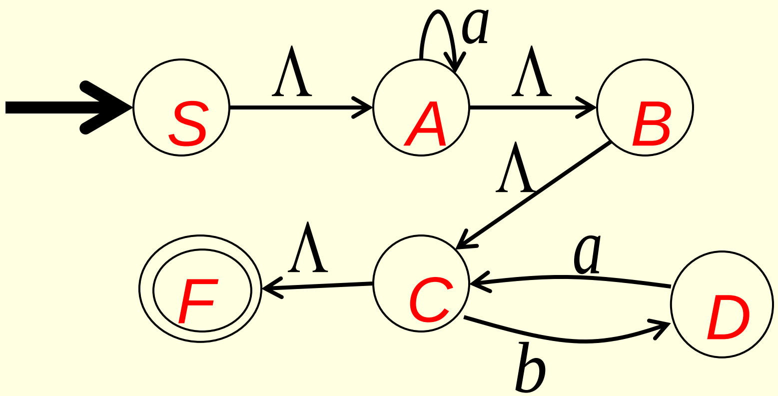
Question: *Why do we want to convert this NFA to a DFA and then convert it to a regular grammar?*

Question: *If we convert this NFA to a regular grammar, would we get the same regular grammar?*

6. Regular Languages & Finite Automata

- Regular Language Topics

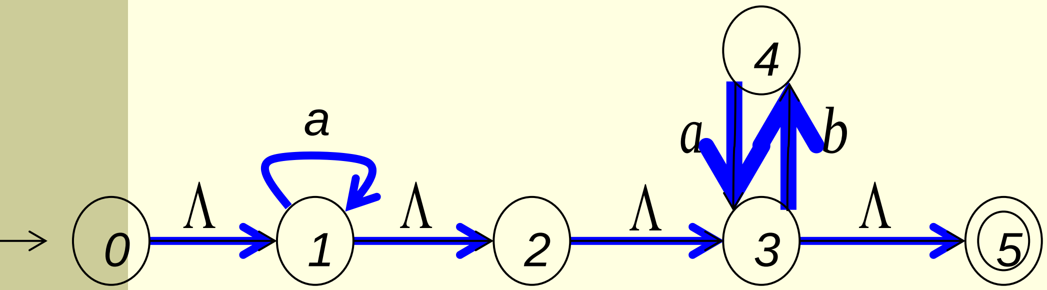
Question: *If we convert this NFA to a regular grammar, would we get the same regular grammar?*

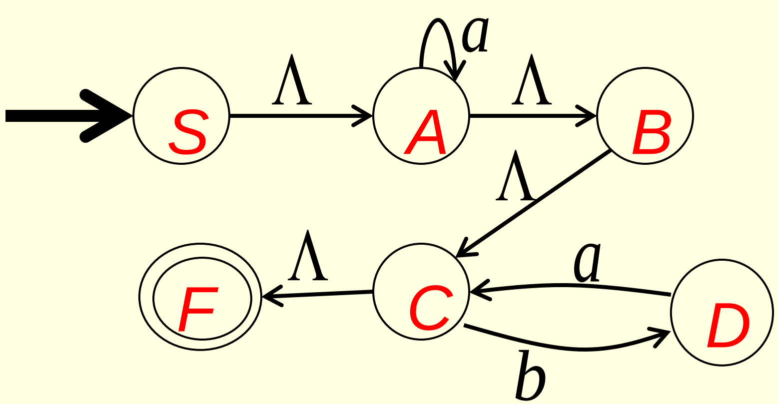


$S \to A$
$A \to aA \mid B$
$B \to C$
$C \to bD \mid F$
$D \to aC$
$F \to \Lambda$

?

$S \to aS \mid baF \mid \Lambda$
$F \to baF \mid \Lambda$





$S \to A$
$A \to aA \mid B$
$B \to C$
$C \to bD \mid F$
$D \to aC$
$F \to \Lambda$

?

$S \to aS \mid C$
$C \to baC \mid \Lambda$

$S \to aS \mid baC \mid \Lambda$
$C \to baC \mid \Lambda$

$S \to aS \mid baF \mid \Lambda$
$F \to baF \mid \Lambda$

6. Regular Languages & Finite Automata

- Regular Language Topics

NFA ← *Regular Grammar*

Regular Expression ← *NFA* ← *Regular Grammar*

Transforming a Regular Grammar to an NFA

1. **Replace any production with multiple terminals by productions with single terminals.**
2. The start state is the grammar start symbol.
3. Transform $I \rightarrow aJ$ into a transition from I to J labeled with a.
4. Transform $I \rightarrow J$ into a transition from I to J labeled with Λ.
5. Transform each $I \rightarrow a$ into a transition from I to new single final state F labeled with a.
6. The final states are F together with each state I with a production $I \rightarrow \Lambda$.

$$\boxed{S \rightarrow abJ \;\Rightarrow\; \begin{cases} S \rightarrow aI \\ I \rightarrow bJ \end{cases}}$$

6. Regular Languages & Finite Automata

- Regular Language Topics

5. Transform each $I \rightarrow a$ into a transition from I to new single final state F labeled with a.

' $I \rightarrow a$ ' means the production stops once ' a ' is produced, so ' a ' must be the label of an edge to a final state.

6. each state I with a production $I \rightarrow \Lambda$ *is a final state.*

' $I \rightarrow \Lambda$ ' means the production stops at I, so I must be a final state.

Example. Transform the following regular grammar into an NFA.

$$\boxed{\begin{aligned} &S \to abS \mid T \mid \Lambda \\ &T \to cT \mid d \end{aligned}}$$

Solution. Transform $S \to abS$ into

$S \to aI$ and $I \to bS$,

so the grammar becomes

$$S \to aI \mid T \mid \Lambda$$

$$I \to bS$$

$$T \to cT \mid d$$

They can be written as :

6. Regular Languages & Finite Automata

- Regular Language Topics

$S \to aI$

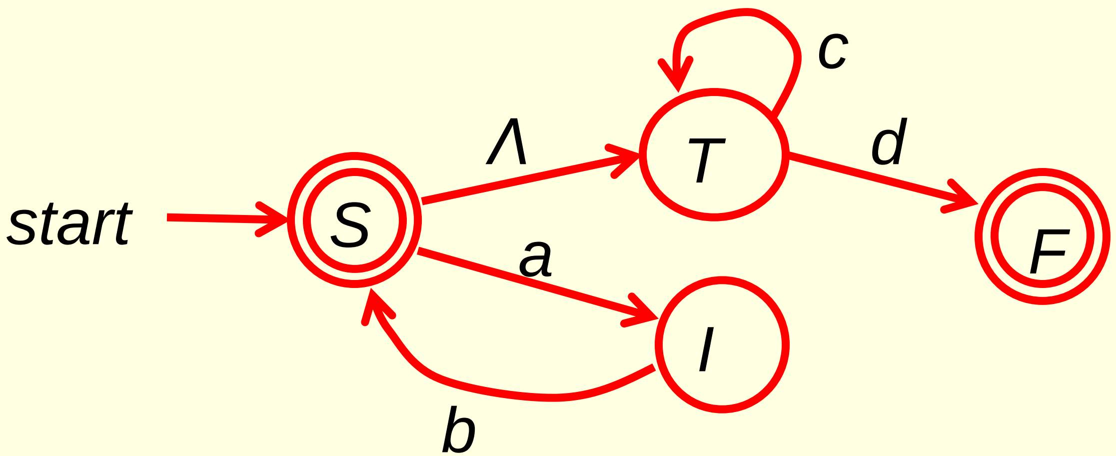
$S \to T$

$S \to \Lambda$

$I \to bS$

$T \to cT$

$T \to d$

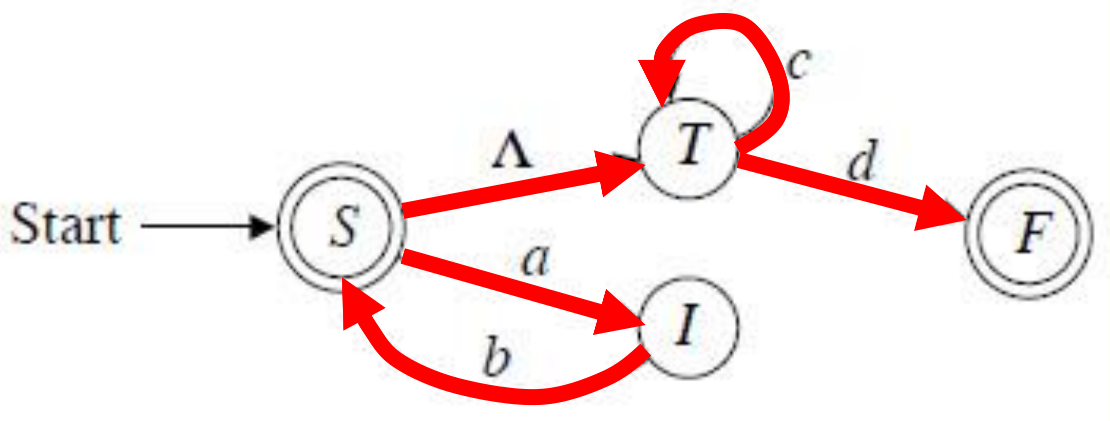


Now the NFA can be drawn:

6. Regular Languages & Finite Automata

- Regular Language Topics

Example. What is the regular expression for the language of the grammar?



Answer:

$$(ab)^* + (ab)^* \Lambda c^* d$$

$$= (ab)^* + (ab)^* c^* d$$

$$= (ab)^* (\Lambda + c^* d)$$

$\neq (ab)^*(c^*d)$

Regular … important prope…

that is, having a middle section of the word repeated an arbitrary number of times

All sufficiently long words in a regular language may be *pumped* to produce a new word that also lies within the same language.

If $w = w_1 w_2 w_3 \in L$

then $w_1 w_2 w_2 w_3 \in L$

$w_1 w_2 w_2 w_2 w_3 \in L$

$w_1 w_2 w_2 w_2 w_2 w_3 \in L$

$\vdots$

Why?
Because of the Pumping Lemma

6. Regular Languages & Finite Automata

- Regular Language Topics

The Pumping Lemma

If *L* is an $\boxed{\text{infinite}}$ regular language, then it is recognized by a DFA with, say, *m* states. If *s* in *L* and $|s| \geq m$, then an acceptance path for *s* must pass through some state twice.

An acceptance path

Why?

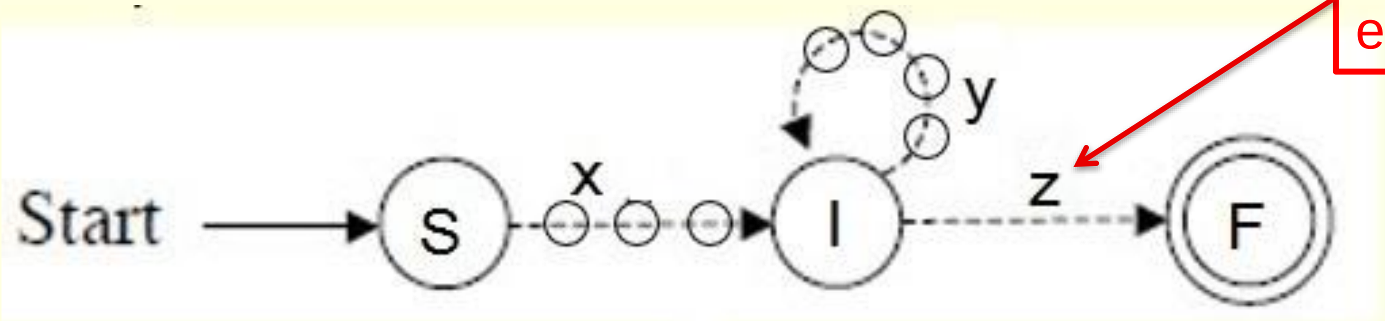
A string of length m is built by m edges and so by m+1 states.

6. Regular Languages & Finite Automata

- Regular Language Topics

The following graph depicts the situation.

x, z could be empty



Dotted arrows represent the path of acceptance for *s*

Letters *x, y,* and *z* represent the concatenation of the letters along the edges of the path. So $s = xyz$ and $y \neq \Lambda$.

Assume that the middle state is the first repeated state on the path. So $|\,xy\,| \leq m$. Since the loop can be traversed any number of times, we have the *Pumping property:* $xy^k z \in L$ for all $k \in \mathbf{N}$.

6. Regular Languages & Finite Automata

- Regular Language Topics

Example. The language $L = \{ a^n b^n \mid n \in \mathbf{N}\}$ is not regular.

Proof: Assume, BWOC, that L is regular.

Since L is infinite, the Pumping Lemma applies.

Choose s = $a^m b^m$.
Then s = xyz, where y ≠ Λ, | xy | ≤ $2m$, and $xy^k z \in L$ for all k ϵ ***N.***

We claim that y consists completely of a's or b's only.
For if y is of the form y = $a^i b^j$, $i > 0, j > 0,$
then we have $x = a^{m-i}$ and z = b^{m-j}.
So, $xy^2z = a^{m-i}(a^i b^j)(a^i b^j)b^{m-j} = a^m b^j a^i b^m \in L$.
But this is a contradiction. So we have either
$y = a^i$, $i > 0$, or $y = b^j$, $j > 0$.

m $\qquad\qquad$ m

s = *aaaaaaaaaaa……...aaaaaaaaa|bbbbbbbb……….bbbbbbbbbbb*

m-i $\qquad\qquad$ m-j

i $\qquad$ j

$x = a^{m-i}$ $\qquad$ $y = a^i b^j$ $\qquad$ $z = b^{m-j}$

Pumping Lemma says:

$$xy^2z = a^{m-i}(a^i b^j)(a^i b^j)b^{m-j} = a^m b^j a^i b^m \in L$$

But this is a contradiction!

6. Regular Languages & Finite Automata

- Regular Language Topics

Example. The language $L = \{ a^n b^n \mid n \in \mathbf{N} \}$ is not regular.

Proof (conti.):

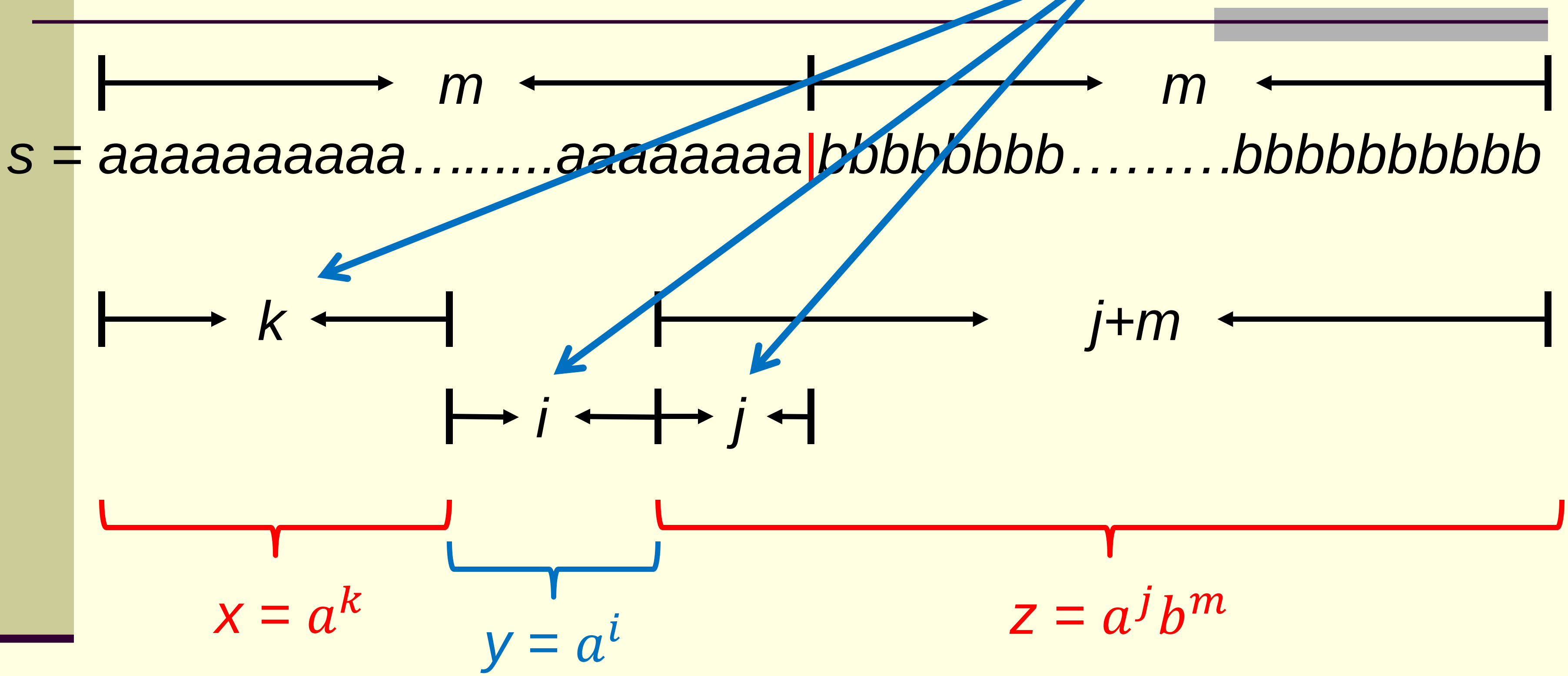
If $y = a^i$ *for some i > 0 then xy is a string of a's.*
Since $s = xyz = a^m b^m$ *and* xy^2 *is also a string of a's,*
we have $xy^2z = a^{m+i}b^m$. By Pumping Lemma, this is supposed to be an element of *L*, but this is impossible because *a* and *b* have different exponents.

Similarly we can also prove that if $y = b^j$ for some *j > 0*, we would also get into a contradiction.

Hence, *L* can not be regular. QED

$\boxed{y = a^i, \quad i > 0}$

$\boxed{k + i + j = m}$



Pumping Lemma says:

$xy^2z = \boxed{\underline{a^k}\,\underline{(a^i)}\,\underline{(a^i)}\,\underline{a^j}}\,b^m = \boxed{a^{k+2i+j}}\,b^m = \boxed{a^{m+i}}\,b^m \in L$

But this is a contradiction!

6. Regular Languages & Finite Automata

- Regular Language Topics

Example. In the previous proof we exhibited a contradiction when $k = 2$. Find similar contradictions for $k = 0$ and $k = 3$.

Answer: ($k = 0$) The pumping property implies $xy^0z \in L$. In other words, $xz \in L$. But $xz = a^{m-i}b^m$, which is not in L because $i > 0$. This contradiction implies that L is not regular. QED.

($k = 3$) The pumping property implies $xy^3z \in L$. But $xy^3z = a^{m+2i}b^m$, which is not in L because $i > 0$. This contradiction implies that L is not regular. QED.

Some time it is possible to transform a regular expression to a regular grammar directly.

Consider: *ab* + c*

Can not use
$S \rightarrow aS$ here

{a, ab, abb, abbb, ... }

{ c }

So we must have:

$S \rightarrow aA$
$A \rightarrow bA \mid \Lambda$

$S \rightarrow c$

$S \rightarrow aA \mid c$
$A \rightarrow bA \mid \Lambda$

Some time it is possible to transform a regular expression to a regular grammar directly.

Consider: $a^*b^* + c$

$\{ a^i b^j \mid i, j \geq 0 \}$

S → aA | bB | Λ
A → aA | bB | Λ
B → bB | Λ

$\{ c \}$

S → c

S → aA | bB | Λ | c
A → aA | bB | Λ
B → bB | Λ

Some time it is possible to transform a regular expression to a regular grammar directly.

Consider: *a*b* + c*

$\{ a^i b^j \mid i, j \geq 0 \}$

$\{ c \}$

or

S → A | B | Λ
A → aA | bB | Λ
B → bB | Λ

S → c

S → A | B | Λ | c
A → aA | bB | Λ
B → bB | Λ

Some time it is possible to transform a regular expression to a regular grammar directly.

Consider: *a*b* + c*

$\{ a^i b^j \mid i, j \geq 0 \}$

$S \rightarrow aS \mid bS \mid \Lambda$

$\{ c \}$

$S \rightarrow c$

Would this set of productions work?

NO

Some time it is possible to transform a regular expression to a regular grammar directly.

Consider: $a^*b^*c + ab$

$\{ b^*c,\ a^+b^*c \}$

$S \rightarrow A$
$A \rightarrow aA \mid B$
$B \rightarrow bB \mid c$

$\{ ab \}$

$S \rightarrow ab$

$S \rightarrow A \mid ab$
$A \rightarrow aA \mid B$
$B \rightarrow bB \mid c$

Some time it is possible to transform a regular expression to a regular grammar directly.

Consider: *$a^*b^*c + ab$*

$\{ b^*c, a^+b^*c \}$

$\{ ab \}$

$S \rightarrow aA \mid B$
$A \rightarrow aA \mid B$
$B \rightarrow bB \mid c$

$S \rightarrow ab$

Would this set of productions work?

YES

Some time it is possible to transform a regular expression to a regular grammar directly.

Consider: *$a^*b^*c + ab$*

$\{ b^*c, a^+b^*c \}$

S → aA | bB | c
A → aA | bB | c
B → bB | c

$\{ ab \}$

S → ab

Or, would this set of productions work?

YES

End of Regular Language and Finite Automata IV