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)**

7. Context-Free Languages & Pushdown Automata- Context-Free Languages

Goal:

To study a non-regular language called **context-free language**,

and the machine called **pushdown automaton** that recognizes this language.

Why do we want to study context-free language?

7. Context-Free Languages & Pushdown Automata- Context-Free Languages

Why do we want to study context-free language?

Because regular languages do not include any programming languages such as Pascal, C, C++, Java or Python as members.

Why? Consider the data item a^2b^2. This item can be a term in a polynomial and yet it is not recognized as a regular language term.

7. Context-Free Languages & Pushdown Automata- Context-Free Languages

Can a machine recognize a language that is not regular?

YES.

The language $\{a^n b^n \mid n \in N\}$ is not regular, but it has a non-regular grammar

S → aSb | Λ

Can have more than one production

A **context-free grammar** has productions of the form

$N \rightarrow w$

where *N* is a non-terminal and *w* is a string containing terminals and/or non-terminals.

7. Context-Free Languages & Pushdown Automata- Context-Free Languages

Why is the grammar called

"context-free grammar" ?

Consider the rule

$S \rightarrow aSb$

What this says is "*wherever you find S, you can replace it with aSb*".

7. Context-Free Languages & Pushdown Automata- Context-Free Languages

Why is the grammar called *"context-free grammar" ?*

$S \rightarrow aSb$

$CSB \rightarrow CaSbB$

==========

In summary, in the first case, you didn't need any context to apply the rule. You can apply it irrespective of the context in which S appears. So, grammars which contain only rules of first kind are called context-free grammars.

7. Context-Free Languages & Pushdown Automata- Context-Free Languages

Why is the grammar called

"context-free grammar" ?

Now consider the rule

$CSB \rightarrow CaSbB$

This says "*You can replace S with aSb only if it is preceded by C and followed by B*". Here it imposes a condition on when S can be replaced with aSb. You can apply this rule only if S appears in this particular **context**. Here, 'context' is used as is generally used in normal English.

7. Context-Free Automata- Context-Free

Think of the strings in the following form:

bb, (ab)bb, (aabb)bb, (aaabbb)bb, …

A ***context-free language*** is strings derived from a context-free grammar.

Example. $\{a^n b^n \mid n \in \mathbf{N}\}$ is a anguage derived from the C-F grammar

$S \rightarrow aSb$

Example. Any regular gramma s context-free.

So regular languages are C-F languages.

Quiz. *Find a grammar for* $\{a^n b^{n+2} \mid n \in \mathbf{N}\}$.

Answer. $S \rightarrow aSb \mid bb.$

7. Context-Free Languages & Pushdown Automata- Context-Free Languages

Quiz. Find a grammar for $\{ w w^R \mid w \in \{a, b\}^* \}$ where w^R is the reverse of *w.*

Answer.

Example. $S \Rightarrow aSa \Rightarrow abSba \Rightarrow ab\Lambda ba \Rightarrow abba$

Question. What if $w \in \{a, b, c, d\}^*$?

Would $S \rightarrow aSa \mid bSb \mid cSc \mid dSd \mid \Lambda$ *work?*

7. Context-Free Languages & Pushdown Automata- Context-Free Languages

Techniques for Constucting Grammars:

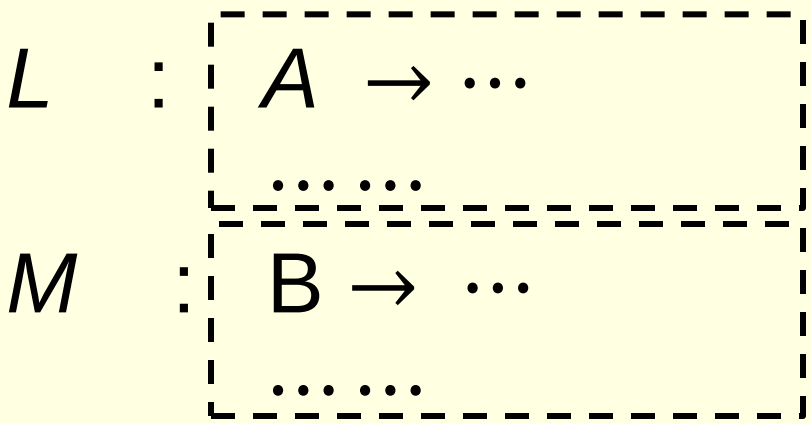
Let L and M be two C-F grammars with **disjoint** sets of **non-terminals** and with start symbols A and B, respectively.

Then

- $L \cup M$ has grammar $S \rightarrow A \mid B$
- LM has grammar $S \rightarrow AB$
- L^* has grammar $S \rightarrow AS \mid \Lambda$

How about the alphabets?

Why do we need a 'Λ' here?



Then

- $L \cup M$: $S \rightarrow A \mid B$ *plus* (1)
- LM : $S \rightarrow AB$ *plus* (2)
- L^* : $S \rightarrow AS \mid \Lambda$ *plus* (3)

L^*M has grammar $S \rightarrow AS \mid B$ (4)

$S \Rightarrow AA\ldots AAS$

$\Rightarrow AA\ldots AAB$

7. Context-Free Languages & Pushdown Automata- Context-Free Languages

Examples. For a string x and letter a let $n_a(x)$ be the number of a's in x. Let $L = \{ x \in \{a, b\}^* \mid n_a(x) = n_b(x) \}$.

A C-F grammar for L with start symbol E can be written as:

$E \rightarrow aEbE \mid bEaE \mid \Lambda.$

Use this information to find grammars for following languages.

1. $\{ x \in \{a, b\}^* \mid n_a(x) = 1 + n_b(x) \}$.

Solution: $S \rightarrow EaE.$

Would $S \rightarrow aE$ or $S \rightarrow Ea$ work?

Can this grammar generate a^4ab^4, $a^2b^3ab^5a^6$, $b^2aa^5b^3$?

Examples. For a string x and letter a let $n_a(x)$ be the number of a's in x. Let $L = \{ x \in \{a, b\}^* \mid n_a(x) = n_b(x) \}$.

A C-F grammar for L with start symbol E can be written as:

$$E \rightarrow aEbE \mid bEaE \mid \Lambda.$$

Question 1:

Would $E \rightarrow aEb \mid bEa \mid \Lambda$ *work for L ?*

NO.

Because we wouldn't be able to generate strings such as: $abba$, $baab$, ab

Can not generate strings whose 2nd half is symmetric to its 1st half

$E \rightarrow aEbE \rightarrow aEbbEaE \rightarrow a\Lambda bb\Lambda a\Lambda = abba$

Examples. For a string x and letter a let $n_a(x)$ be the number of a's in x. Let $L = \{ x \in \{a, b\}^* \mid n_a(x) = n_b(x) \}$.

A C-F grammar for L with start symbol E can be written as:

$$E \rightarrow aEbE \mid bEaE \mid \Lambda.$$

Question 2:

Why don't we need an 'E' on the left hand side of the productions, i.e., why don't we need productions of the following form?

$$E \rightarrow EaEbE \mid EbEaE \mid \Lambda$$

Because the left most symbol of each string in L is either 'a' or 'b'. If it is 'a', we can start with the first production. If it is 'b', we can start with the second production.

7. Context-Free Languages & Pushdown Automata- Context-Free Languages

1. $\{ x \in \{a, b\}^* \mid n_a(x) = 1 + n_b(x) \}$.

Solution: $S \rightarrow EaE.$

Woul $S \rightarrow aE$ or $S \rightarrow Ea$ work ?

Can this grammar generate a^4ab^4, $a^2b^3ab^5a^6$, $b^2aa^5b^3$?

$a^4ab^4 = a\,(a^4\,b^4)$

?

?

$a^2b^3ab^5a^6 = (a^2b^3a)(b^5a^5)a$

$b^2aa^5b^3 = (b^2a^2)a(a^3b^3)$

7. Context-Free Languages & Pushdown Automata- Context-Free Languages

Examples. For a string x and letter a let $n_a(x)$ be the number of a's in x. Let $L = \{x \in \{a, b\}^* \mid n_a(x) = n_b(x)\}$.

A grammar for L with start symbol E can be written as:

$$E \rightarrow aEbE \mid bEaE \mid \Lambda.$$

Use this information to find grammars for following languages.

2. $\{x \in \{a, b\}^* \mid n_a(x) = 2 + n_b(x)\}$.

Solution: $S \rightarrow EaEaE.$

Why not $S \rightarrow EaaE$?

Can this grammar generate $a^4ab^2ab^2$, $a^2b^3ab^5aa^6$?

$a^4ab^2ab^2 = aa(a^2ab^2ab^2)$

$a^2b^3ab^5aa^6 = (a^2b^3ab^5aa^4)aa$

7. Context-Free Languages & Pushdown Automata- Context-Free Languages

Examples. For a string x and letter a let $n_a(x)$ be the number of a's in x. Let $L = \{x \in \{a, b\}^* \mid n_a(x) = n_b(x)\}$.

A grammar for L with start symbol E can be written as:

$E \rightarrow aEbE \mid bEaE \mid \Lambda.$

Use this information to find grammars for following languages.

3. $\{ x \in \{a, b\}^* \mid n_a(x) > n_b(x) \}$.

Solution: $S \rightarrow EaET$

$T \rightarrow aET \mid \Lambda.$

Would the following T work?
$T \rightarrow TEa \mid \Lambda$

To ensure the number of extra a's can be as many as possible

What is T for?

$S \rightarrow EaET \rightarrow EaEaET \rightarrow EaEaEaET \rightarrow \ldots.$

7. Context-Free Languages & Pushdown Automata- Context-Free Languages

Examples. For a string x and letter a let $n_a(x)$ be the number of a's in x. Let $L = \{x \in \{a, b\}^* \mid n_a(x) = n_b(x)\}$.

A grammar for L with start symbol E can be written as:

$$E \rightarrow aEbE \mid bEaE \mid \Lambda.$$

Use this information to find grammars for following languages.

3. $\{ x \in \{a, b\}^* \mid n_a(x) > n_b(x) \}$.

Solution: $S \rightarrow EaET$

$T \rightarrow aET \mid \Lambda.$

Questions: would $S \rightarrow EaES \mid \Lambda$ *or* $S \rightarrow EaES \mid a$ *work?*

None would work!!! Why?

7. Context-Free Languages & Pushdown Automata- Context-Free Languages

Examples. For a string x and letter a let $n_a(x)$ be the number of a's in x. Let $L = \{x \in \{a, b\}^* \mid n_a(x) = n_b(x)\}$.

A grammar for L with start symbol E can be written as:

$$E \rightarrow aEbE \mid bEaE \mid \Lambda.$$

Use this information to find grammars for following languages.

4. $\{x \in \{a, b\}^* \mid n_a(x) < n_b(x)\}$.

Solution: $S \rightarrow EbET$ $T \rightarrow bET \mid \Lambda.$

compare #4 with #3

7. Context-Free Languages and Pushdown Automata- Pushdown Automata

Infinite memory but restricted access

A pushdown automaton (PDA):

a finite automaton with a stack

stack operations: pop, push, and nop.

Do nothing

a PDA is an automaton that can memorize things

UNDO

What can we do with a PDA ?
can be used to model a **parser** for instance.

(Most programming languages have **deterministic PDA's**)

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

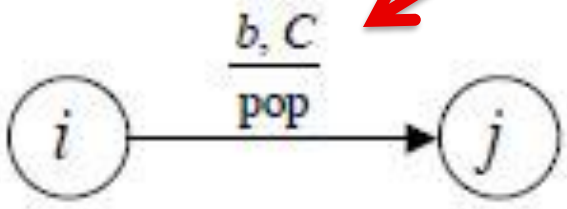
A PDA starts with one designated symbol on the stack.

A state transition is determined by:

current state
input symbol
top of the stack

Representation:

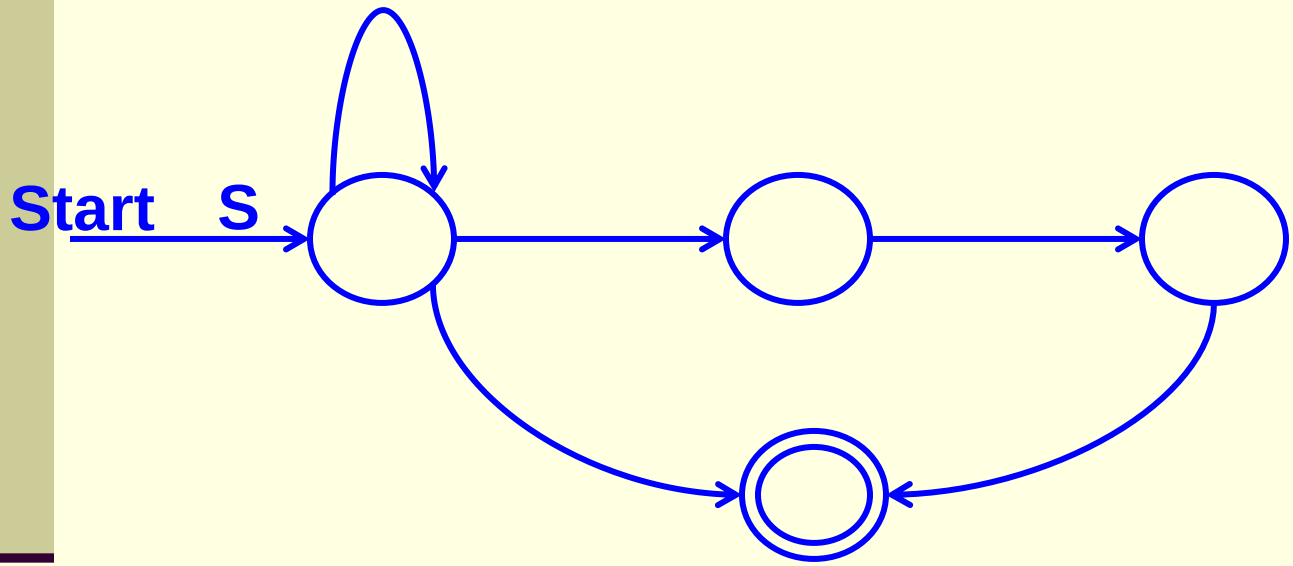
If the machine is in state *i* and the input letter is *b* and *C* is on top of the stack, then pop the stack and enter state *j*

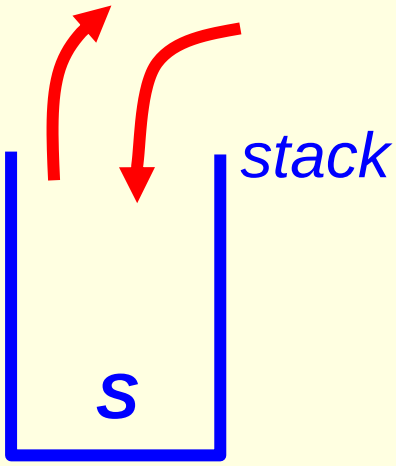


or

(i, b, C, pop, j)

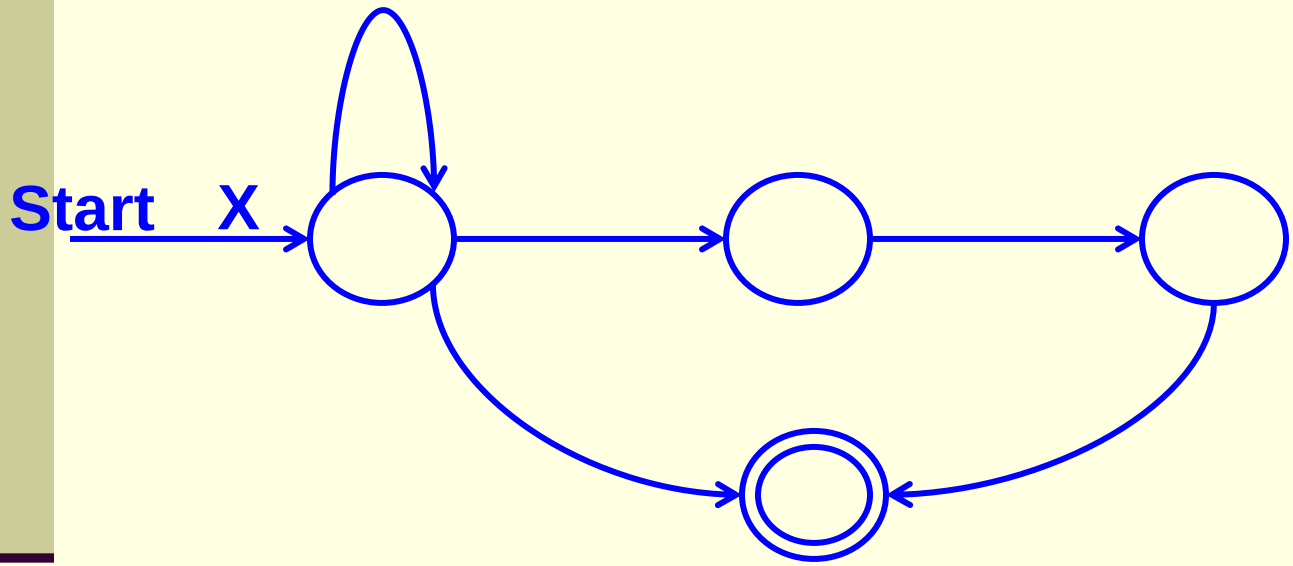
Pushdown Automata, intuitively

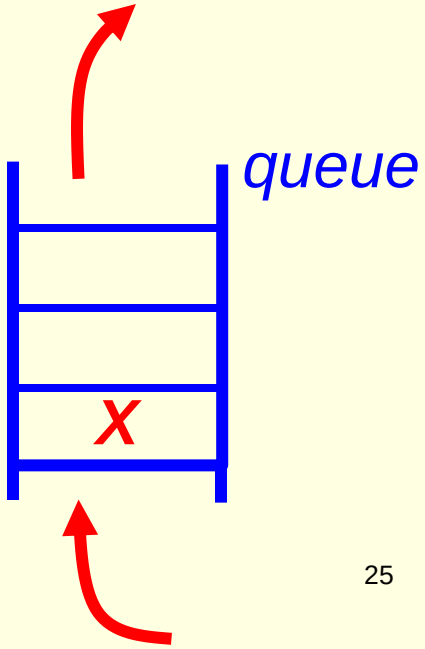




A stack is used when **events occurred more recently** are more important

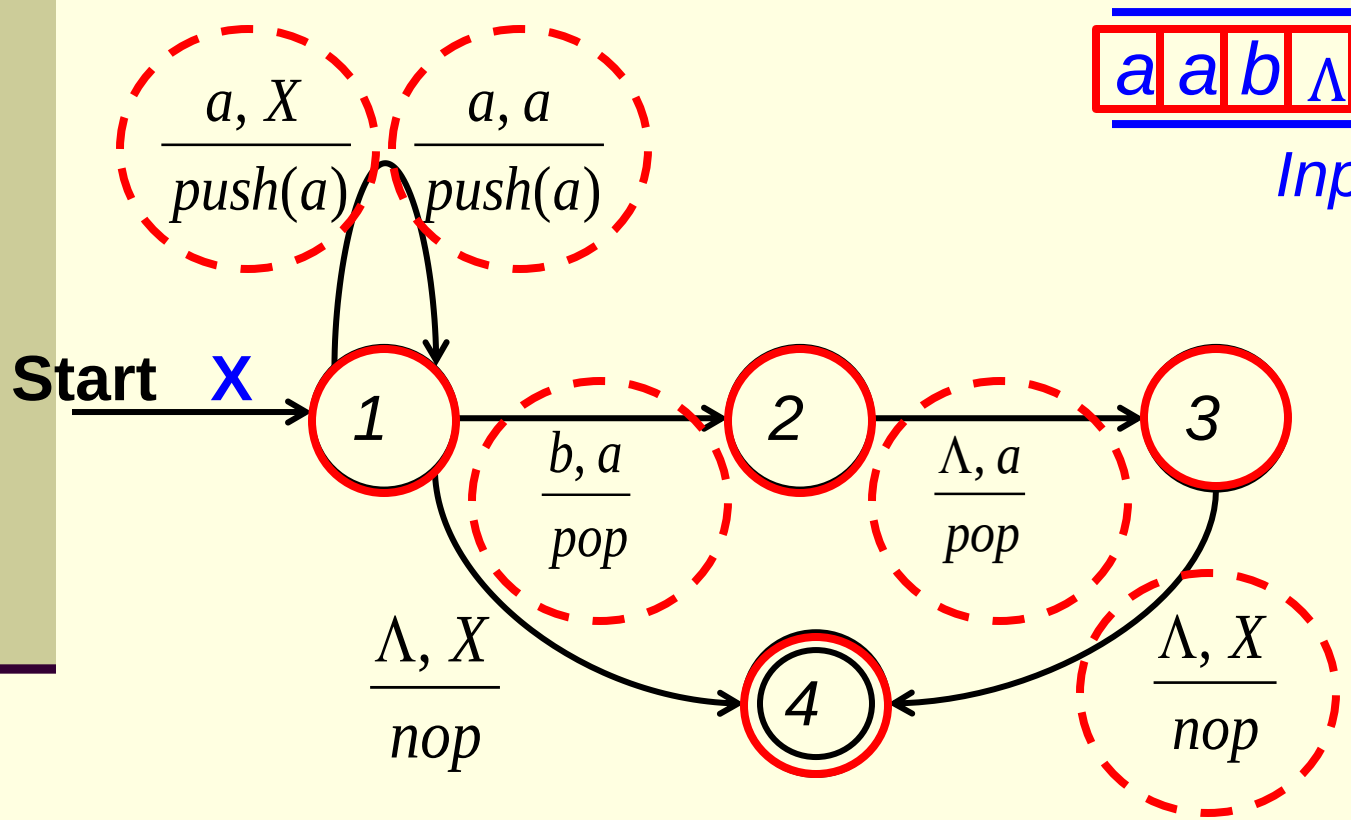
Why not a Pushup Automaton?



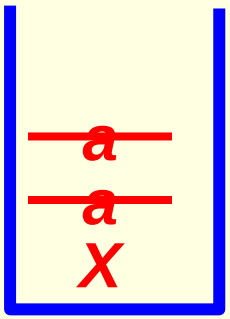


A queue is used when the **order of occurrence** is important

For the PDA example in slide 26:



Input string



stack

b a a

Output queue

Why is stack data structure important?

(1) can reverse a word;

(2) for "undo" mechanism;

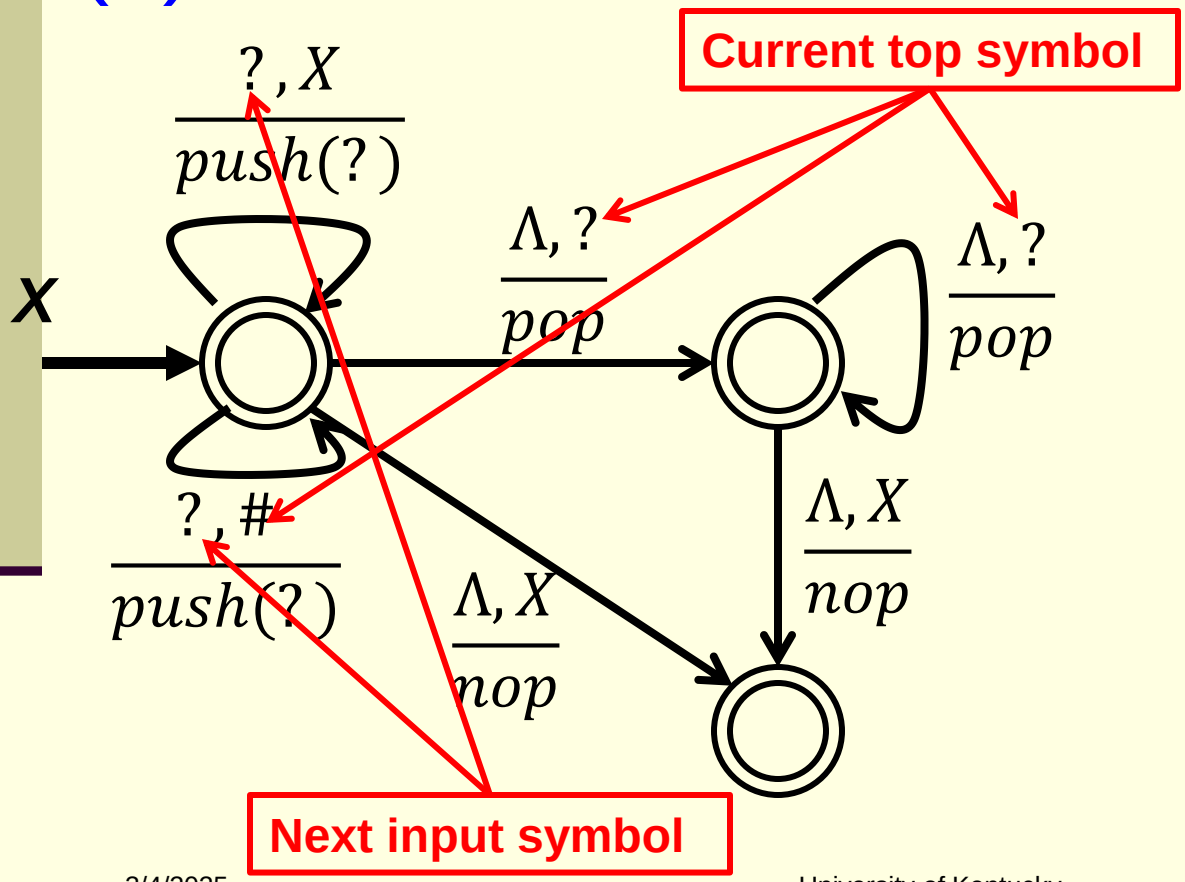
(3) can be used to match braces, …

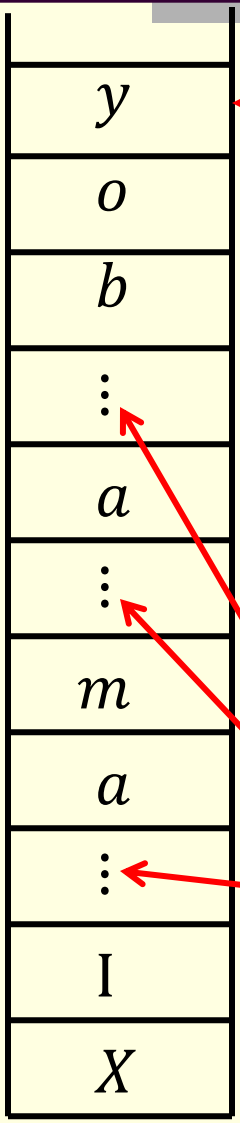
Very useful for language processing

Question: Can a queue do the same thing?

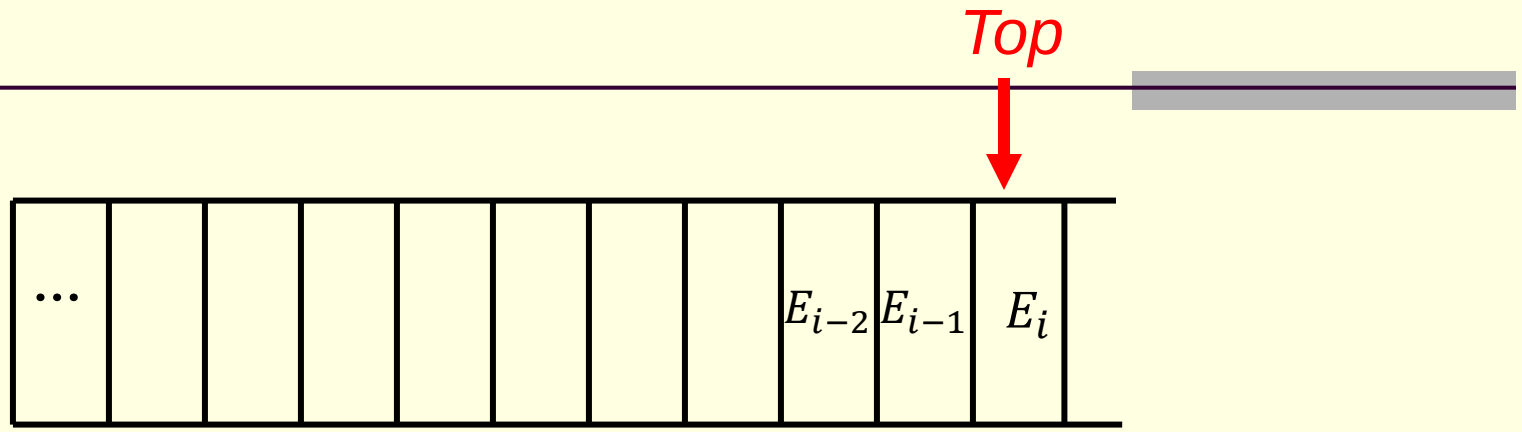
Why is stack data structure important?

(1) can reverse a word;

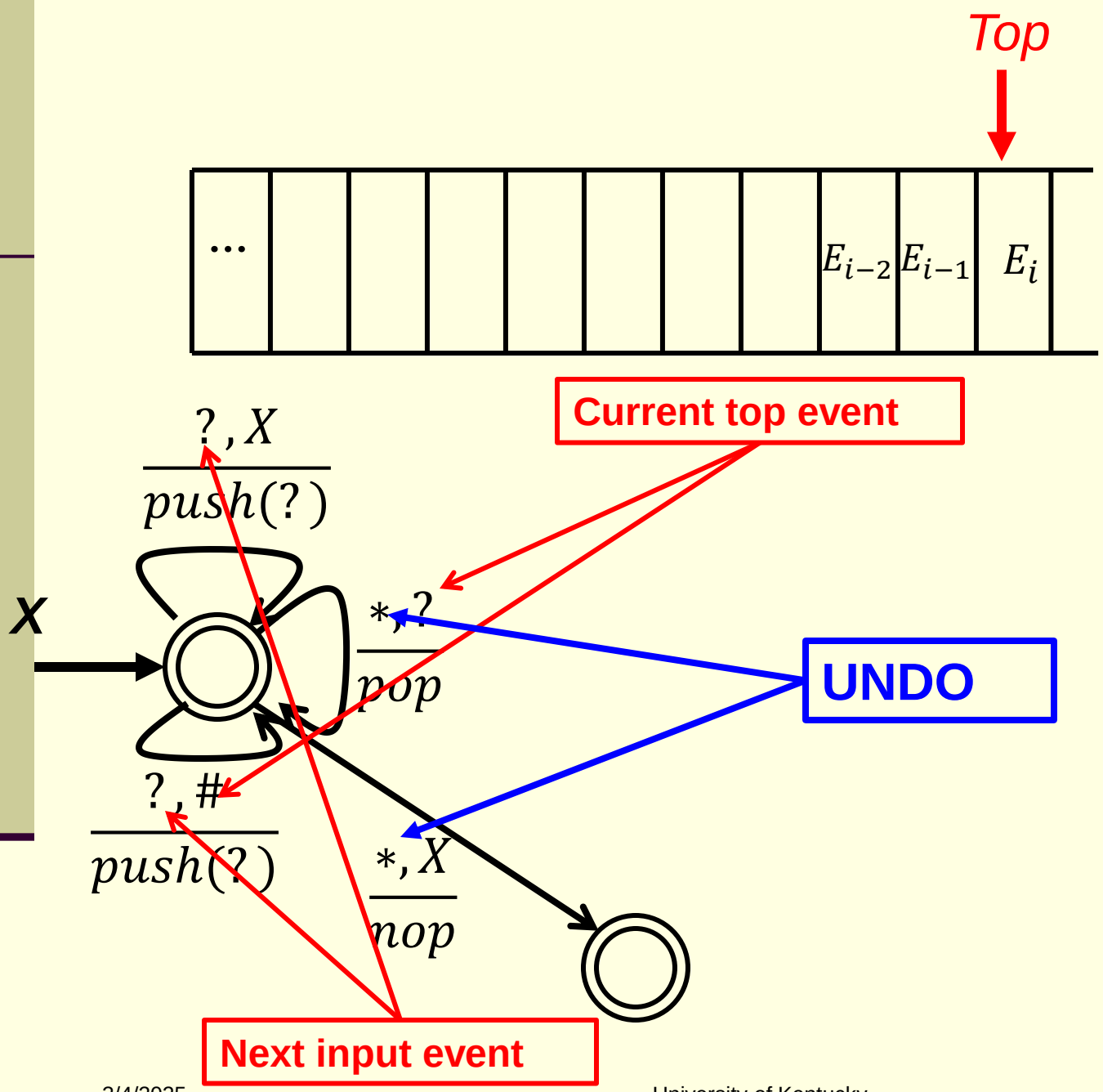




(2) for “undo” mechanism;



How should the previous PDA be modified?

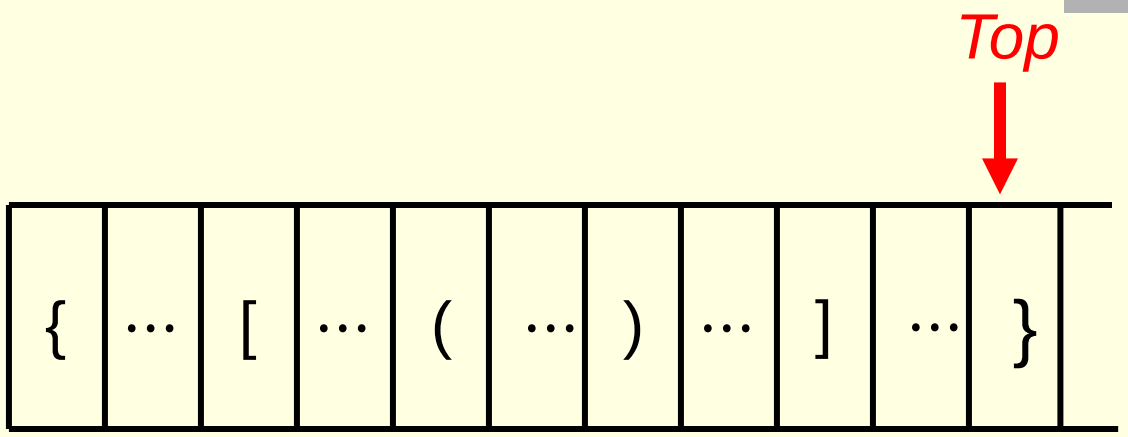

Top
E_{i-2}
E_{i-1}
E_i
Current top event
?, X
push(?)
X
*, ?
pop
UNDO
?, #
push(?)
*, X
nop
Next input event

UNDO and REDO can be implemented with two stacks

- Push all operations *to* **Undo-stack**
- When **UNDO** is called, pop operations from **Undo-stack** and push it to **Redo-stack.**
- When **REDO** is called, pop operations from **Redo-stack** and push it to **Undo-stack**.

How should the previous PDA be modified?

(3) can be used to match braces



{ [( )] }

How should the previous PDA be modified?

(use two stacks, one for every input symbol, one for braces only)

Final State acceptance

vs Empty-stack acceptance

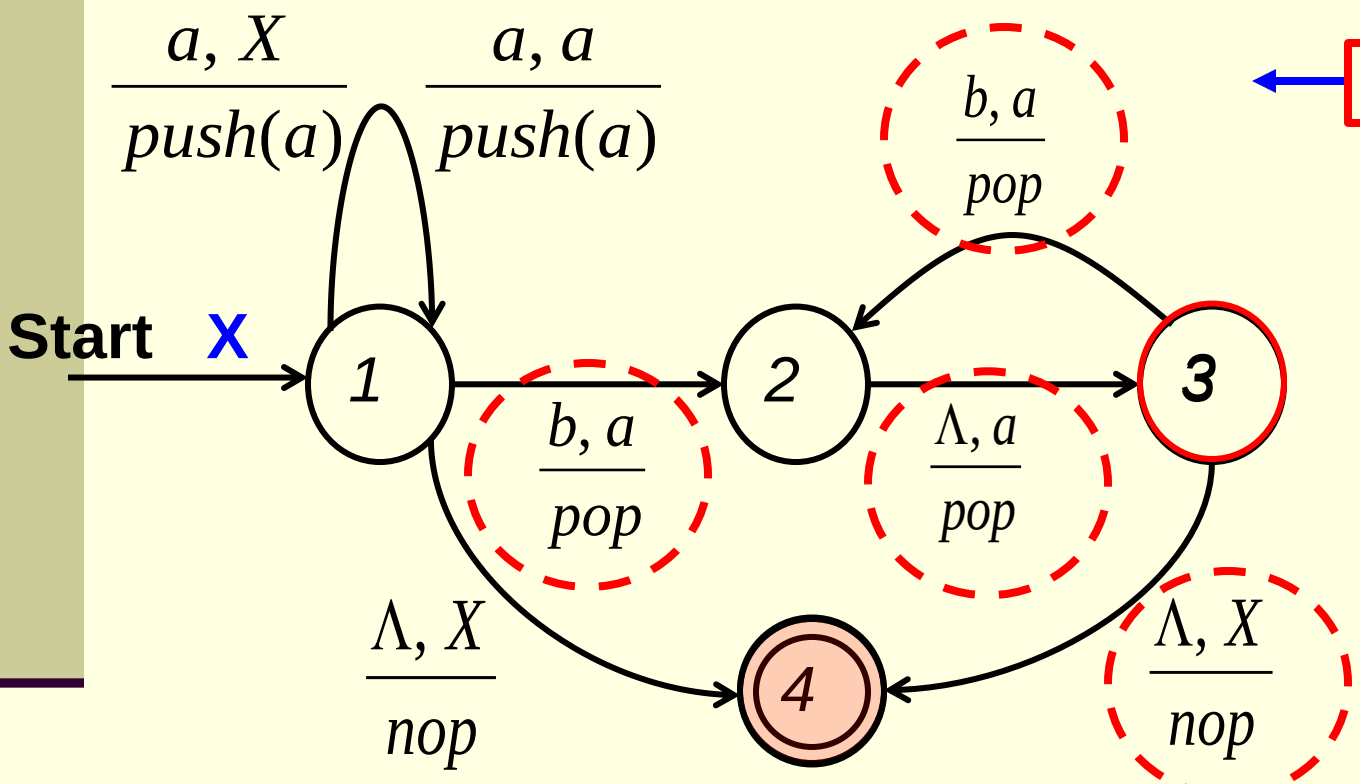
Acceptance of a string:

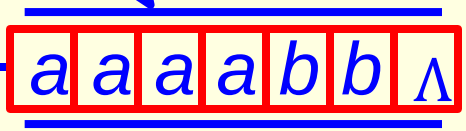
A string w is accepted by a PDA if there is a path from the start state to a final state such that the symbols on the path edges concatenate to w. Otherwise, w is rejected.

This kind acceptance is called *Final State Acceptance*

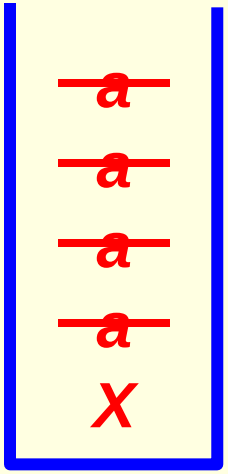
Final State acceptance

accepted





Input string



stack

b a a b a a

Output queue

Final State acceptance *vs* Empty-stack acceptance

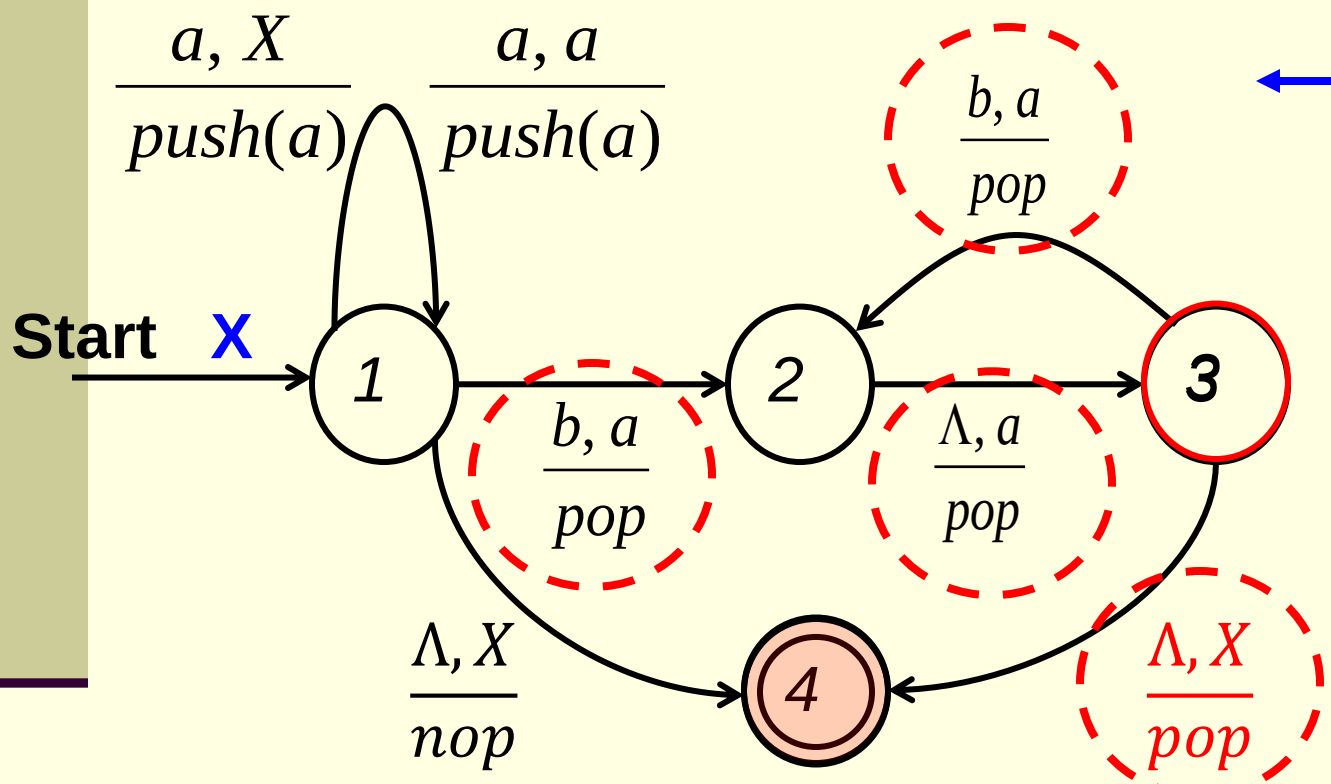
Empty-Stack Acceptance.

Instead of final-state acceptance, a PDA can also be defined to accept a string by the condition that the stack is empty.

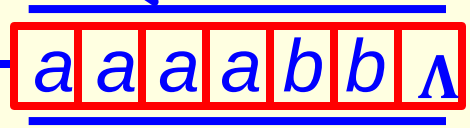
The two types of acceptance are equivalent in that they define PDAs that accept the same class of languages.

empty-stack PDA ⇔ final-state PDA

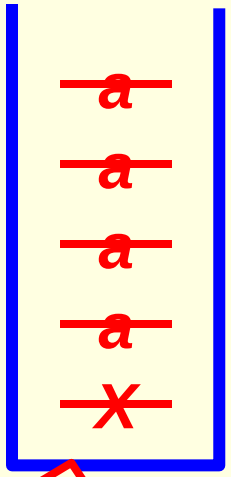
Empty Stack acceptance



accepted



Input string



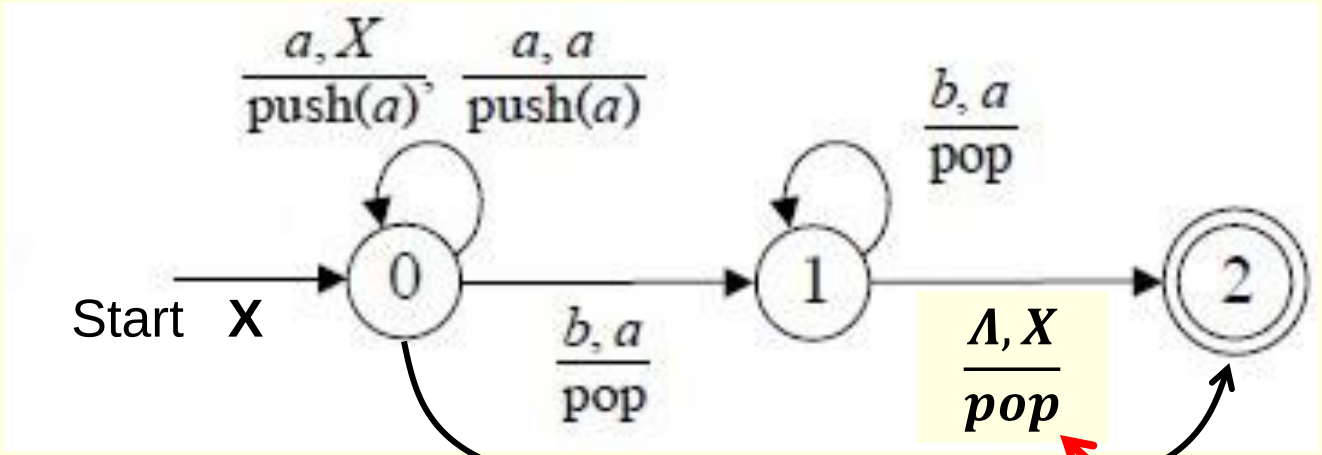
Stack is empty

b a a b a a

Output queue

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

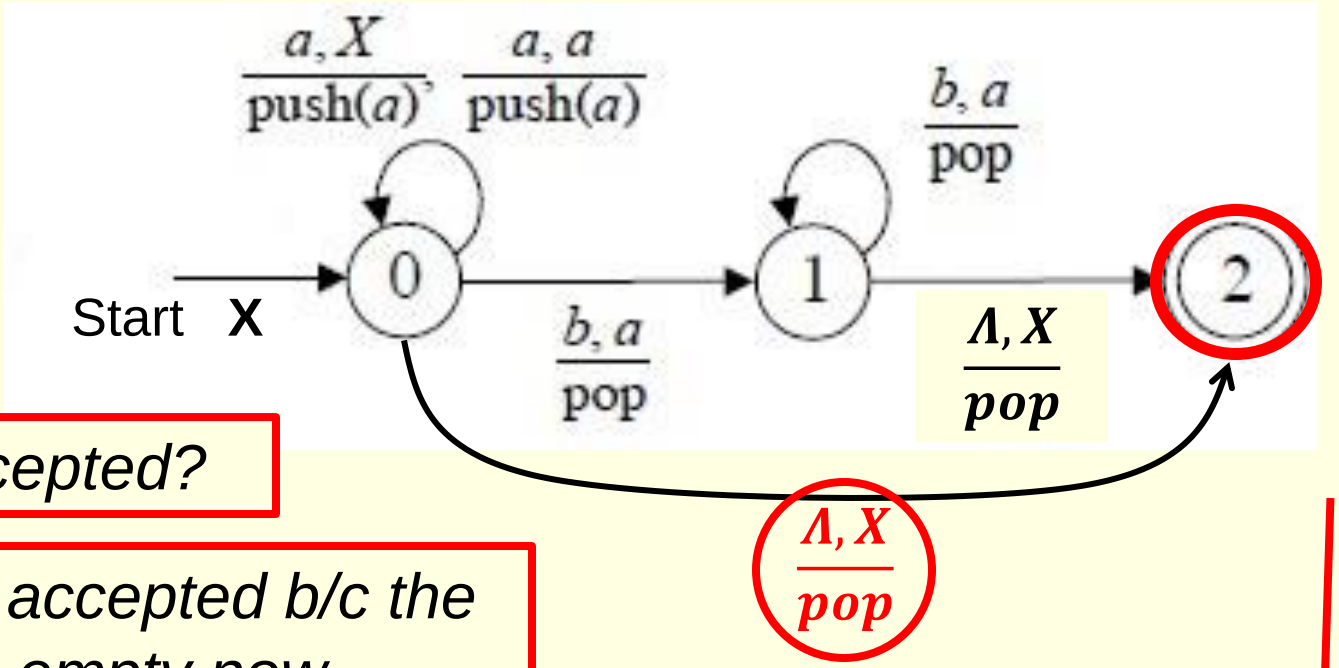
Question: is the following PDA a final-state acceptance PDA for the language L = { $a^n b^n$ | $n \in N$ } ?



YES

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Question: is the following PDA an empty-stack acceptance PDA for the language L = { $a^n b^n \mid n \in N$ } ?

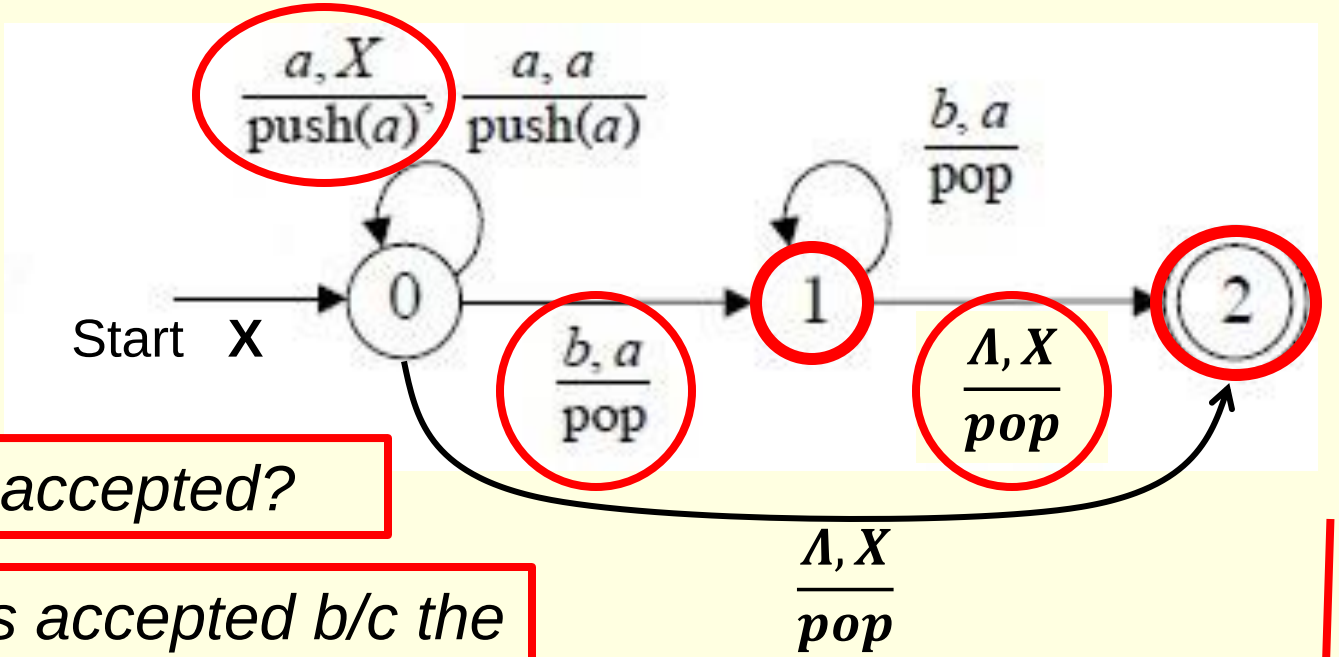


Is Λ accepted?

So Λ is accepted b/c the stack is empty now

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Question: is the following PDA an empty-stack acceptance PDA for the language L = { $a^n b^n$ | $n \in N$ } ?



Is ~~ab~~Λ accepted?

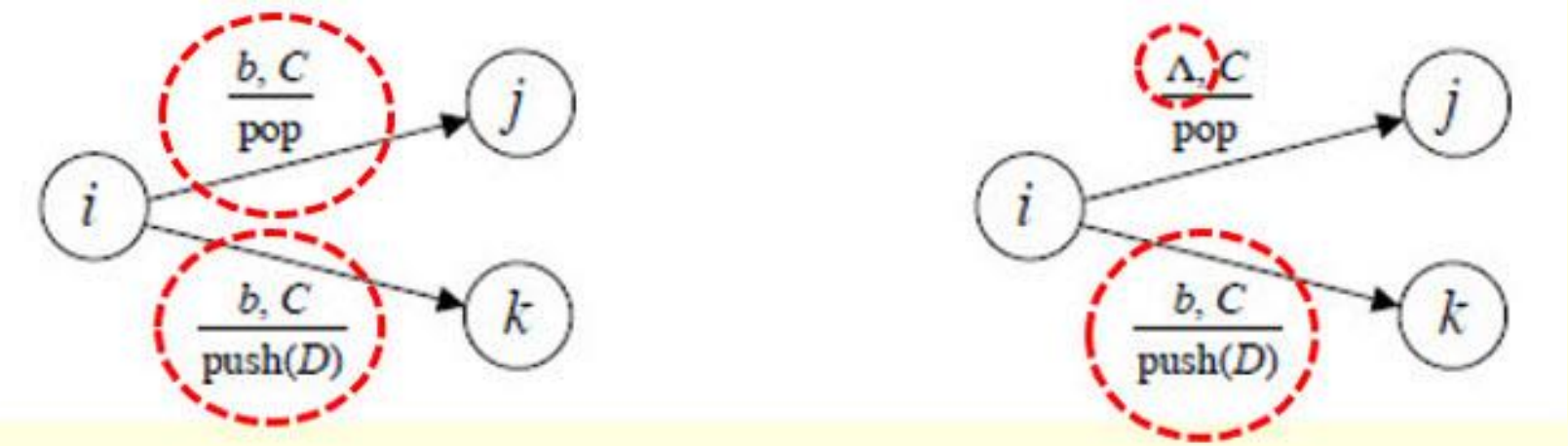
So ab is accepted b/c the stack is empty now

Try aabb, aaabbb, … yourself

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Deterministic PDA: at most one legal transition for the same combination of input symbol, state and top stack symbol

Nondeterminism can occur in two ways:



7

A

$\delta(i, a\ C)$ means when you are in state *i*, the next input symbol is a and the top element in the stack is *C*, the number of instructions that can be used for this combination

Deterministic PDA: at most one legal transition for the same combination of input symbol, state and top stack

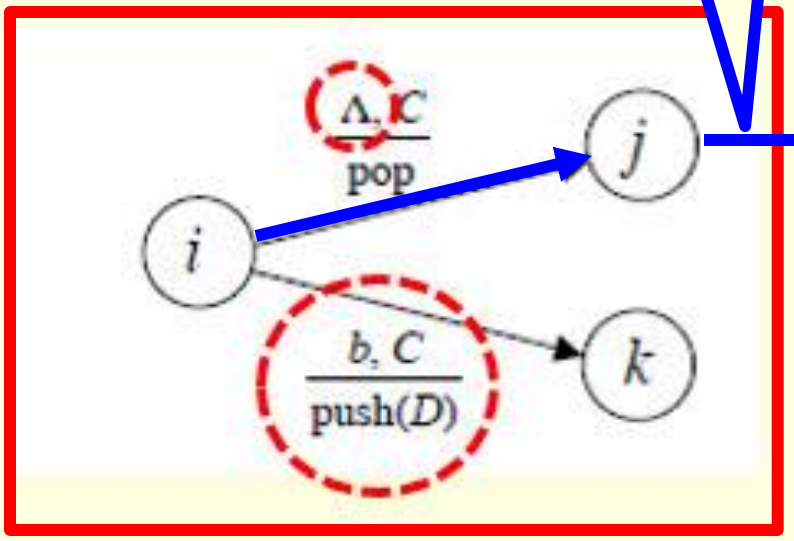
$\frac{b, D}{pop}$

Nondeterminism: the 2nd case:

b=Λb, hence,

if δ(i, Λ, C) is not empty,

then processing δ(i, b, C) could give us two different legal transitions. Why?

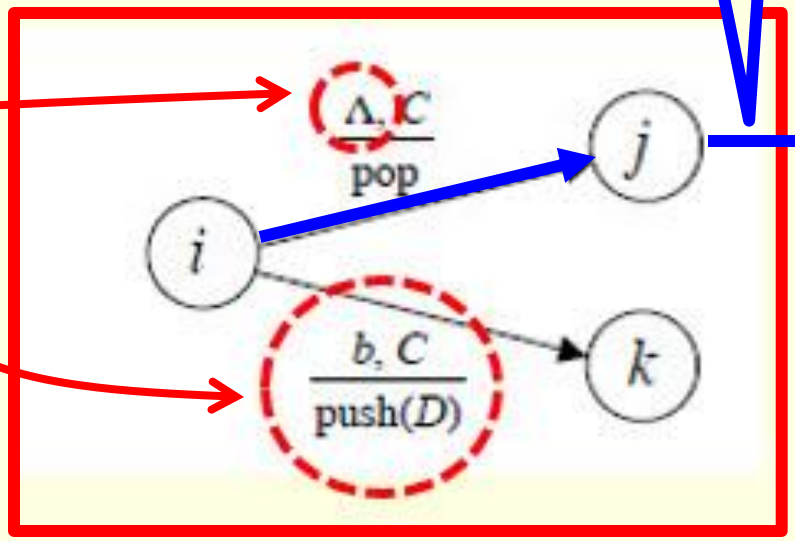


7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Deterministic PDA: at most one legal transition for the same combination of input symbol, state and top stack symbol

$\frac{b, D}{pop}$

Nondeterminism: the 2nd case:



In addition to getting δ(i, b, C),

we can get δ(i, Λ, C) first and

then followed by δ(j, b, D) where

j is the next state of δ(i, Λ, C) and

D is a stack symbol right below C (after popping C).

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Intuitively,

Deterministic **PDA**: never have a choice

Nondeterminisitic **PDA**: may have several alternatives how to continue

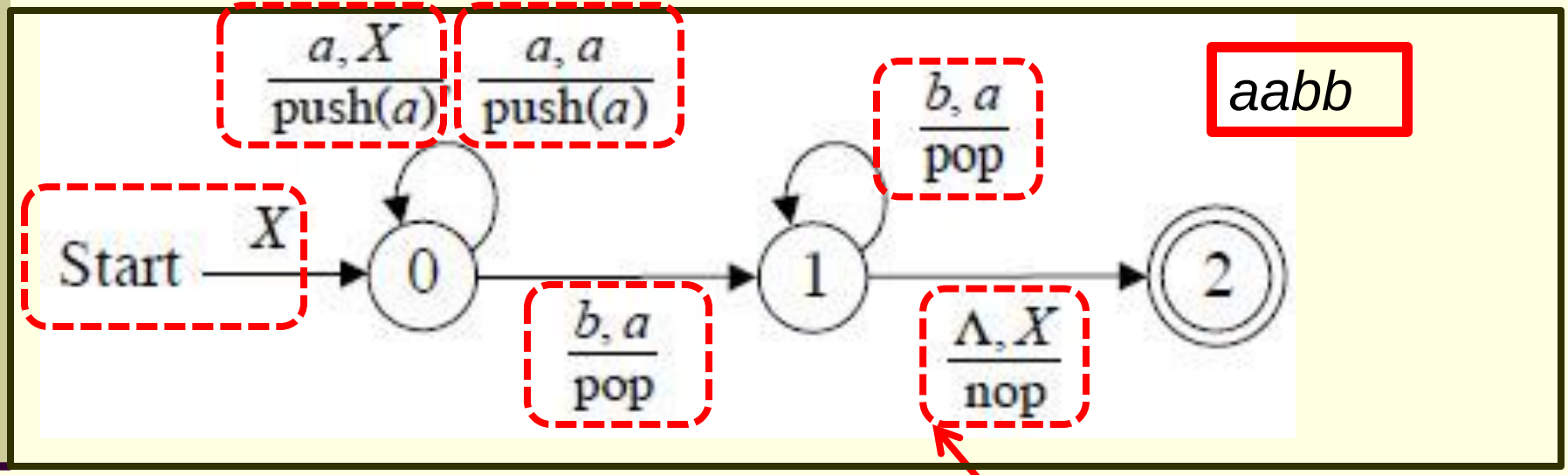
A PDA is deterministic if

$$|\delta(i, a, C)| + |\delta(i, \Lambda, C)| \leq 1$$

for any state *i*, any alphabet symbol *a,* any stack symbol *C.*

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example. A PDA to accept the language $\{ a^n b^n \mid n > 0 \}$ as a graph and as a set of 5-tuples.



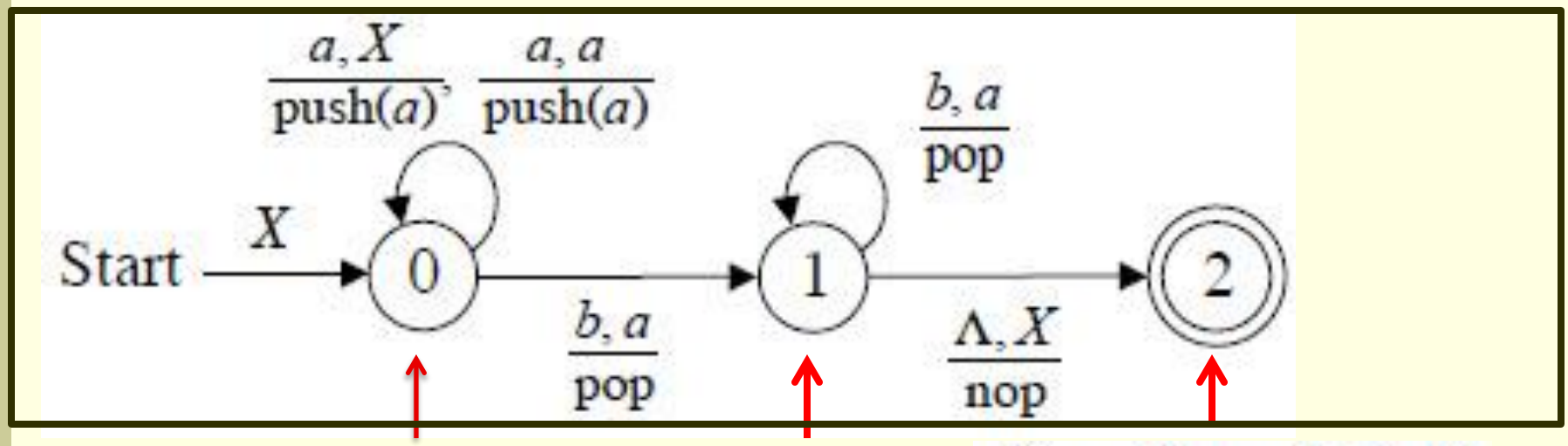
Deterministic or nondeterministic?

Will this PDA accept aabbb?

NO. Why?

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example. A PDA to accept the language $\{ a^n b^n \mid n > 0 \}$ as a graph and as a set of 5-tuples.



Consider: $aaabbb\Lambda$

Accepted (final state)

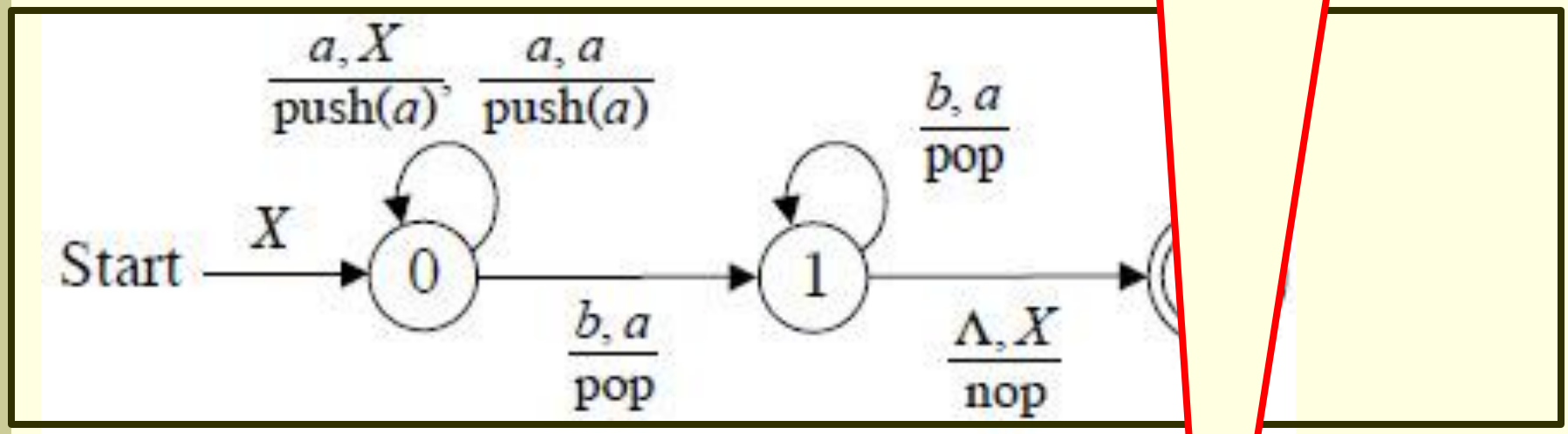
→ $(0, a, X, \text{push}(a), 0)$
→ $(0, a, a, \text{push}(a), 0)$
→ $(0, b, a, \text{pop}, 1)$
→ $(1, b, a, \text{pop}, 1)$
→ $(1, \Lambda, X, \text{nop}, 2)$.

7. Context-Free La

Automata- Pushdown Automata

The stack plays the role of a counter here, implicitly.

Example. A PDA to accept the language $\{ a^n b^n$ | as a graph and as a set of 5-tuples.

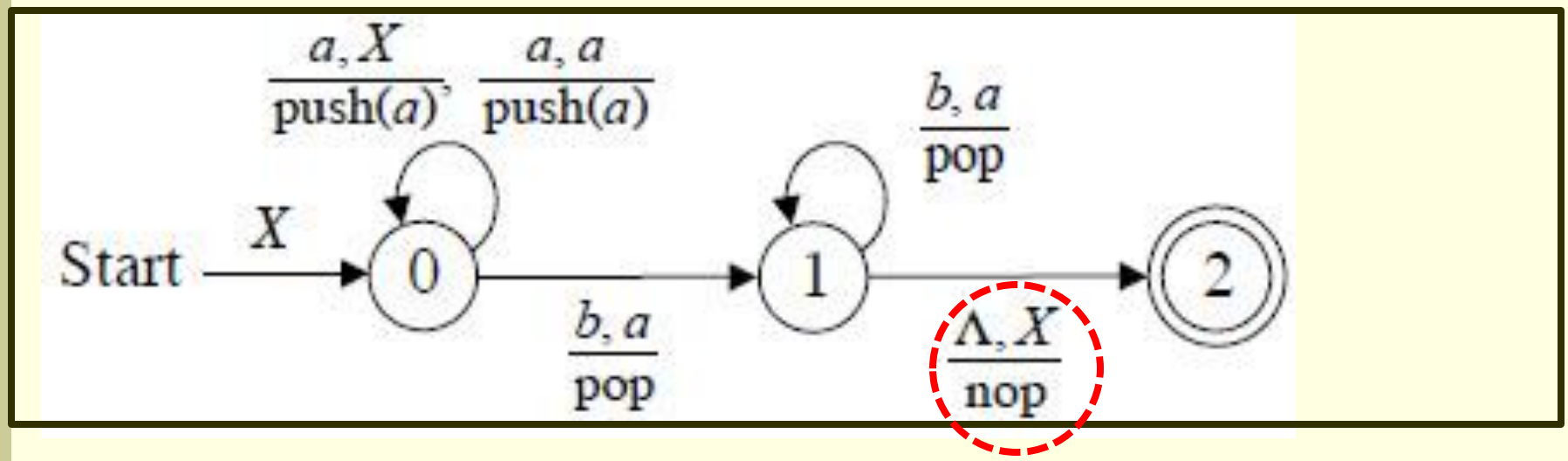


In this case, we can recover all the a's in the given string *aaabbb*. (output: bababa)

Each time we pop an 'a' out from the stack, we get one 'a' back. What is the significance of this?

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example. A PDA to accept the language $\{ a^n b^n \mid n > 0 \}$ as a graph and as a set of 5-tuples.

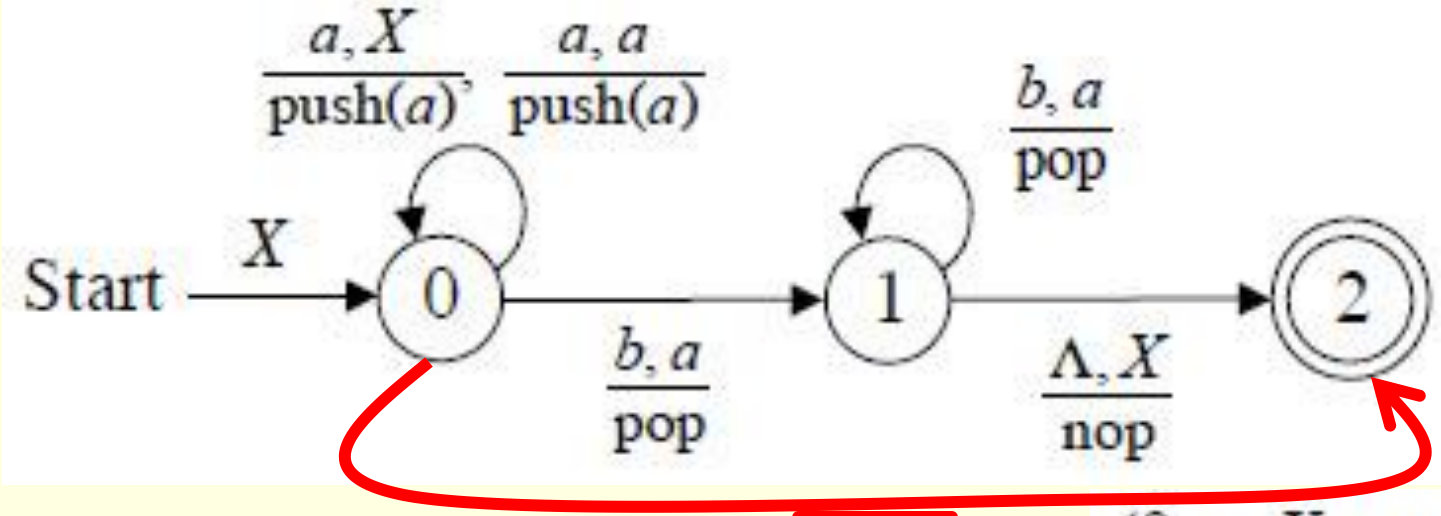


This PDA does not accept *aabbab,* but would accept *aabb*. Why?

Because after the substring *aabb* is processed, we are in state 1 and we can only execute the instruction Λ,X/nop then.

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Quiz. How should you modify the machine to accept
$\{a^n b^n \mid n \in N\}$ **(final state)*?***



Answer: Add the instruction

$(0, \Lambda, X, \text{nop}, 2)$

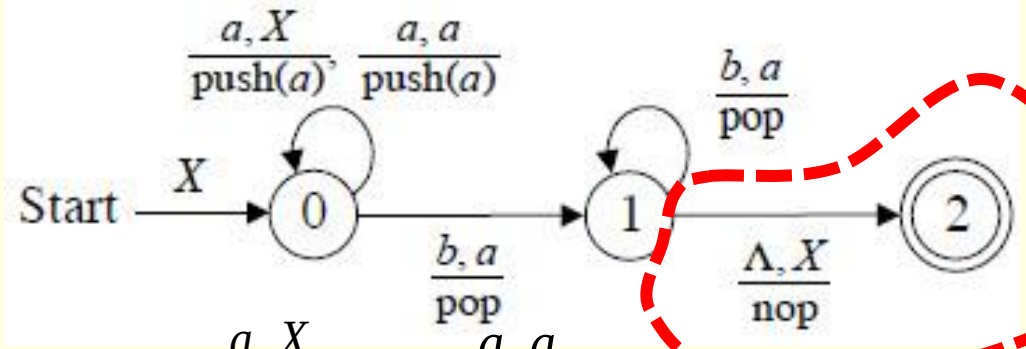
$$\frac{\Lambda, X}{nop}$$

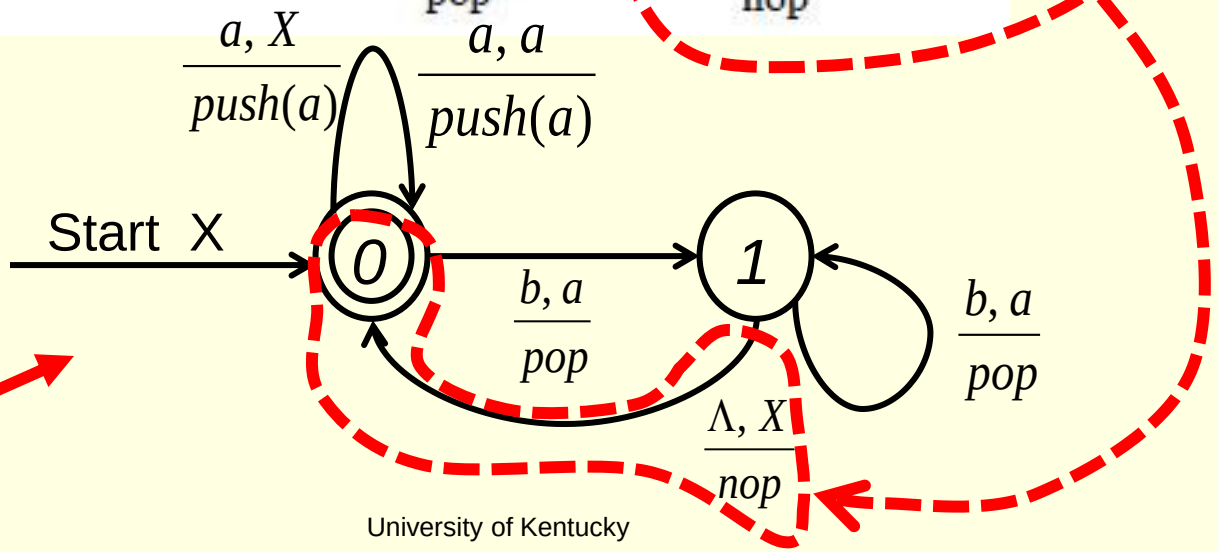
$(0, a, X, \text{push}(a), 0)$
$(0, a, a, \text{push}(a), 0)$
$(0, b, a, \text{pop}, 1)$
$(1, b, a, \text{pop}, 1)$
$(1, \Lambda, X, \text{nop}, 2).$

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example. A PDA to accept the language $\{ a^n b^n \mid n > 0 \}$ as a graph and as a set of 5-tuples.

Instead of



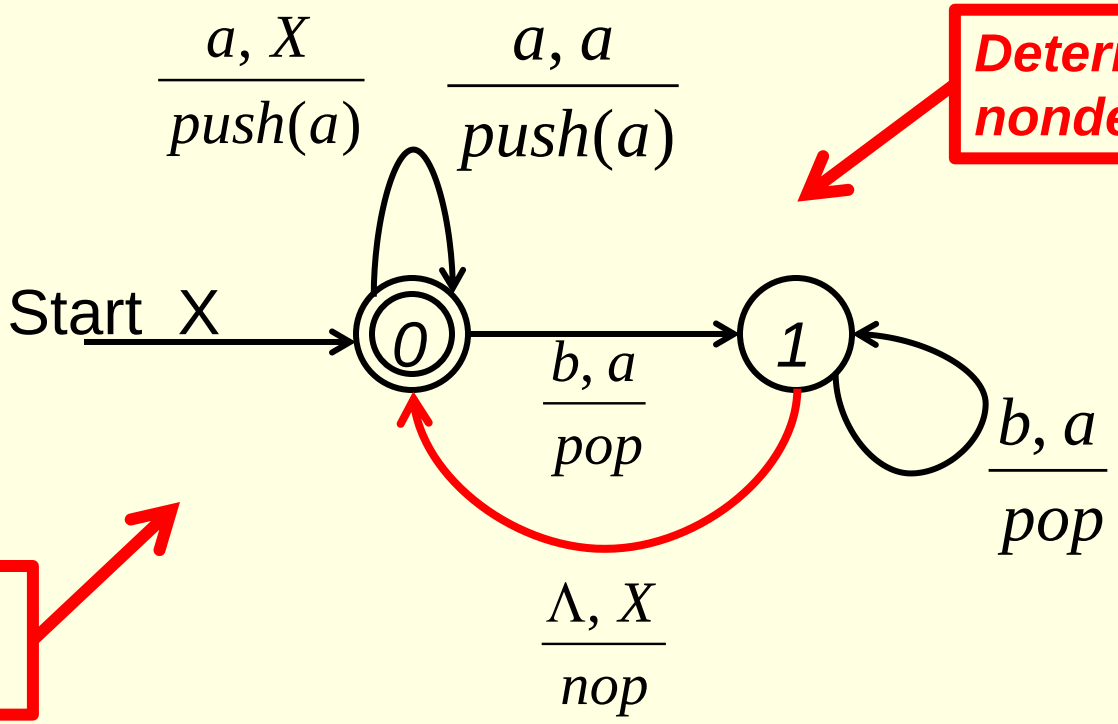


Would this PDA also work?

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example. A PDA to accept the language { $a^n b^n \mid n > 0$} as a graph and as a set of 5-tuples.

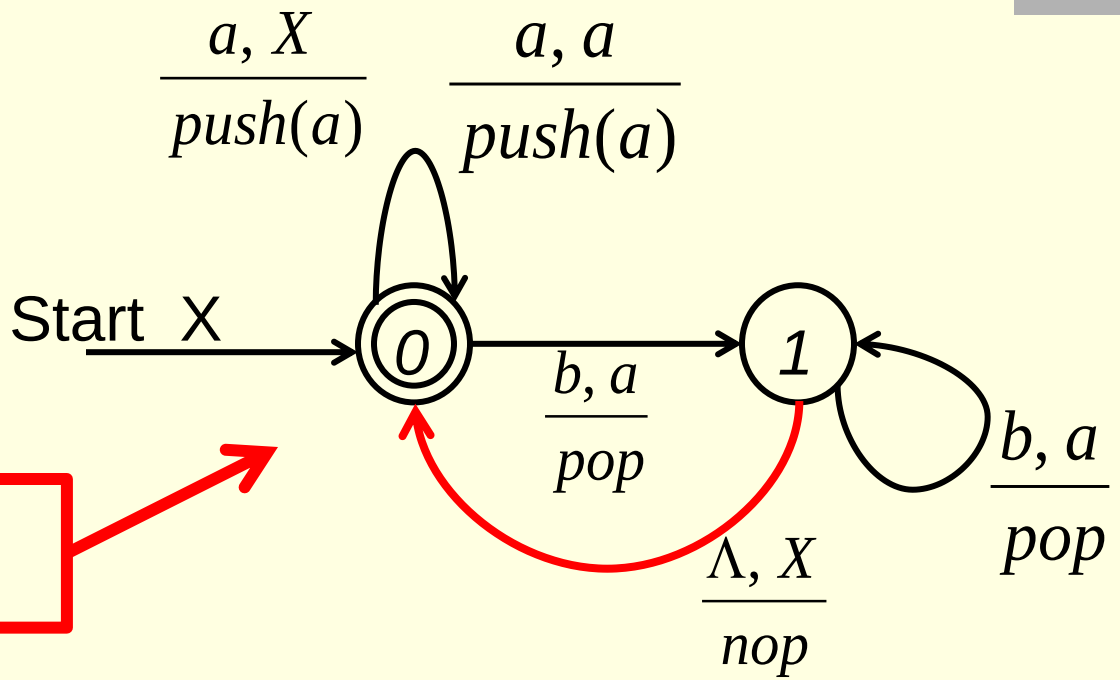
Take a closer look



Deterministic or nondeterministic?

Would this PDA also work?

Example. A PDA to accept the language $\{ a^n b^n \mid n > 0 \}$ as a graph and as a set of 5-tuples.



Would this PDA also work?

NO

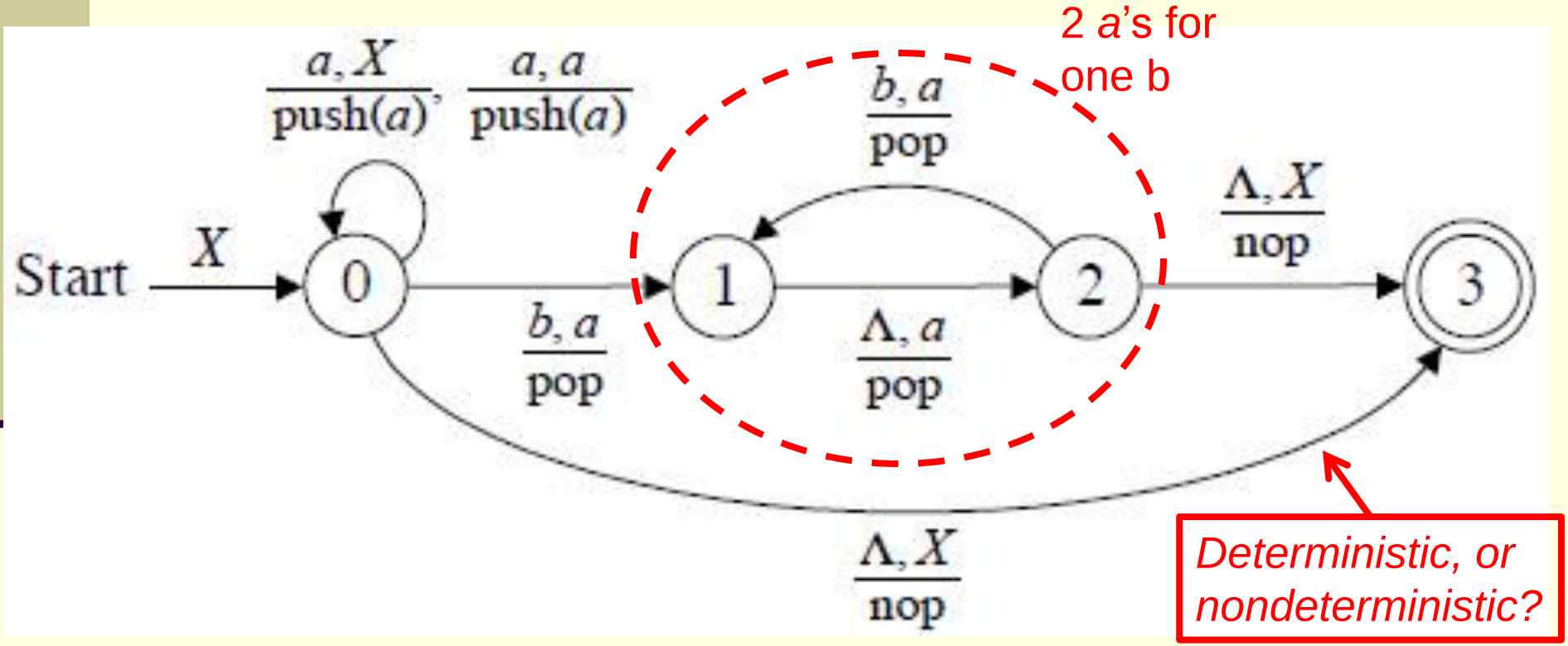
This PDA accepts
$\{(a^{n_1}b^{n_1})(a^{n_2}b^{n_2})\cdots(a^{n_k}b^{n_k})a^m \mid n_i, k, m \in N\}$

such as: Λ, a, aa, aabb, aabbaa, …, aabbaaabbbaa, …

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example. Find a PDA to accept the language $\{a^{2n}b^n \mid n \in \mathbf{N}\}$.

A solution:



Does every high-level programming language have a **deterministic PDA**?

YES or NO

Yes. Why? because each high-level language has a context-free grammar, and a CFG is equivalent to a DPDA

7. Context-Free Languages & Pu

Automata- Pushdown Automata

Plays the role of a stack

An important definition:

Instantaneous Description (ID) :

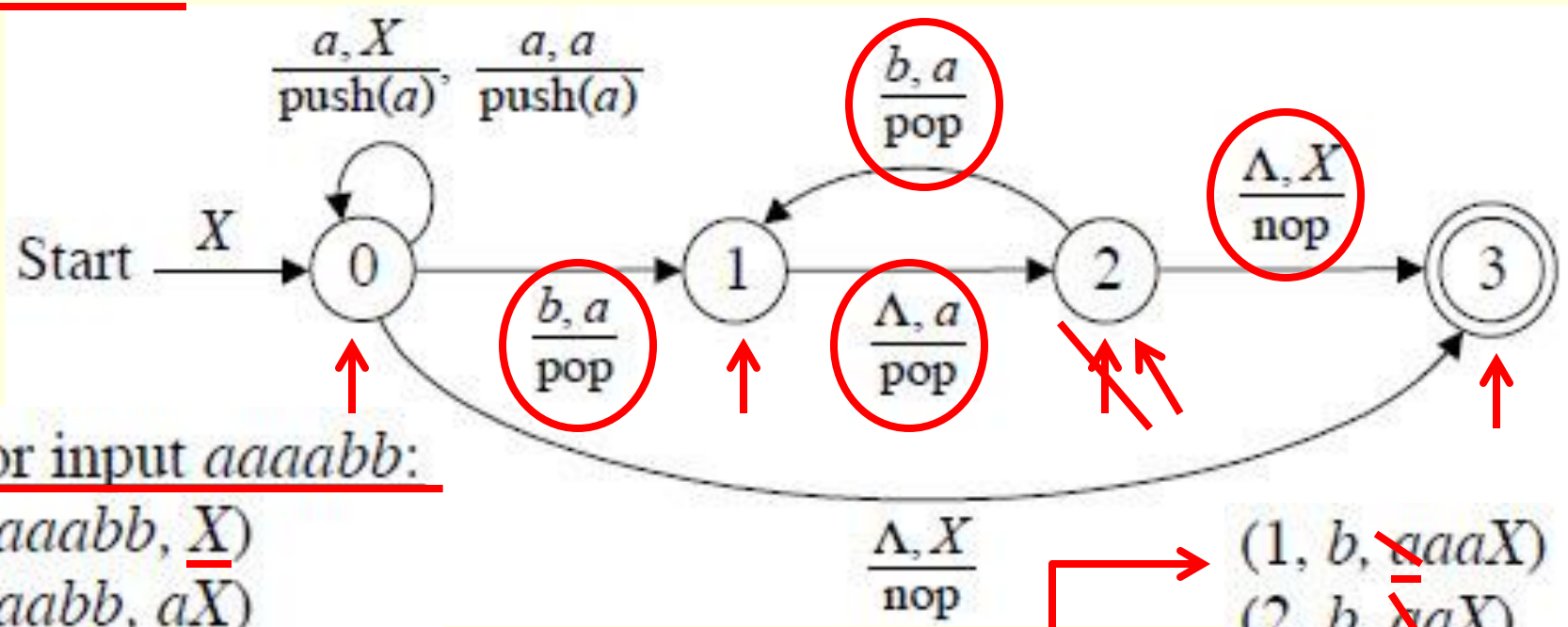
(current state, unconsumed input, stack contents)

e.g., *(0, aaaabb, X)* : *start in state 0 with X on the stack and the input string being aaaabb*

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example. Find a PDA to accept the language $\{a^{2n}b^n \mid n \in \mathbf{N}\}$.

A solution:



ID for input *aaaabb*:

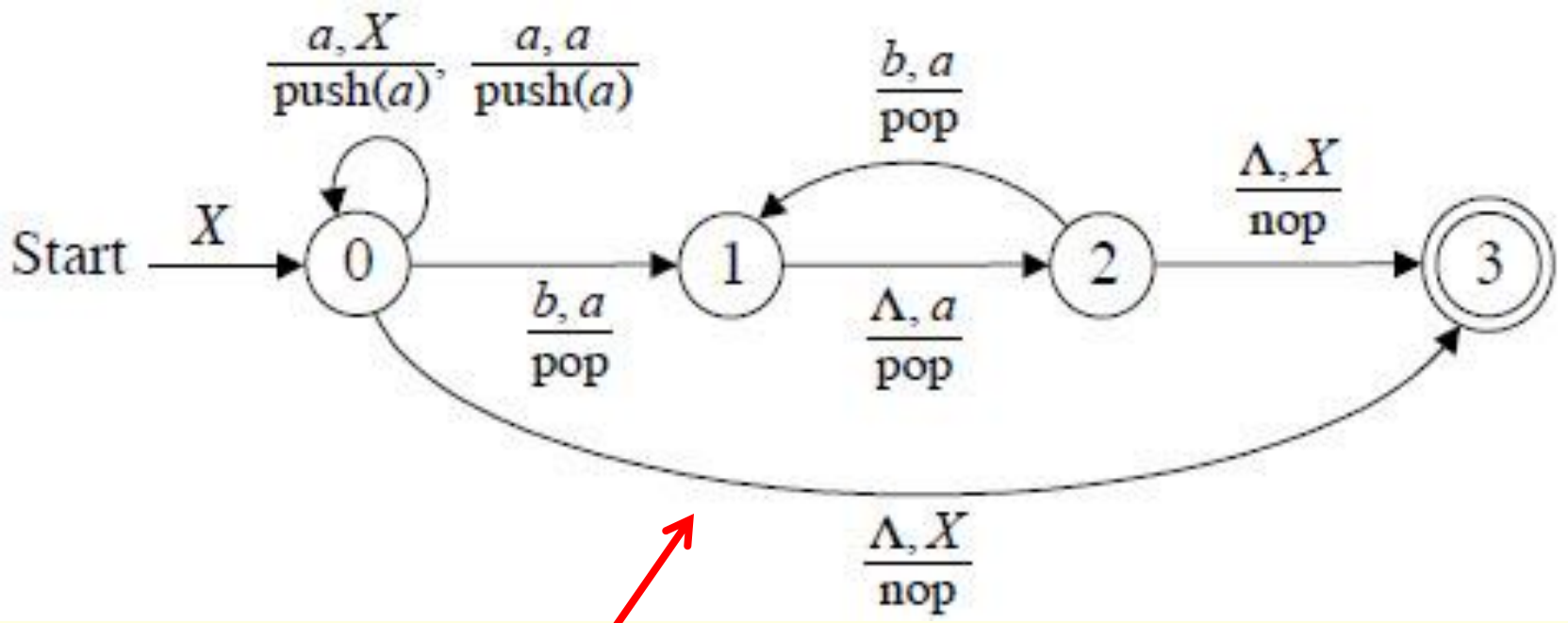
(0, *aaaabb*, *X*)
(0, *aaabb*, *aX*)
(0, *aabb*, *aaX*)
(0, *abb*, *aaaX*)
(0, *bb*, *aaaaX*)

(1, *b*, *aaaX*)
(2, *b*, *aaX*)
(1, Λ, *aX*)
(2, Λ, *X*)
(3, Λ, *X*) accepted

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example. Find a PDA to accept the language $\{a^{2n}b^n \mid n \in \mathbf{N}\}$.

A solution:



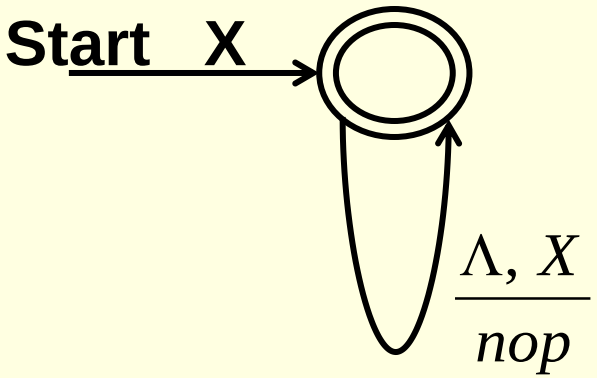
How do I know the PDA should be built this way?

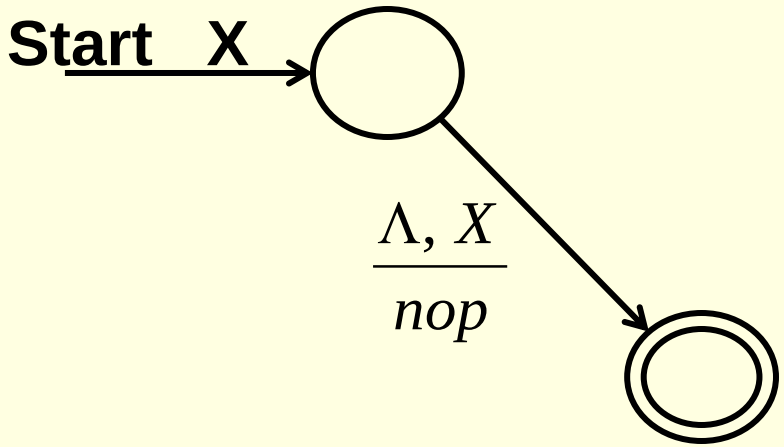
7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example. Find a PDA to accept the language $\{a^{2n}b^n \mid n \in \mathbf{N}\}$.

A solution:

First, to accept Λ *, we have two choices:*

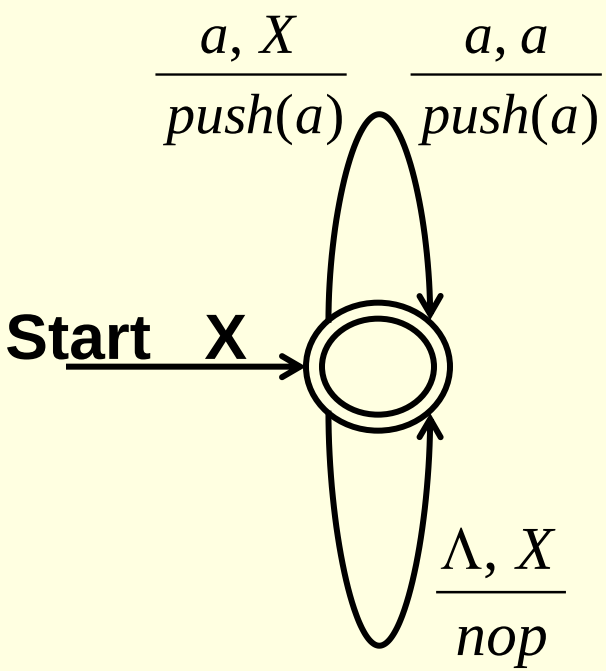


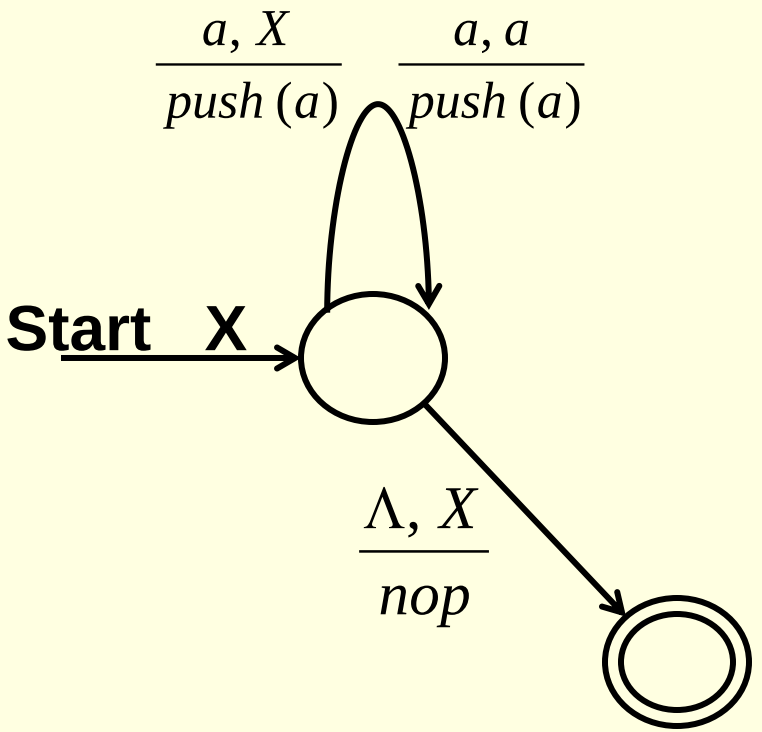


7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example. Find a PDA to accept the language $\{a^{2n}b^n \mid n \in \mathbf{N}\}$.

A solution: Then, build mechanism to push a^m ***onto the stack***

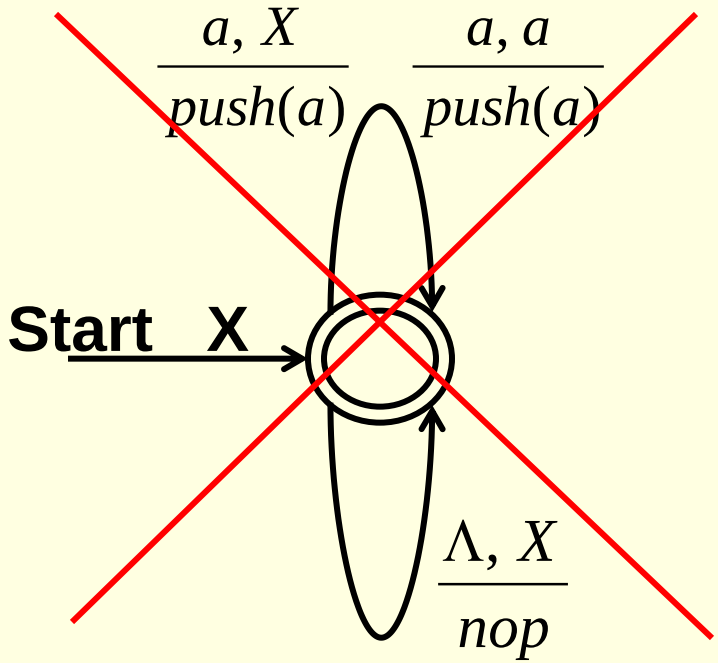


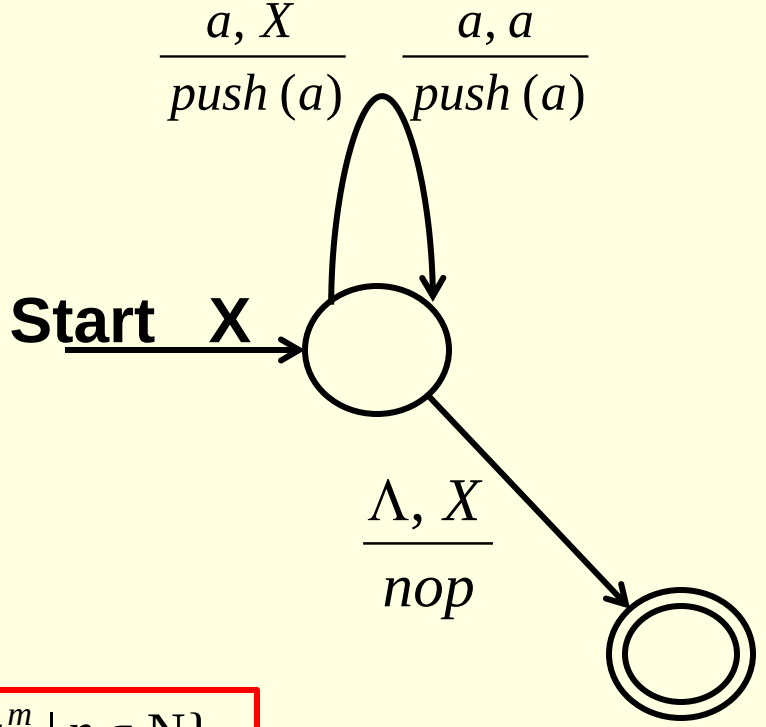


7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example. Find a PDA to accept the language $\{a^{2n}b^n \mid n \in \mathbf{N}\}$.

A solution: Then, build mechanism to push a^m ***onto the stack***



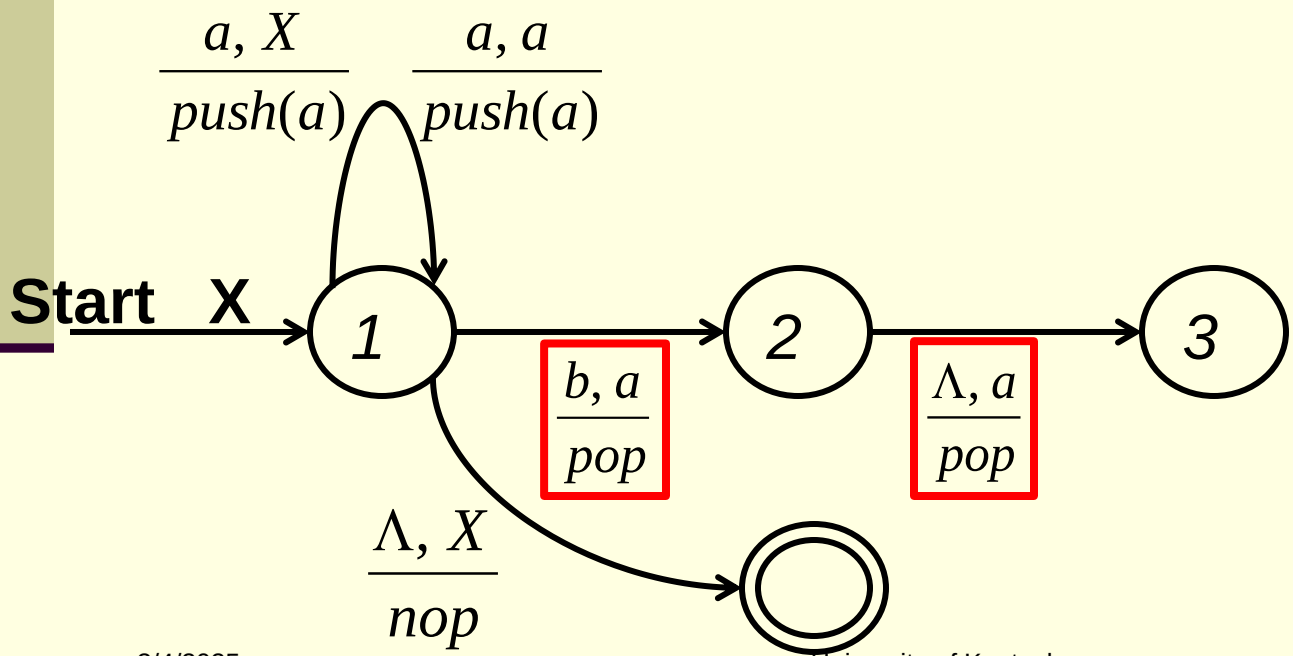


Ignore this one because it accepts $\{a^m \mid n \in \mathrm{N}\}$

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example. Find a PDA to accept the language $\{a^{2n}b^n \mid n \in \mathbf{N}\}$.

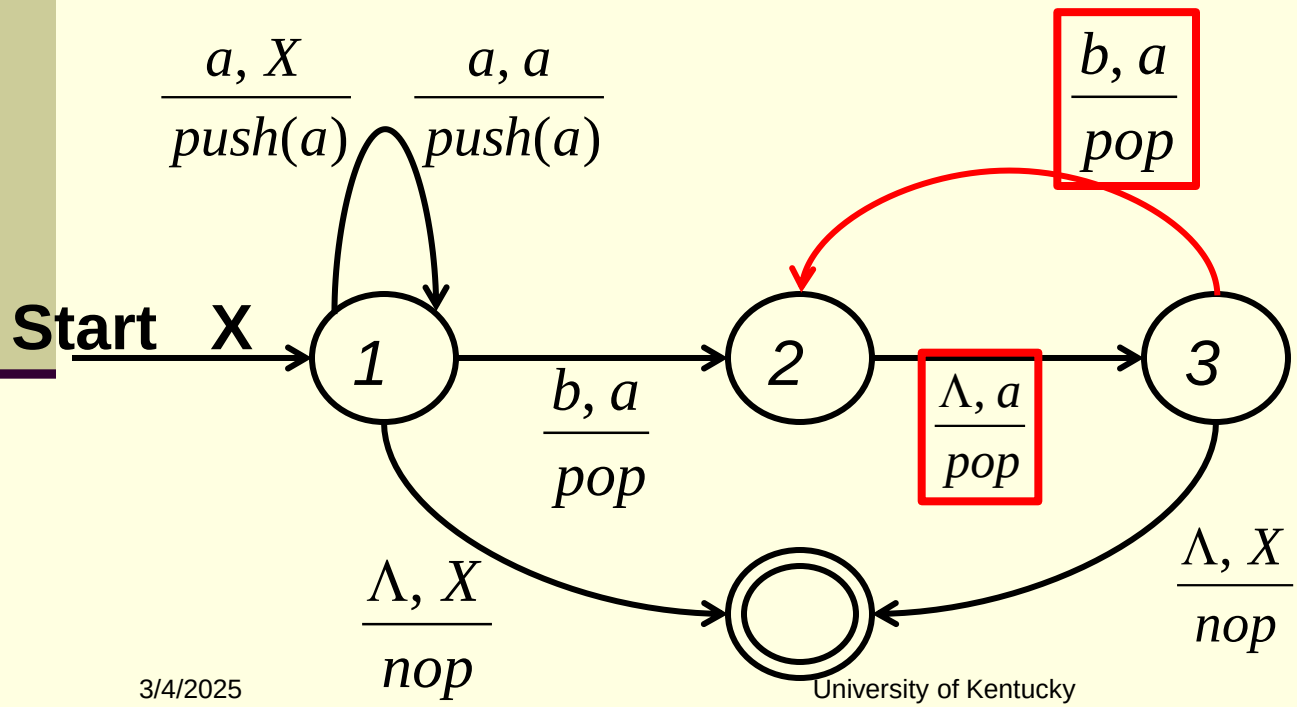
A solution: ***Then, build mechanism to pop*** a ***from the stack 2 times for a given*** b



7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example. Find a PDA to accept the language $\{a^{2n}b^n \mid n \in \mathbf{N}\}$.

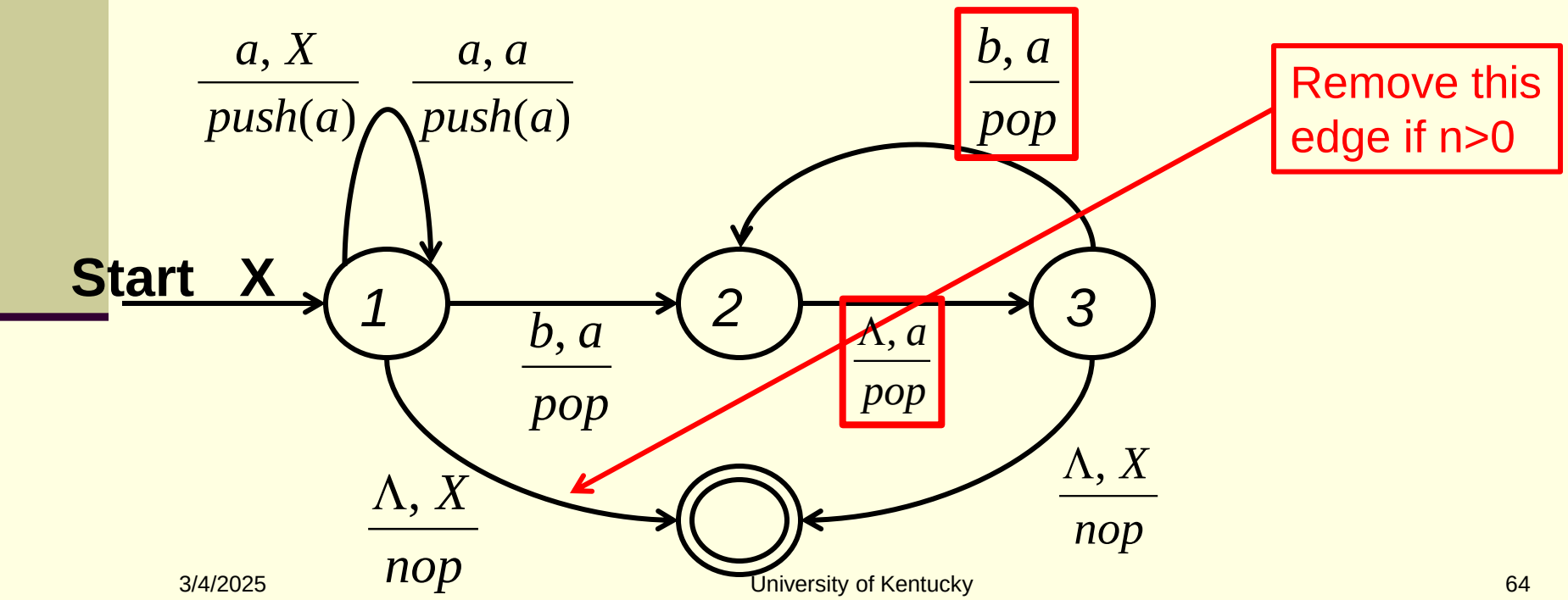
A solution: ***Then, build mechanism to pop*** a ***from the stack 2 times for each*** b, ***to accept*** $a^{2n}b^n$



7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example. Find a PDA to accept the language $\{ a^{2n} b^n \mid n \in \mathbf{N} \}$.

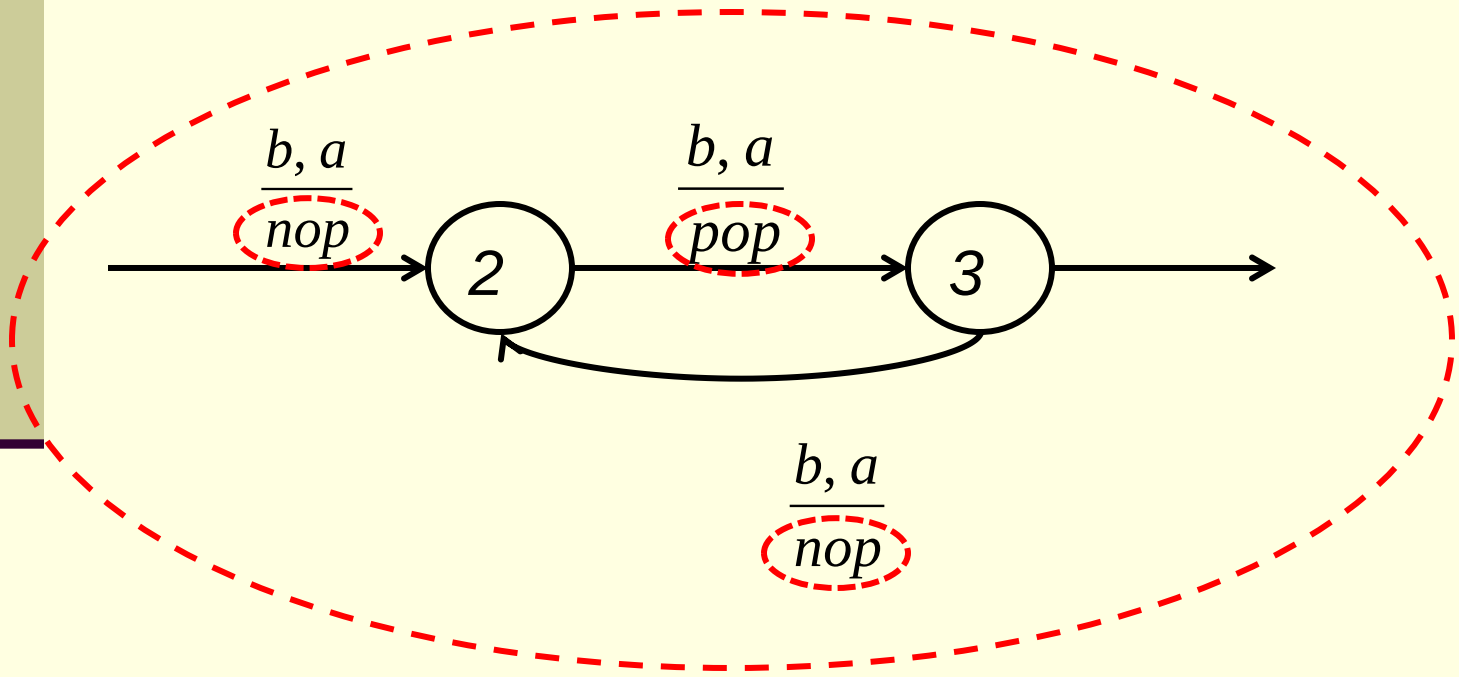
A solution: ***Then, build mechanism to pop*** a ***from the stack 2 times for each*** b, ***to accept*** $a^{2n} b^n$



7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example. Find a PDA to accept the language $\{a^n b^{2n} \mid n \in \mathbf{N}\}$.

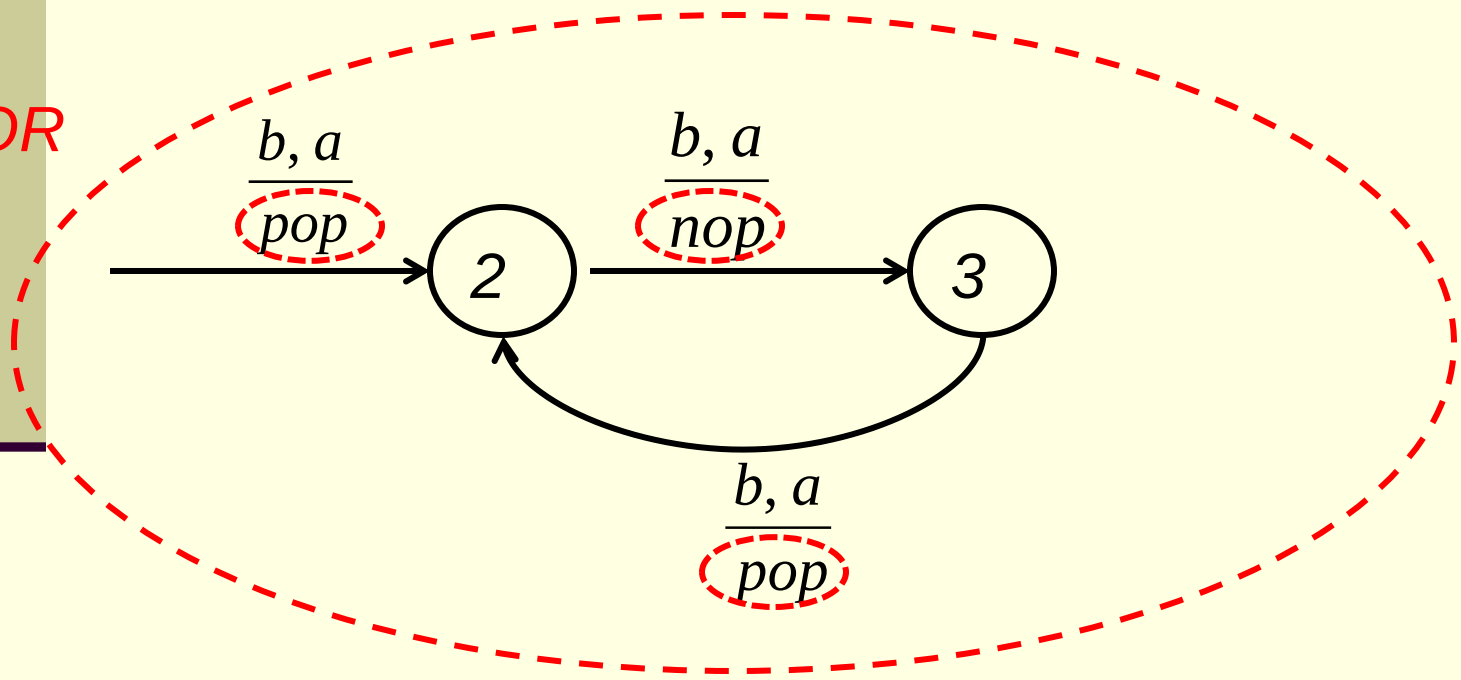
Idea: ***build mechanism to pop*** a ***from the stack once for for every two*** b***'s***



7. Context-Free Languages & Pushdown Automata- Pushdown Automata

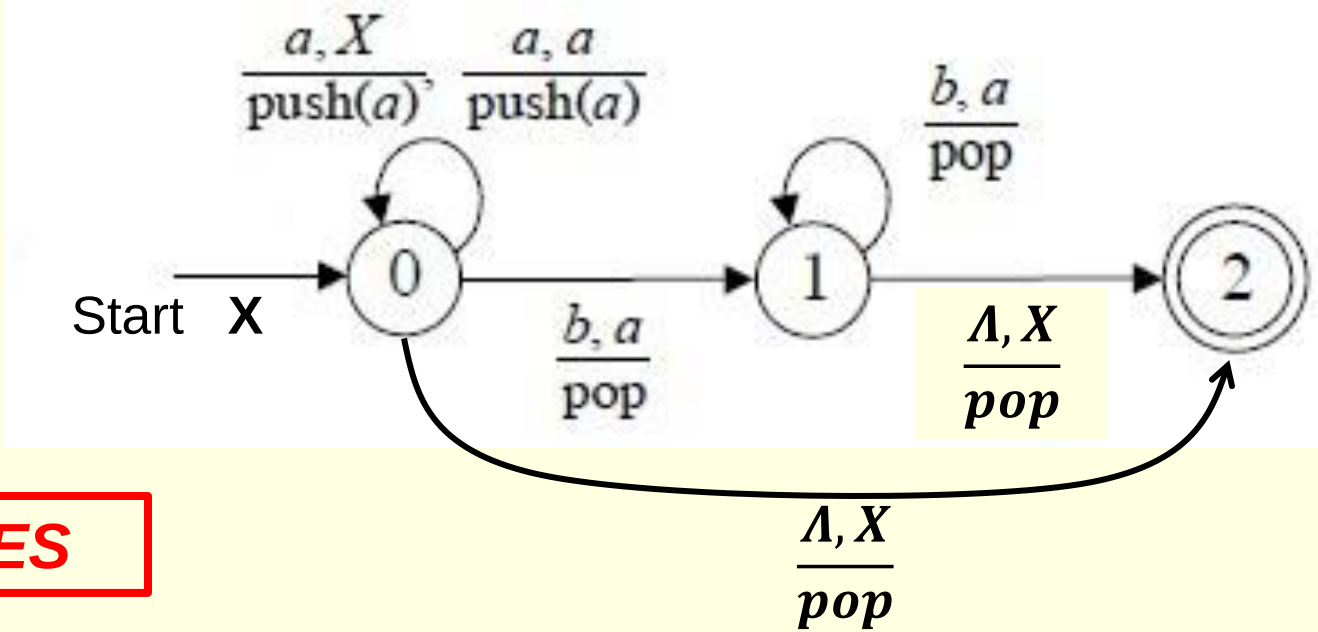
Example. Find a PDA to accept the language $\{a^n b^{2n} \mid n \in \mathbf{N}\}$.

Idea: ***build mechanism to pop*** a ***from the stack once for for every two*** b***'s***



7. Context-Free Languages & Pushdown Automata- Pushdown Automata

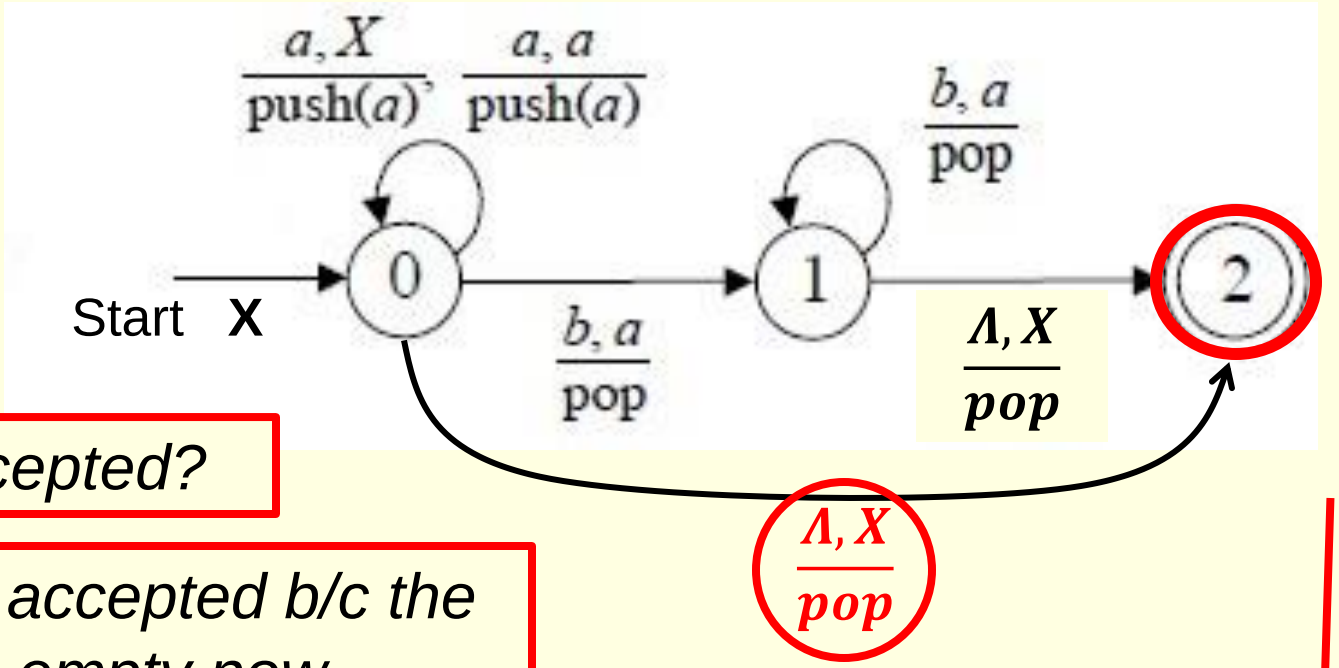
Question: is the following PDA a final-state acceptance PDA for the language L = { $a^n b^n$ | $n \in N$ } ?



YES

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Question: is the following PDA an empty-stack acceptance PDA for the language L = { $a^n b^n \mid n \in N$ } ?

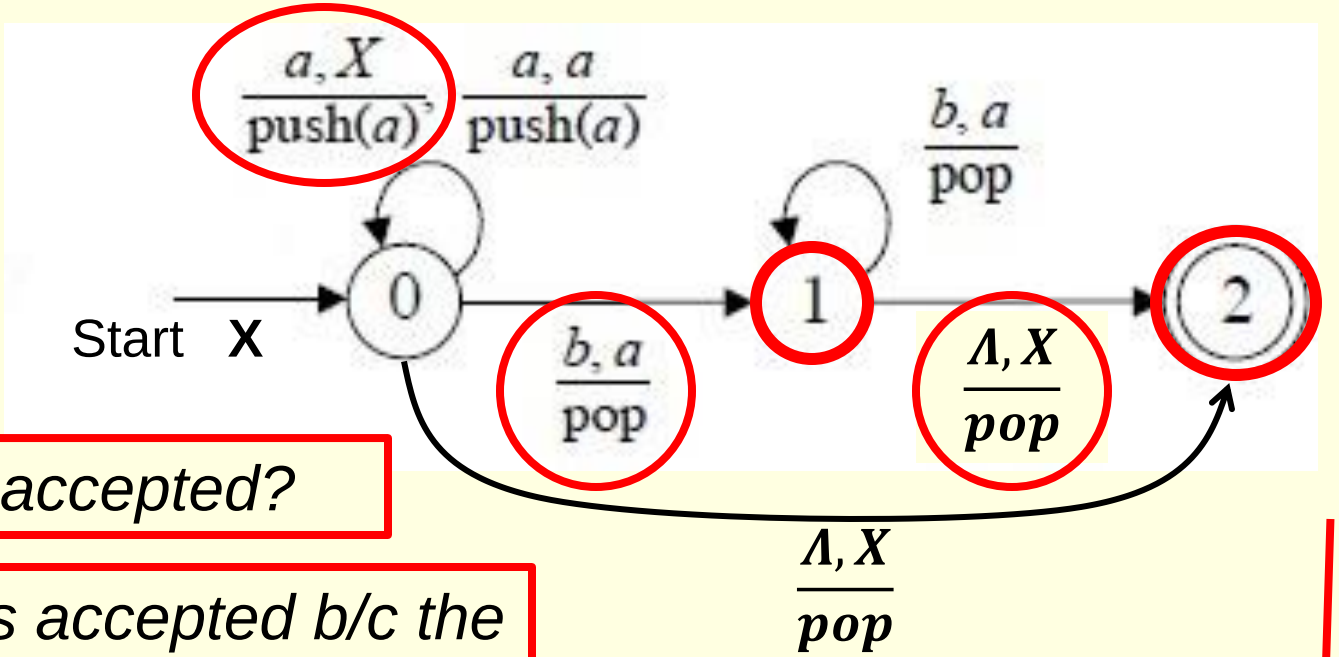


Is Λ accepted?

So Λ is accepted b/c the stack is empty now

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Question: is the following PDA an empty-stack acceptance PDA for the language L = { $a^n b^n$ | $n \in N$ } ?



Is ~~ab~~Λ accepted?

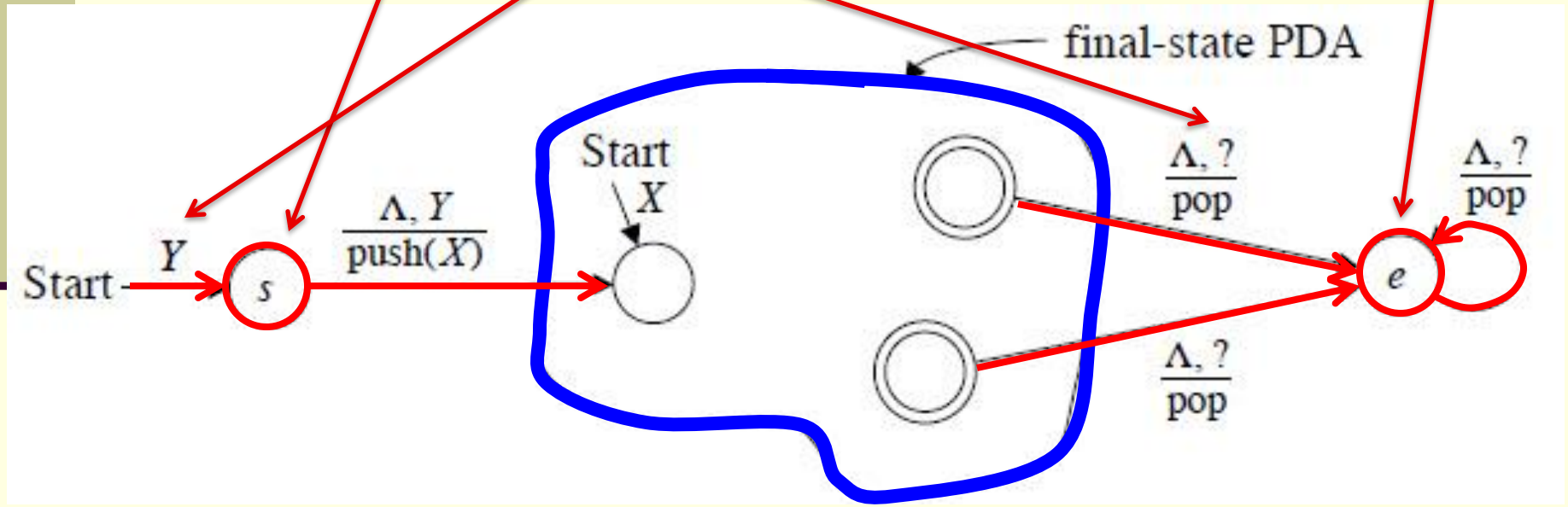
So ab is accepted b/c the stack is empty now

Try aabb, aaabbb, … yourself

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Transform final-state acceptance to empty-stack acceptance:

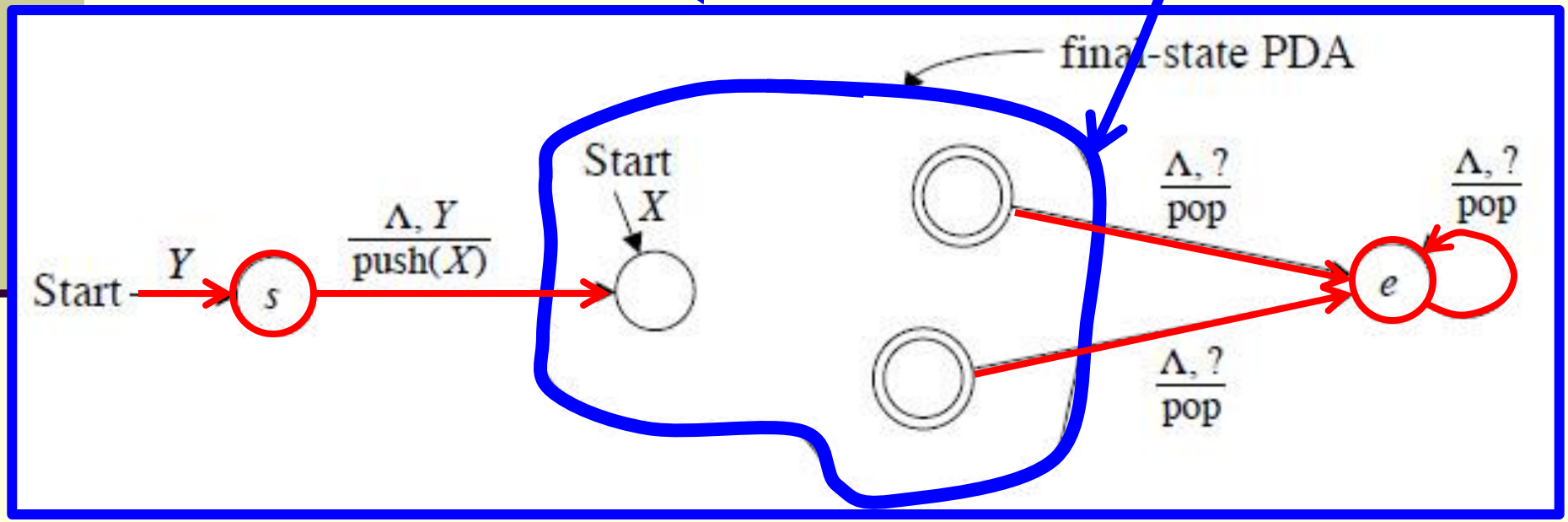
Introduce a new start state *s*, a new stack symbol *Y*, a new empty state *e*, and construct edges and actions as shown in the diagram below. Note that ? stands for any stack symbol.



The arrangement here ensures that if xxxx is accepted by the final-state PDA then /\xxxx/\ is accepted by the empty-stack PDA

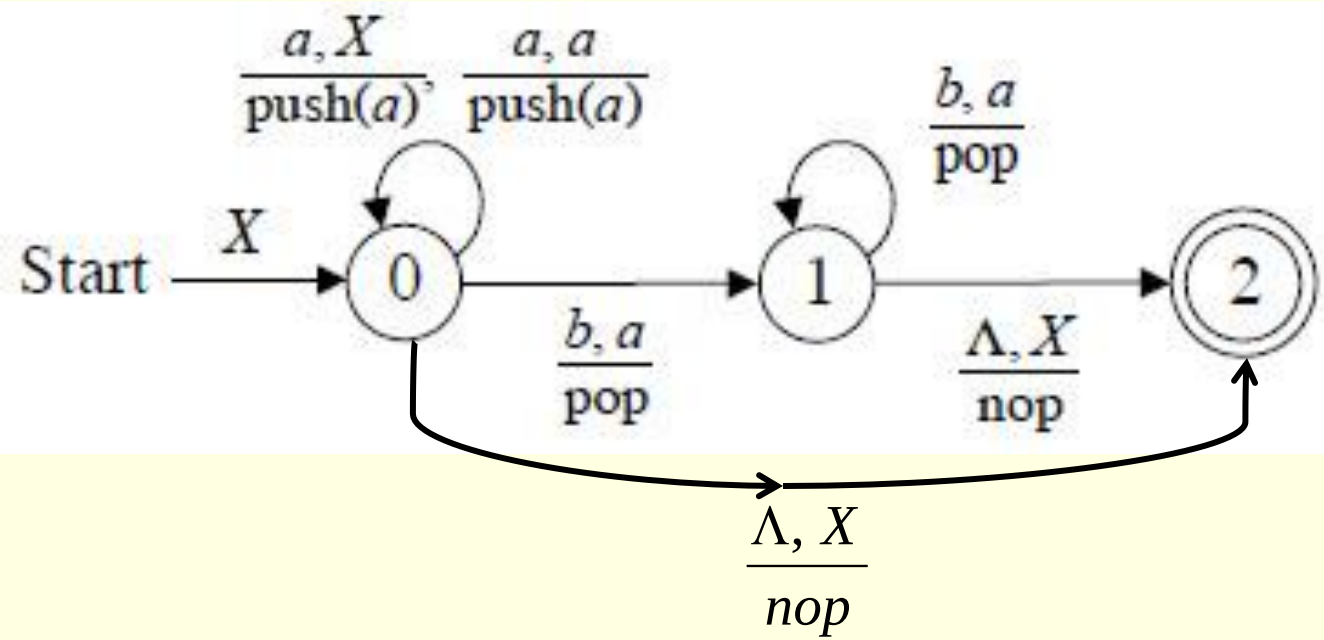
Transform final-state acceptance to empty-stack acceptance:

Introduce a new start state *s*, a new stack symbol *Y*, a new empty state *e*, and construct edges and actions as shown in the diagram below. Note that ? stands for any stack symbol.



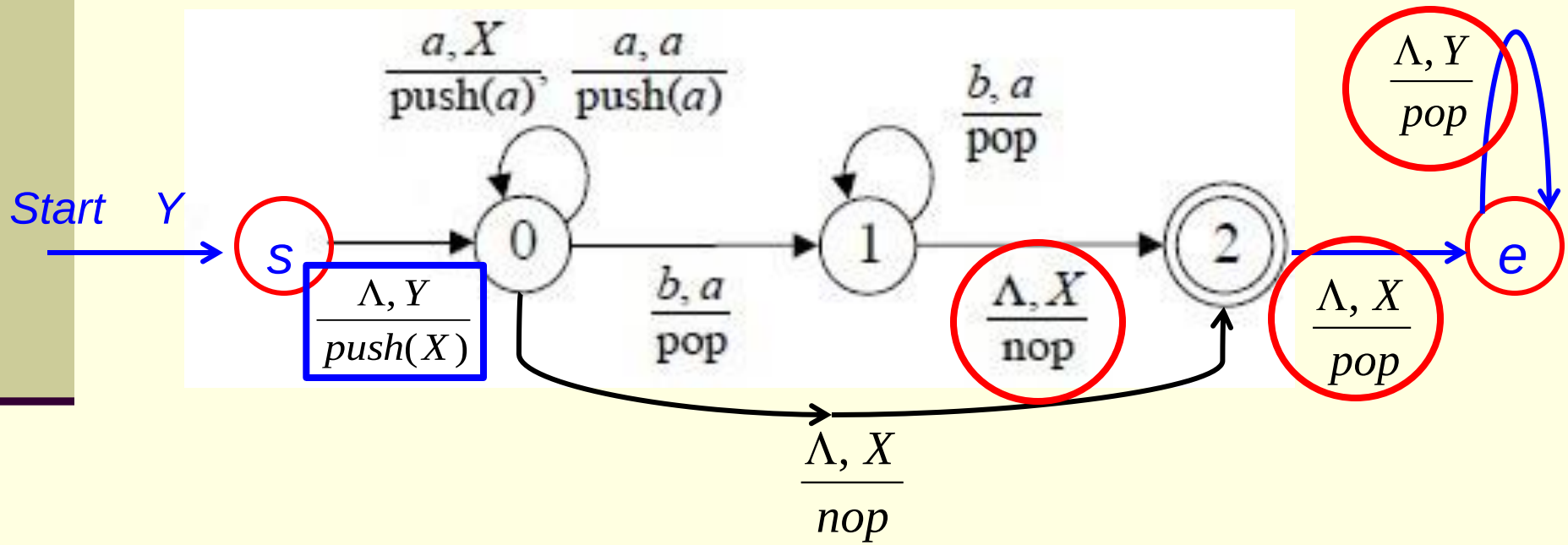
7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example: given the following final-state PDA, convert it to a empty-stack PDA



7. Context-Free Languages & Pushdown Automata- Pushdown Automata

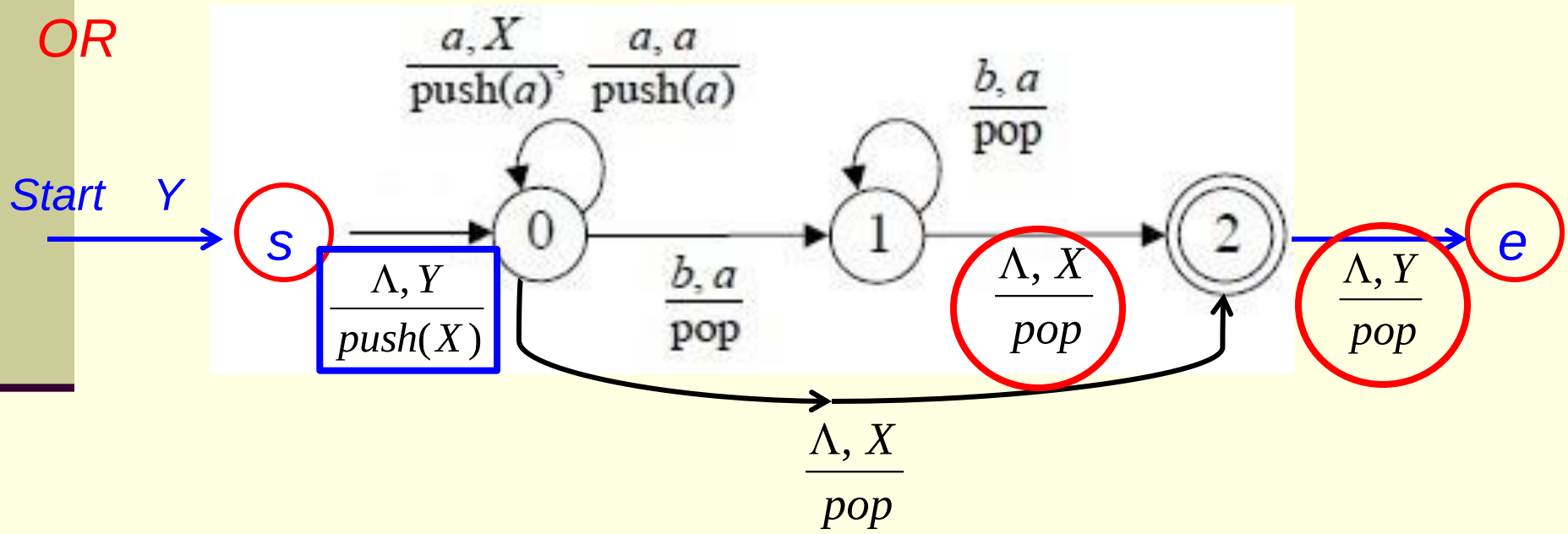
Example: given the following final-state PDA, convert it to a empty-stack PDA



7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example: given the following final-state PDA, convert it to a empty-stack PDA

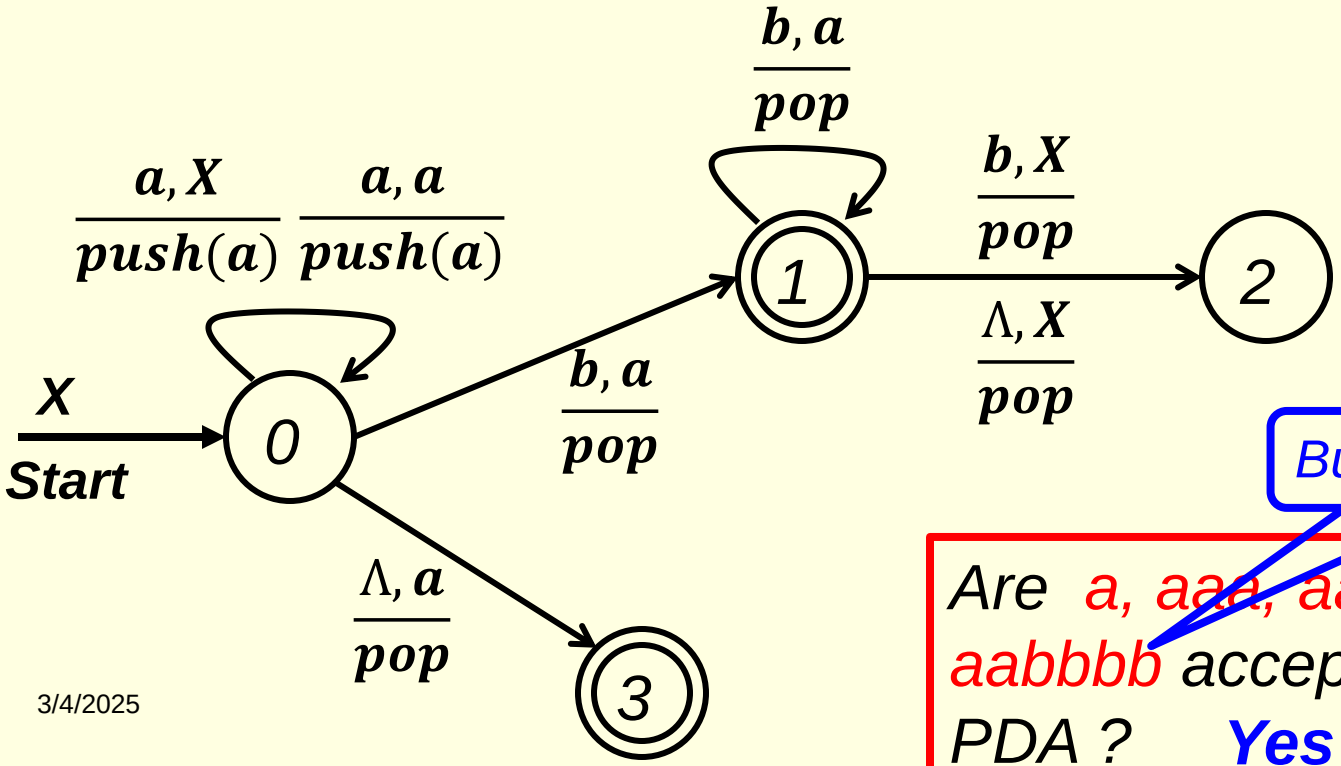
OR



Question: is the new stack symbol 'Y' really needed here ?

First, non-final states have nothing to do with 'Y' or 'e'.

Consider the following final state acceptance PDA that accepts the CFL L={ $a^n b^m \mid n, m \in N;\ n > m$ }

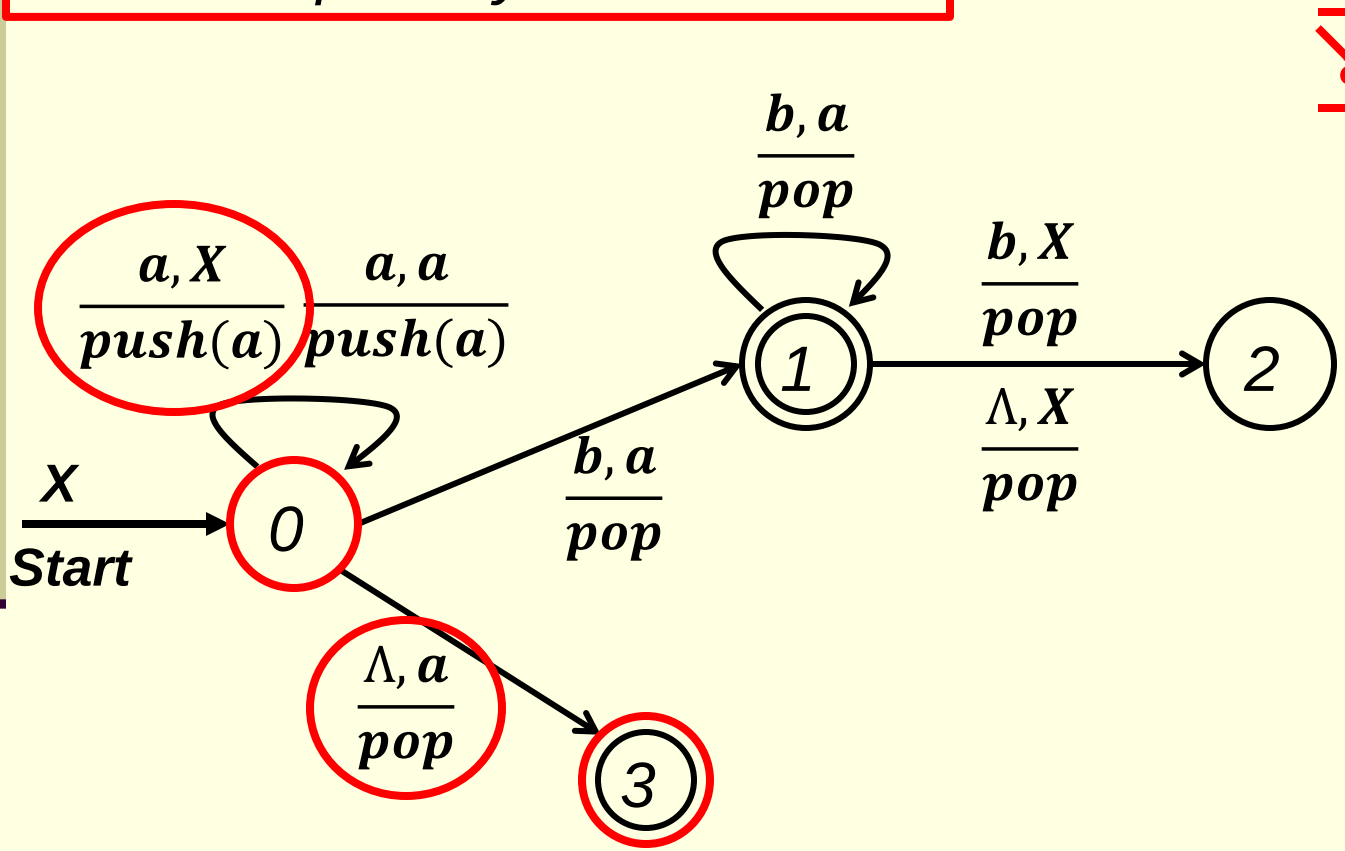


But not this one

Are a, aaa, aab, aaaabb, aabbbb accepted by this PDA ? ***Yes***

Consider the following final state acceptance PDA that accepts the CFL L={ $a^n b^m \mid n, m \in N;\ n > m$ }

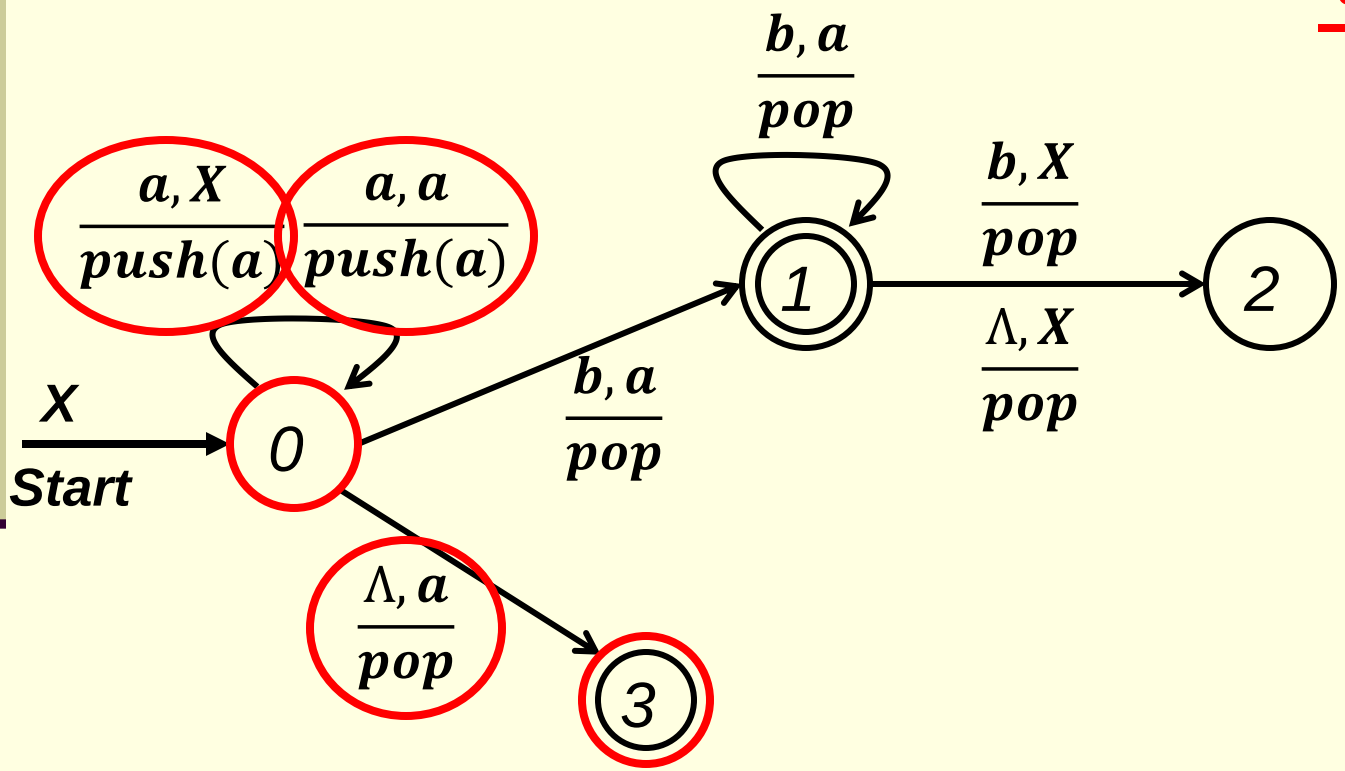
Is a accepted by this PDA ?

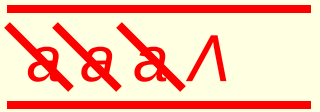


Yes

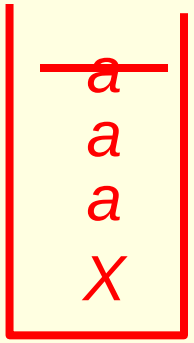
Consider the following final state acceptance PDA that accepts the CFL L={ $a^n b^m \mid n, m \in N;\ n > m$ }

Is aaa accepted by this PDA ?





Input queue

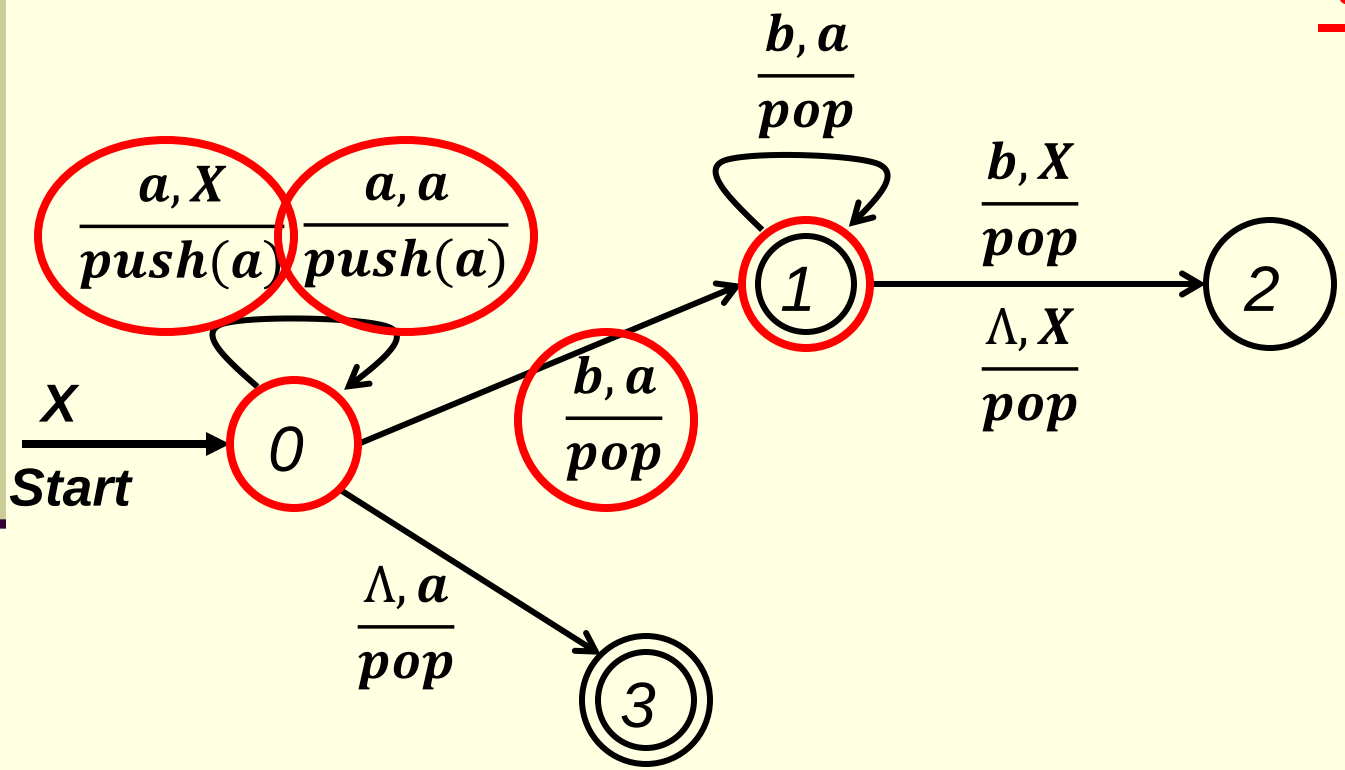


stack

Yes

Consider the following final state acceptance PDA that accepts the CFL L={ $a^n b^m \mid n, m \in N;\ n > m$ }

Is aab accepted by this PDA ?



a a b Λ

Input queue

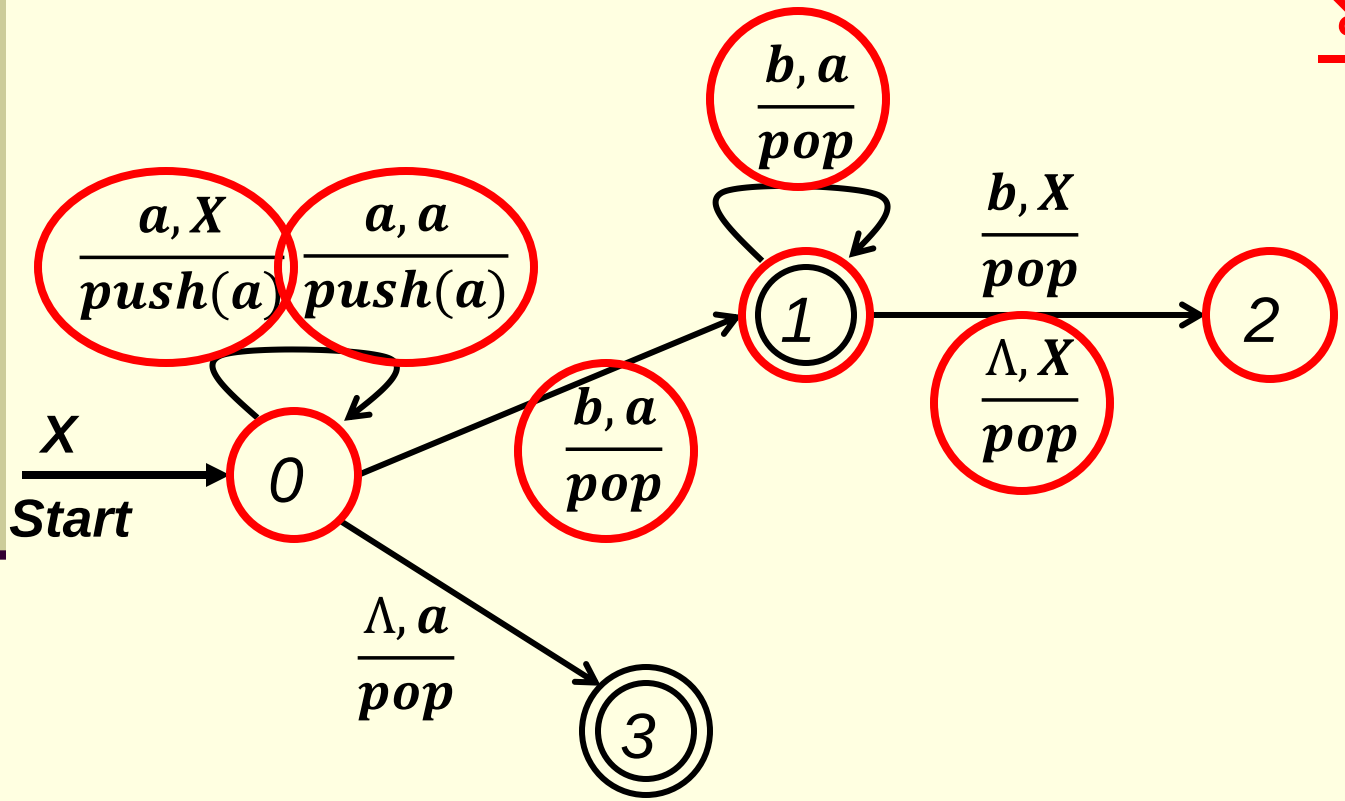
a
a
X

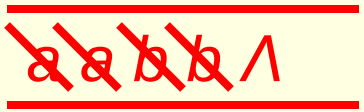
stack

Yes

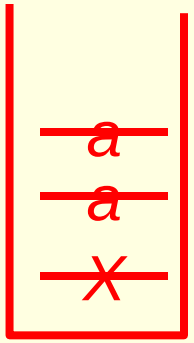
Consider the following final state acceptance PDA that accepts the CFL L={ $a^n b^m \mid n, m \in N;\ n > m$ }

Is aabb accepted by this PDA ?





Input queue

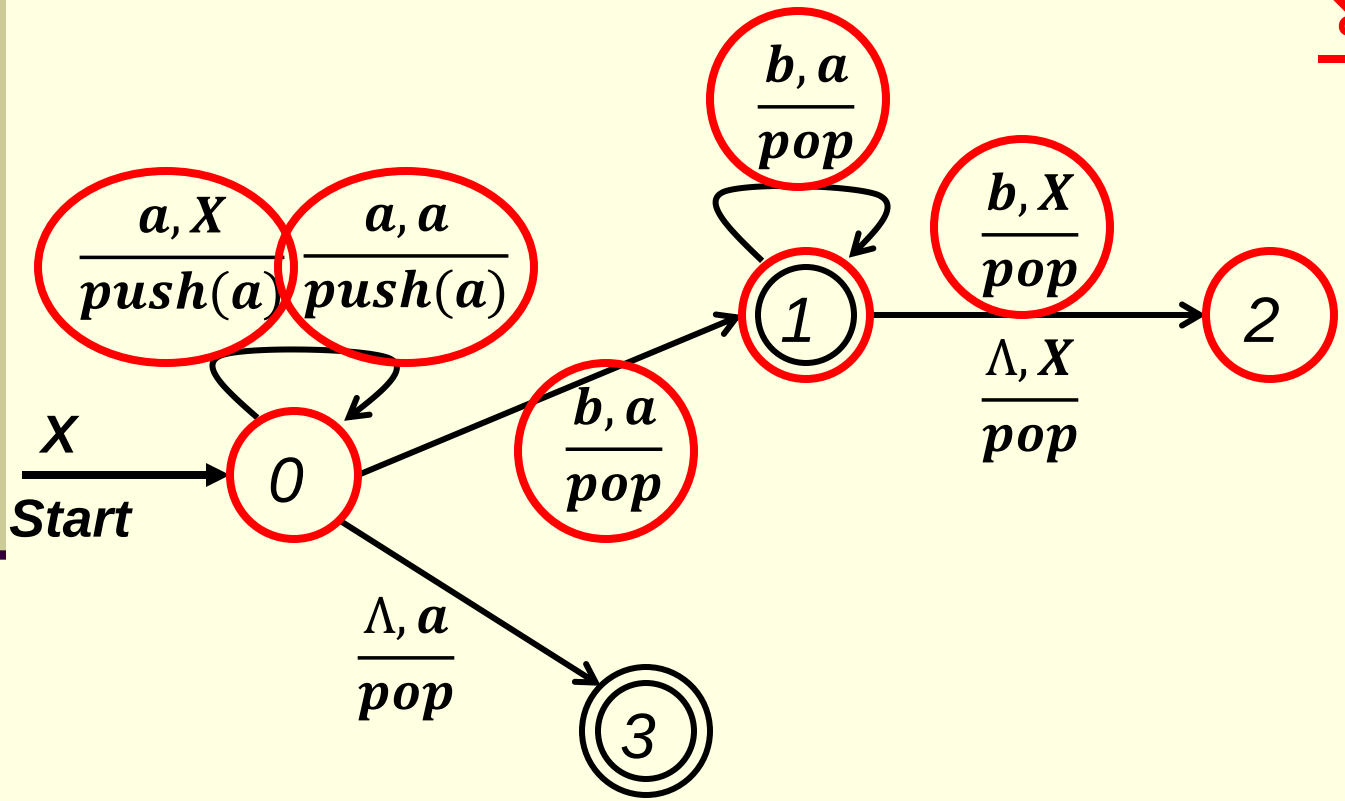


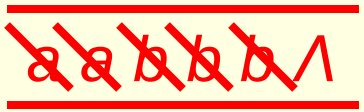
stack

Yes or No

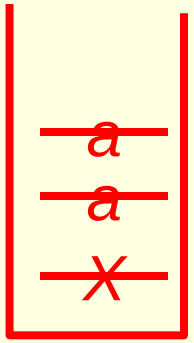
Consider the following final state acceptance PDA that accepts the CFL L={ $a^n b^m \mid n, m \in N;\ n > m$ }

Is aabbb accepted by this PDA ?





Input queue

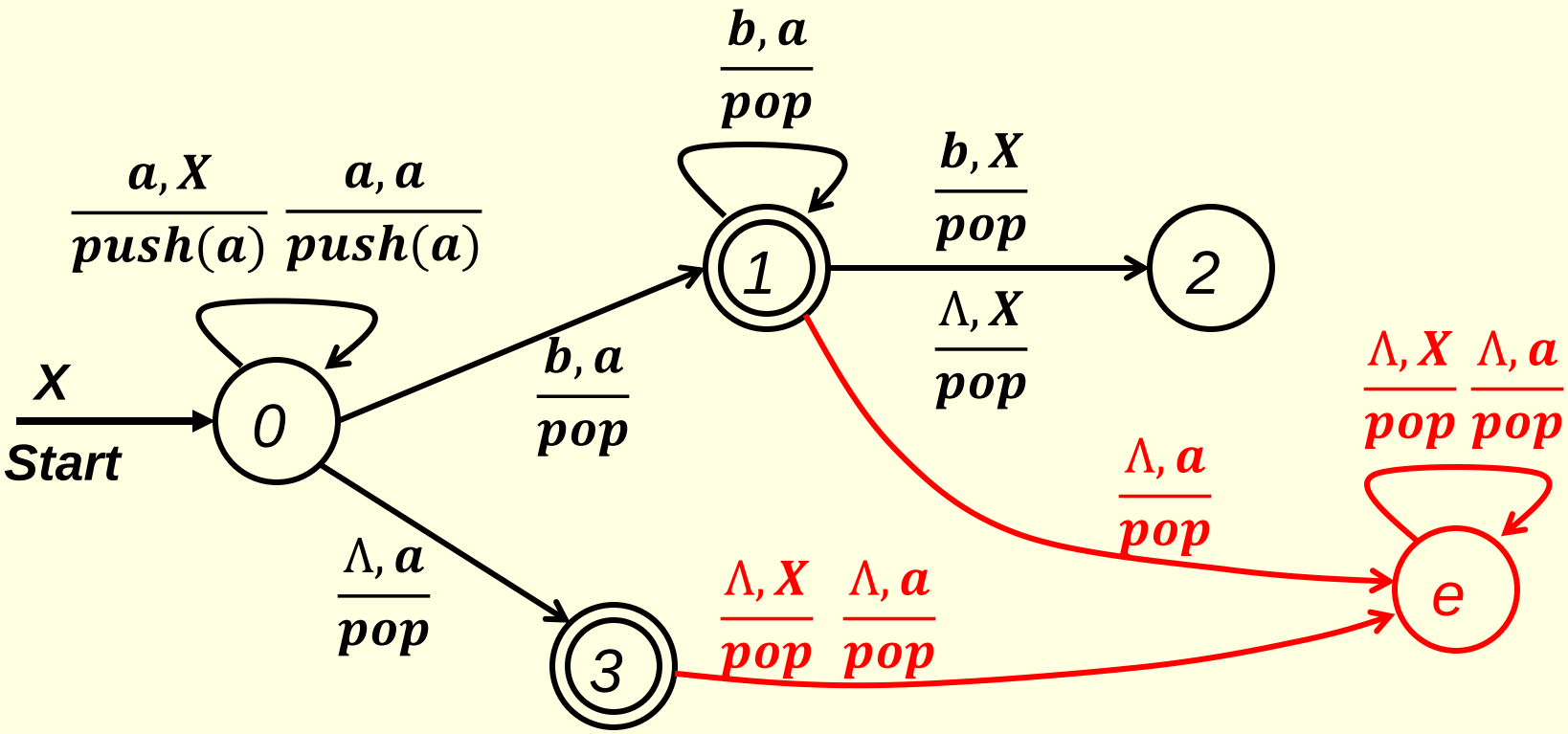


stack

No

If we add an empty state 'e' and construct new edges and instructions as follows, would this give us an empty-stack acceptance PDA for $L = \{ a^n b^m \mid n, m \in N;\ n > m \}$?

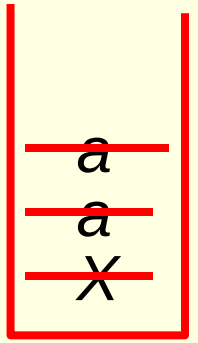
NO. Why?



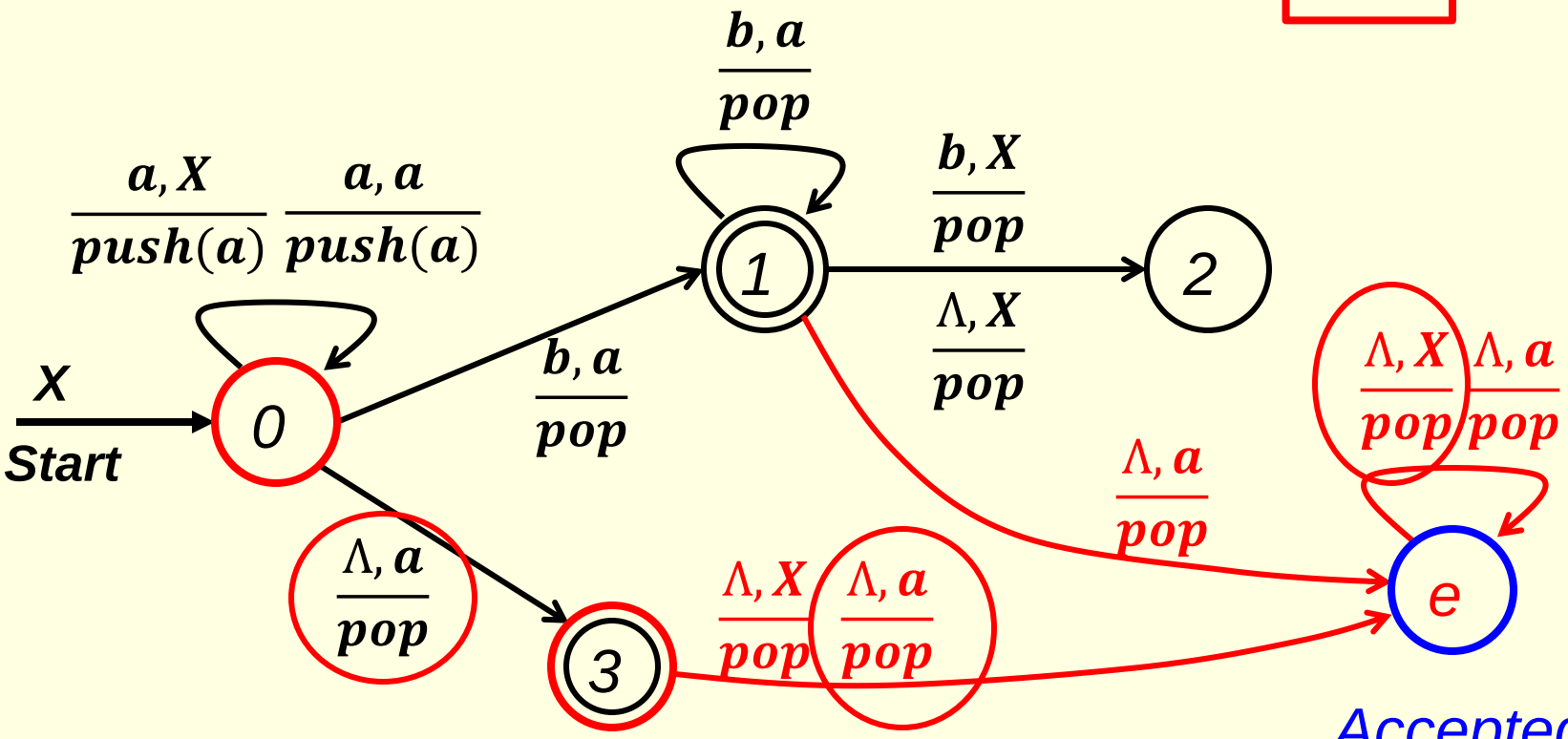
It accepts strings a^n, $n \geq 1$, by empty stack

a a Λ Λ Λ

Input queue



stack

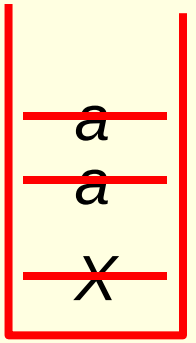


Accepted

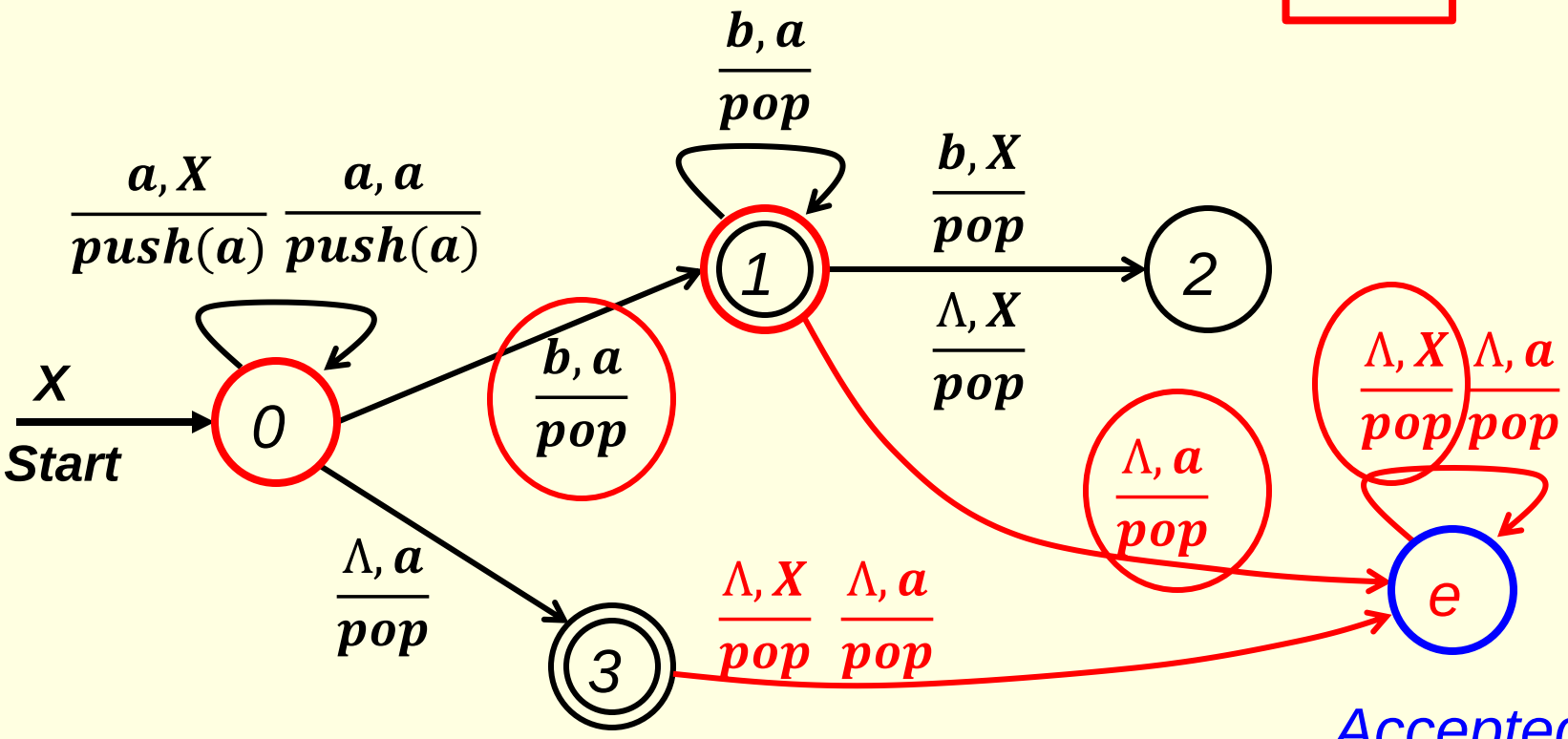
It accepts strings $a^n b^m$, $n \geq m$, by empty stack

a a b Λ Λ

Input queue



stack

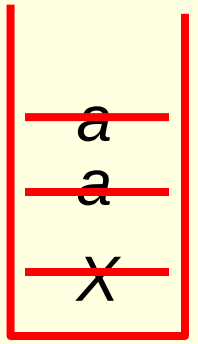


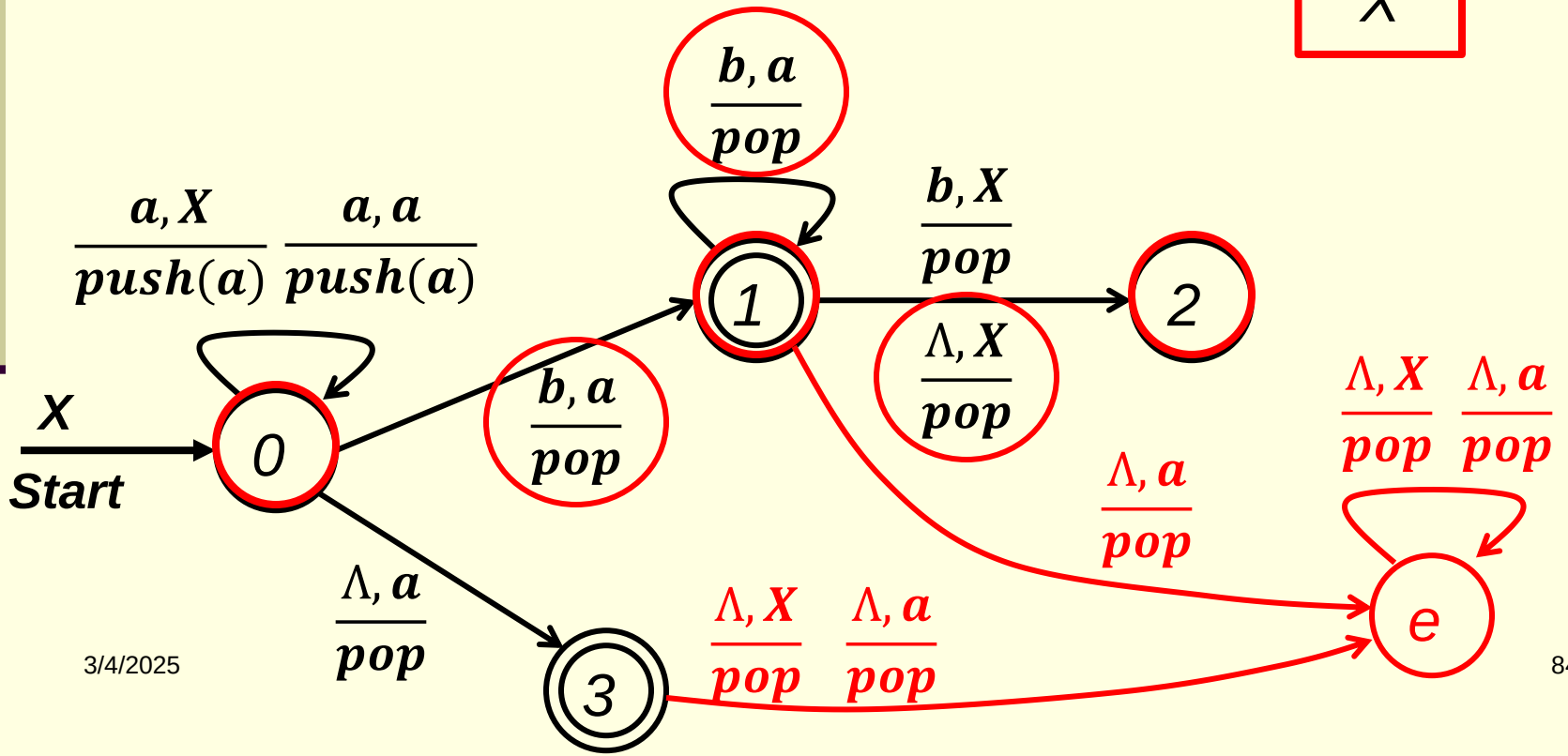
But it would accept strings $a^n b^n$, $n \geq 1$, by empty stack

a a b b Λ

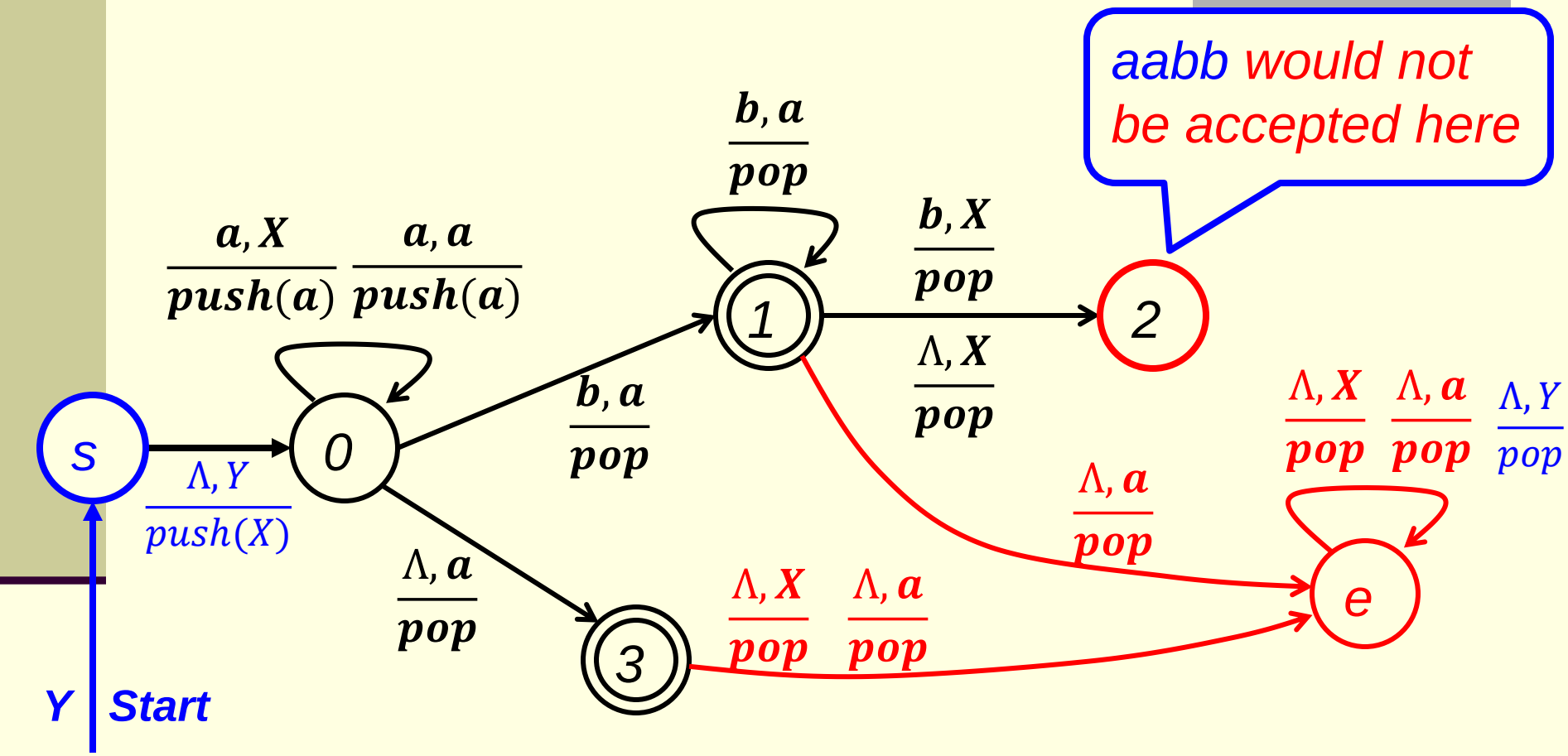
Input queue

accepted





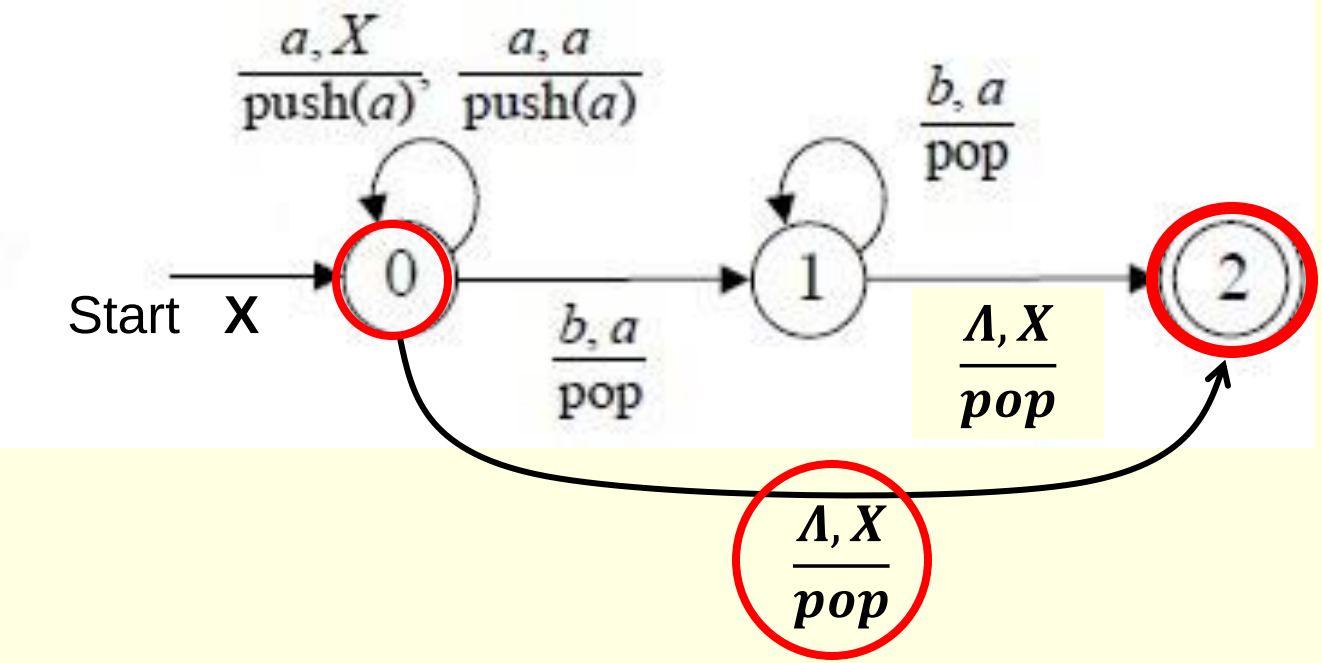
Using a new stack symbol 'Y' in the transformation process is necessary here.



So Y is really needed here

A final state acceptance PDA for the language

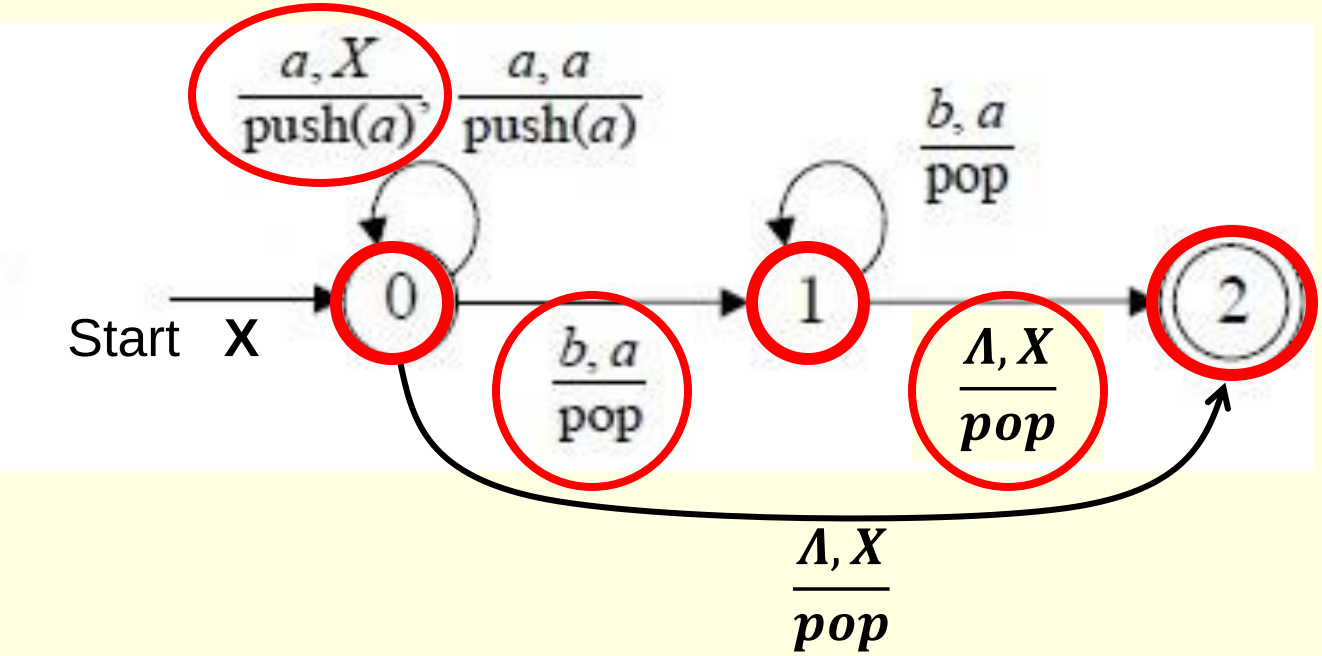
L = { $a^n b^n$ | $n \in N$ } ?



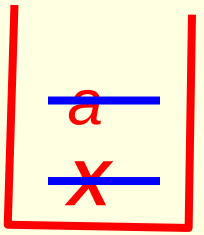
When Λ is accepted, the stack is empty

A final state acceptance PDA for the language

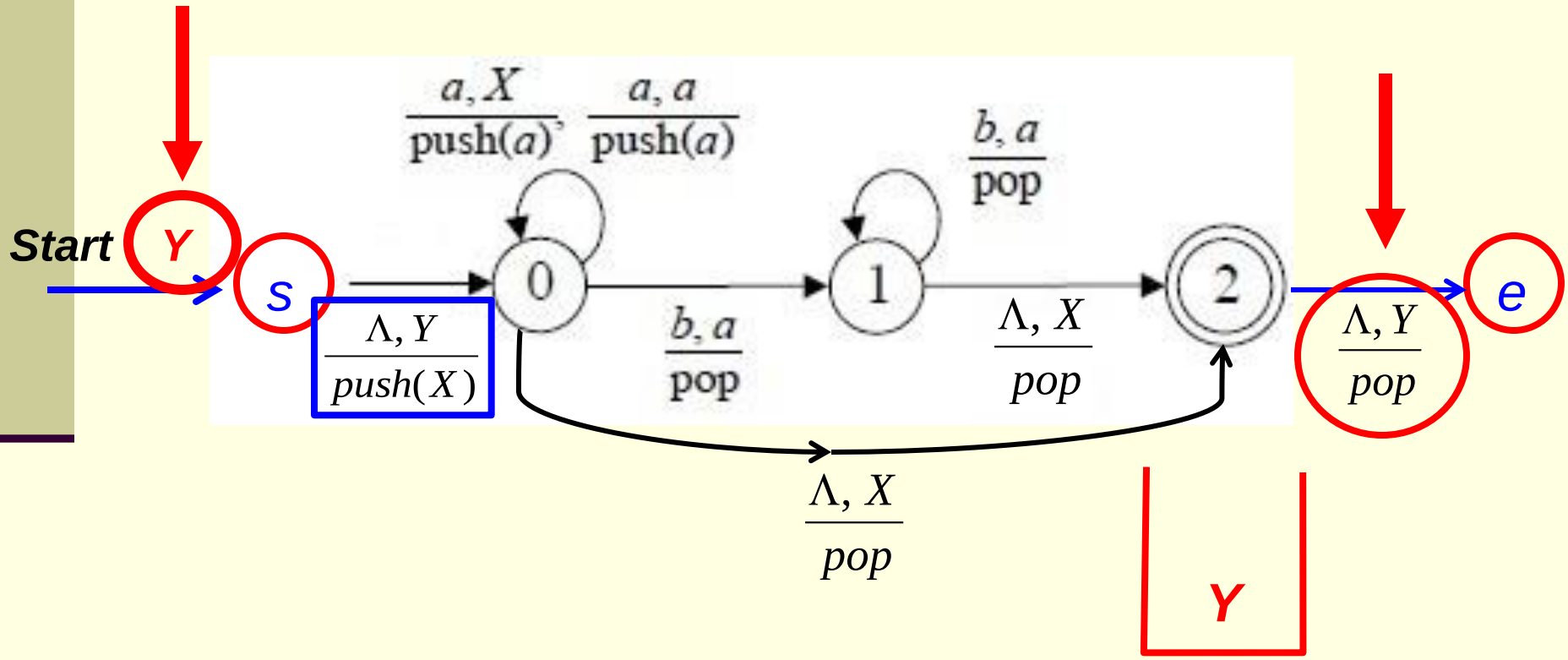
L = { $a^n b^n$ | $n \in N$ } ?



When aabΛ is accepted, the stack is empty



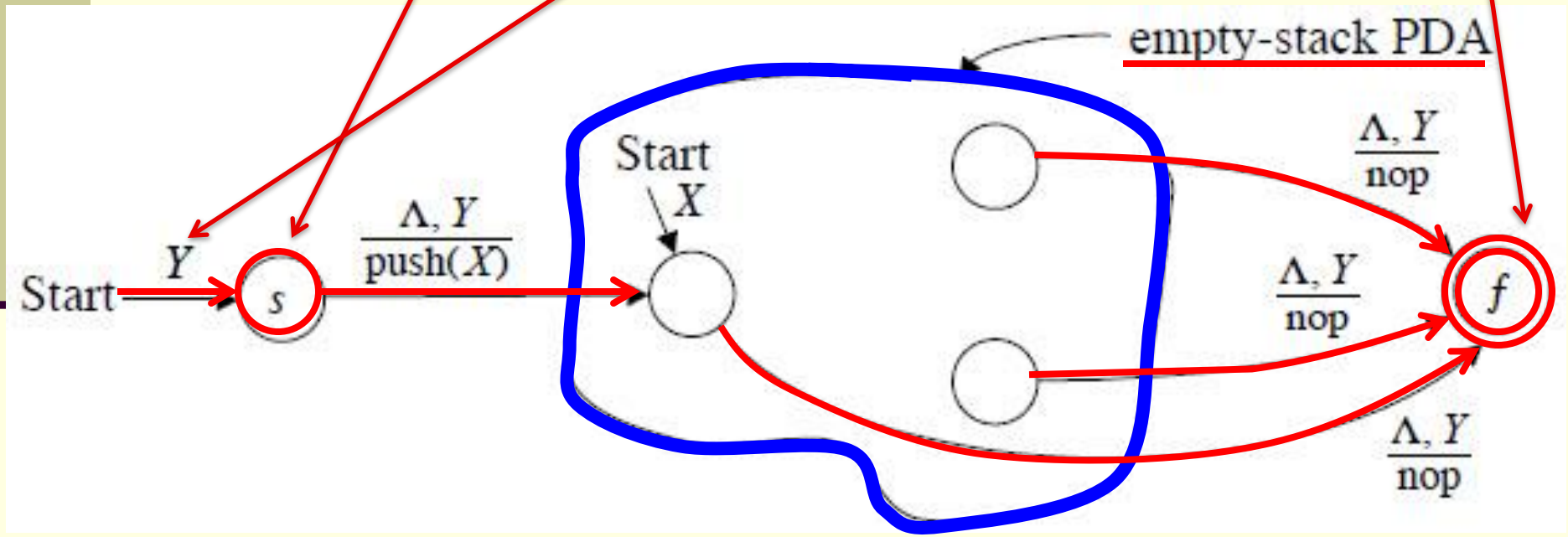
In this case, when we do the transformation, a new stack symbol 'Y' is also needed b/c otherwise we wouldn't be able to move from state 2 to state 'e'.



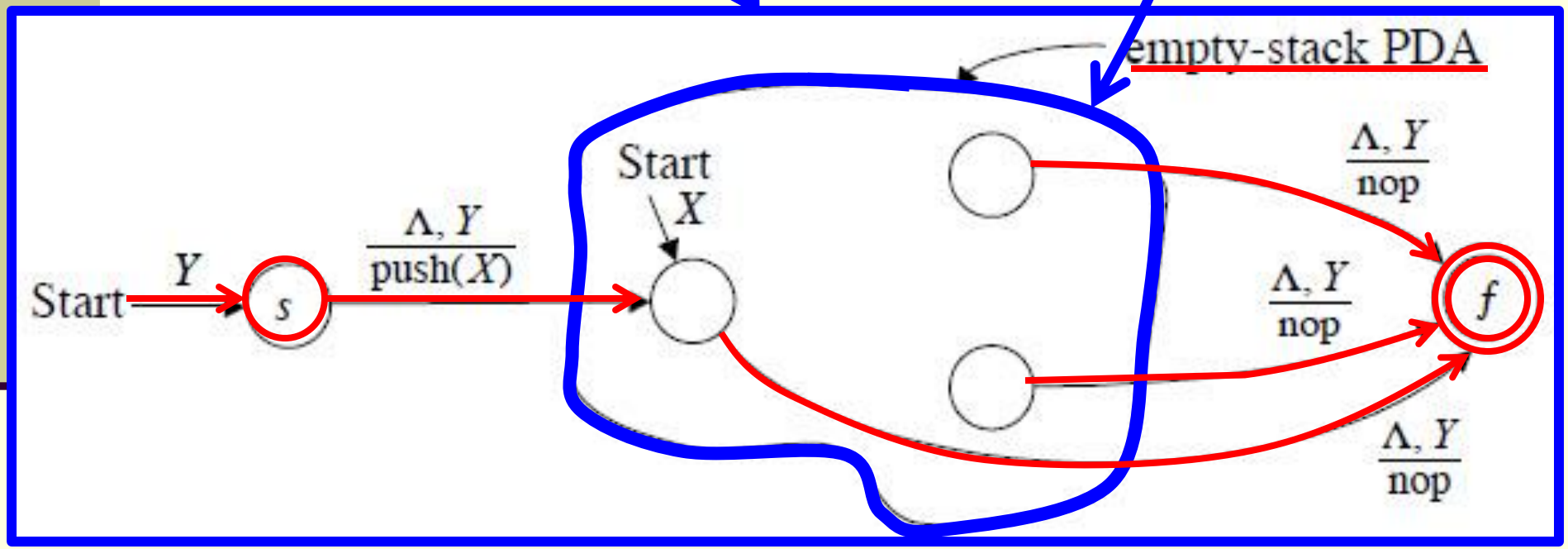
7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Transform empty-stack acceptance to final-state acceptance:

Introduce a new start state *s*, a new stack symbol *Y*, a new final state *f*, and construct edges and actions as shown in the diagram below.



The arrangement here ensures that if xxxx is accepted by the empty-stack PDA then ΛxxxxΛ is accepted by the final-state PDA



7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Theorem: The context-free (C-F) languages are exactly the languages accepted by PDAs.

Proof: We'll see two algorithms,

one to transform a C-F grammar to an empty-stack PDA,

& one to transform an empty-stack PDA to a C-F grammar. QED.

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

For each programming language, there is a PDA to recognize that programming language.

Why?
b/c each programming language is a context-free language

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Transform C-F grammar to empty-stack PDA:

- The idea is to use the stack of the PDA to simulate the (leftmost) derivation of a string in the grammar.
- The PDA has **ONE STATE**, but we allow multiple stack operations.
- The stack symbols are the terminals and nonterminals of the grammar.
- The designated starting stack symbol is the grammar start symbol.

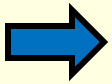
Transform C-F grammar to empty-stack PDA:

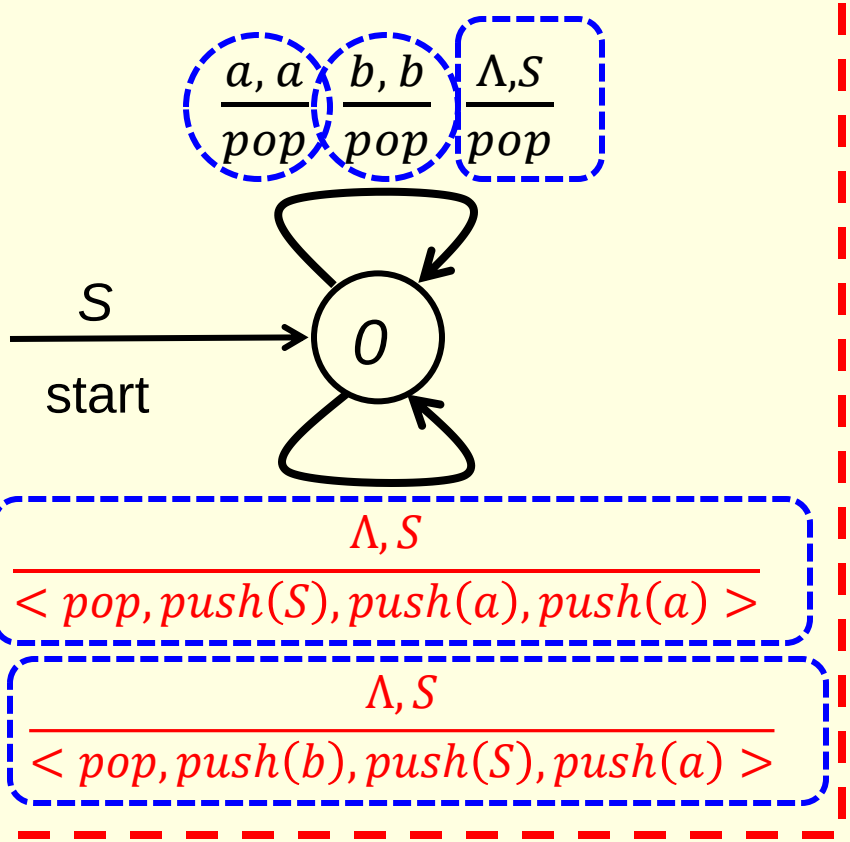
N = { S }

T = { a, b }

S : start symbol

P = { S → Λ ; S → aSb ; S → aaS }





Transform C-F grammar to empty-stack PDA:

N = { S }

T = { a, b }

S : start symbol

$$P = \left\{ \begin{array}{l} S \rightarrow \Lambda \\ S \rightarrow aSb \\ S \rightarrow aaS \end{array} \right\}$$

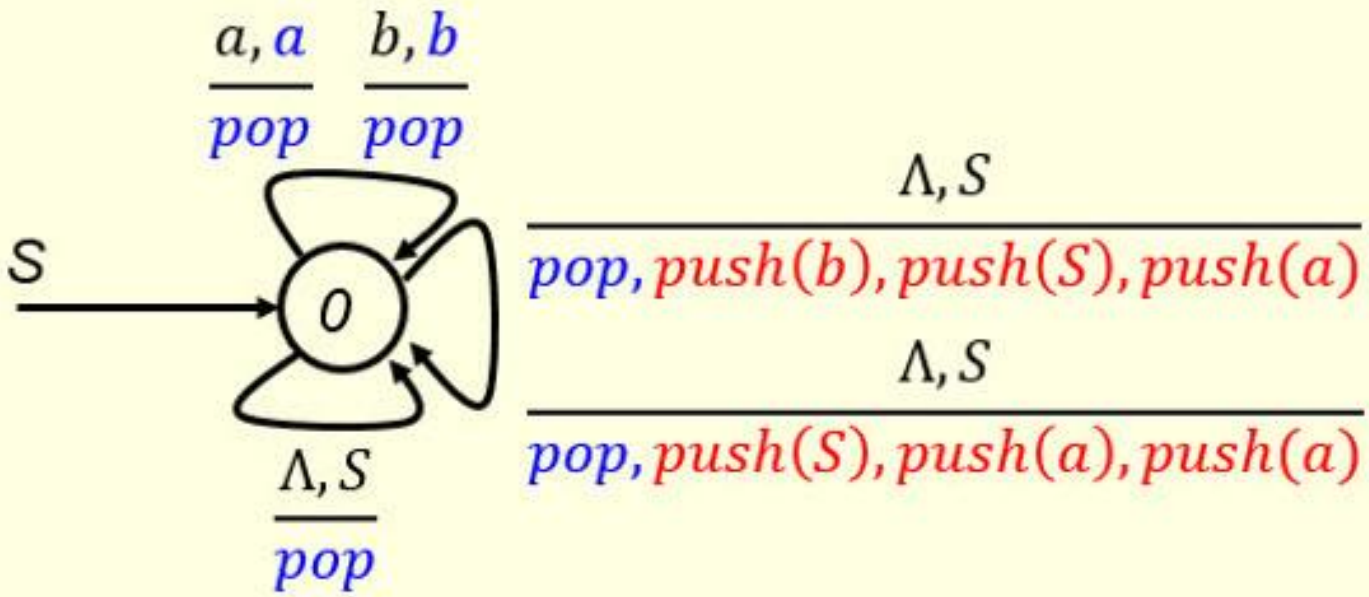
What is the language of this C-F grammar?

$$\{a^{m+2n}b^m \mid n, m \in N\}$$

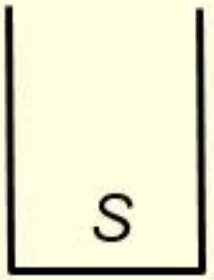
Λ, aa, aaaa, … , a^{2n}, ...
ab, aaab, aaaaab, …
aabb, aaaabb, aaaaabbb, …

Transform C-F grammar to empty-stack PDA:

Now, consider the following PDA:



Deterministic or non-deterministic?

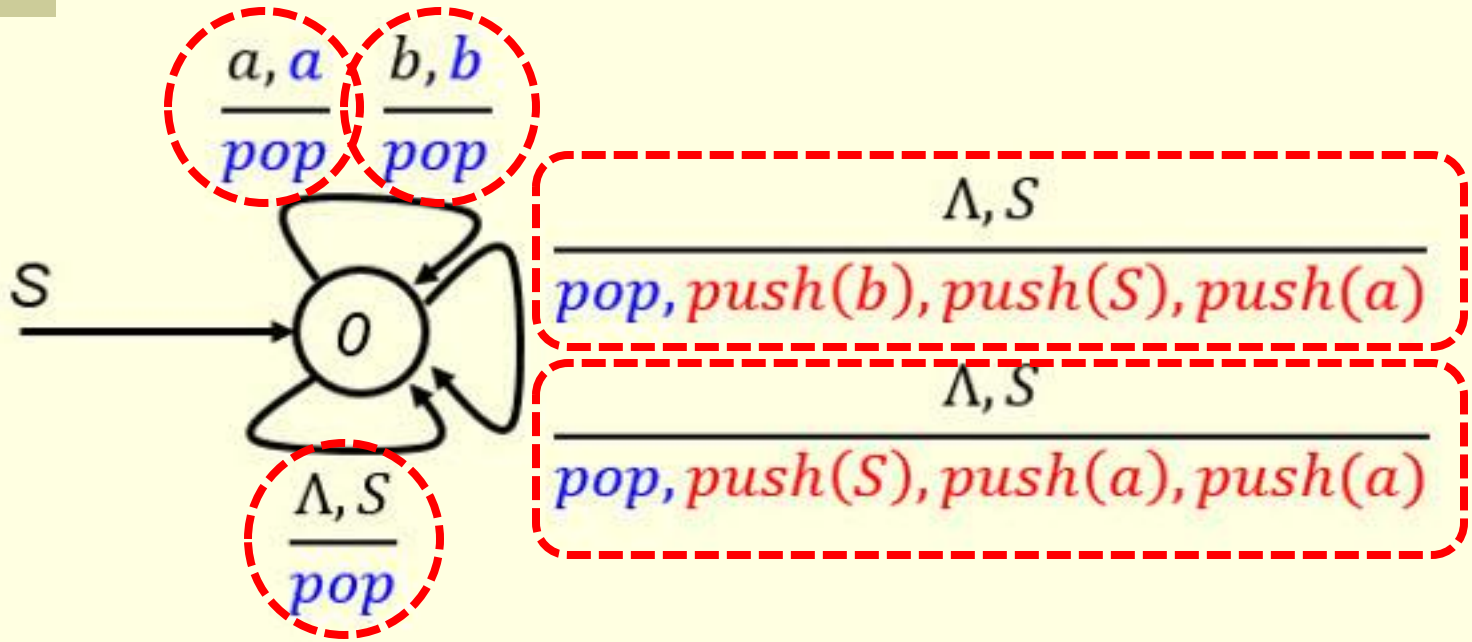


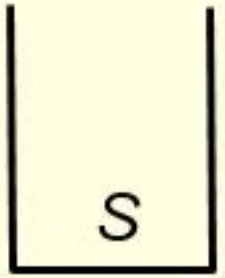
DFA vs NFA

For a DFA, you have only one chance to determine if a given string is accepted by the DFA because there will be only one path in the DFA for that string. If that path does not lead to a final state, then the only conclusion is that string is not accepted by the DFA.

For an NFA, there could be several different paths in the NFA for that string. If one path does not lead to a final state, you can try other available paths to see if one of them would work.

Transform C-F grammar to empty-stack PDA:





PDA instructions:

(0, a, a, pop, 0)

(0, b, b, pop, 0)

(0, Λ, S, pop, 0)

(0, Λ, S, <pop, push(b), push(S), push(a)>, 0)

(0, Λ, S, <pop, push(S), push(a), push(a)>, 0)

Transform C-F grammar to empty-stack PDA:

How do I know the PDA should be constructed the above way?

The work of a PDA reverses the work of a grammar in the sense that a grammar generates a string while a PDA executes a string

For instance: $S \Longrightarrow aSb \Longrightarrow aaaSb \Longrightarrow aaab$

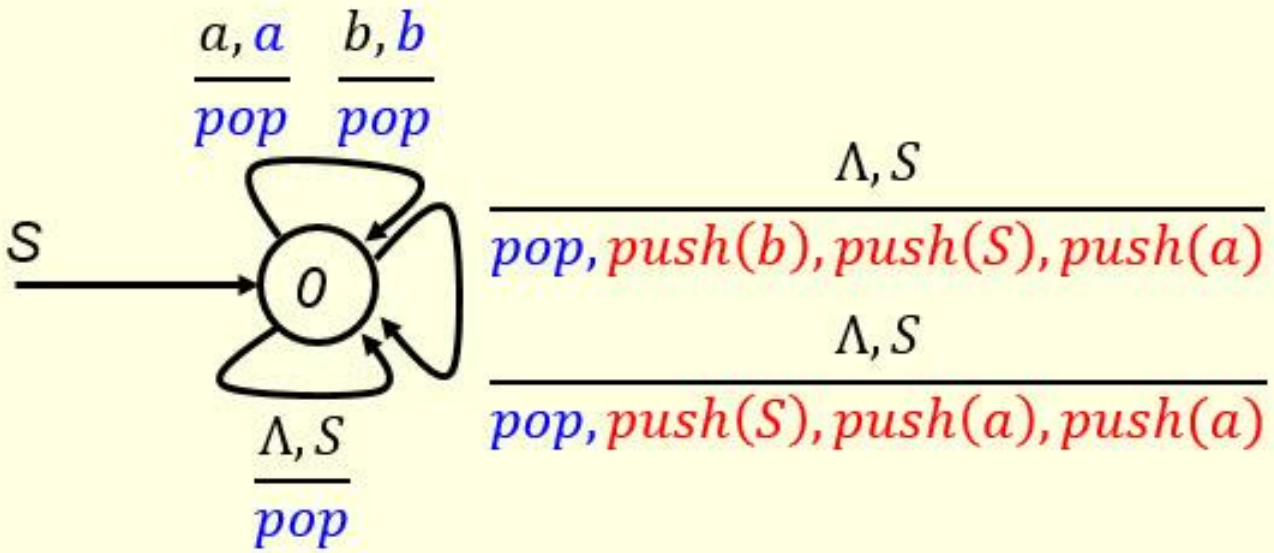
On the other hand: $aaab \rightarrow aab \rightarrow ab \rightarrow b \rightarrow \Lambda$

Key Idea:

when top symbol of the stack is a nonterminal and next input symbol is not a Λ

then put a Λ in front of the un-consumed portion of the input string

and the PDA should have an instruction for you to replace the top symbol of the stack with the right hand side of a grammar production.



$(0, \Lambda aaab, S)$

$(0, aaab, aSb)$

$(0, \Lambda aab, Sb)$

$(0, aab, aaSb)$

$(0, ab, aSb)$

$(0, \Lambda b, Sb)$

$(0, b, b)$

$(0, \Lambda, \Lambda)$

a
a
S
a
S
b
S

$(0, \Lambda, S, \langle pop, push(b), push(S), push(a) \rangle, 0)$

$(0, a, a, pop, 0)$

$(0, \Lambda, S, \langle pop, push(S), push(a), push(a) \rangle, 0)$

$(0, a, a, pop, 0)$

$(0, a, a, pop, 0)$

$(0, \Lambda, S, pop, 0)$

$(0, b, b, pop, 0)$

Accepted

$(0, \Lambda aaab, S)$

$(0, aaab, aSb)$

$(0, \Lambda aab, Sb)$

$(0, aab, aaSb)$

$(0, ab, aSb)$

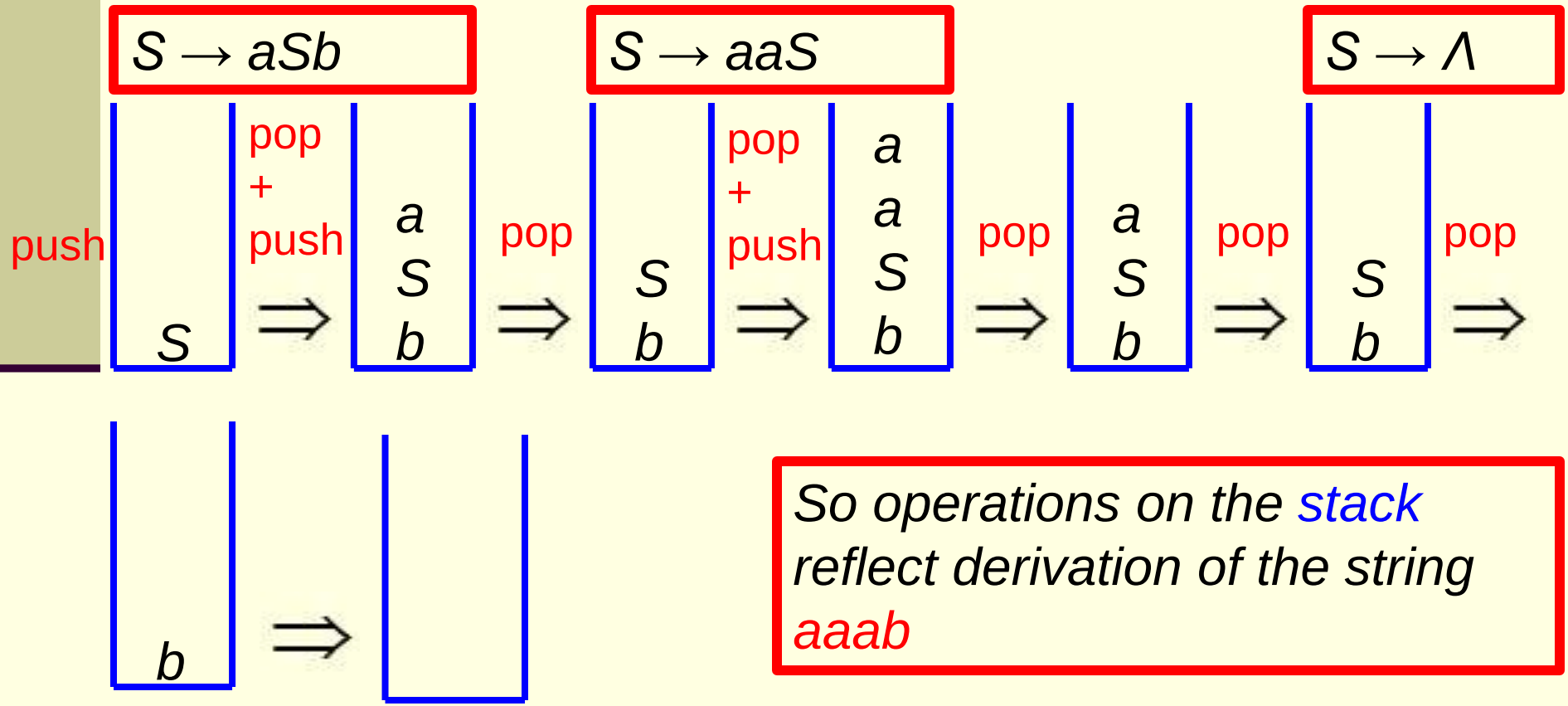
$(0, \Lambda b, Sb)$

$(0, b, b)$

$(0, \Lambda, \Lambda)$

(Left-most derivation of aaab)

$S \Rightarrow aSb \Rightarrow aaaSb \Rightarrow aaab$



So operations on the stack reflect derivation of the string aaab

Transform C-F grammar to empty-stack PDA:

So, when the PDA processes the string aaab, operations on the stack of the PDA reflect the derivation of the string aaab

Now the process of transforming a C-F grammar to an empty-stack PDA

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Transform C-F grammar to empty-stack PDA:

Basic idea:

– Use the stack of the PDA to simulate the (leftmost) derivation of a string in the grammar

- Push S (start symbol of CFG) on the stack
- From this point on, there are two moves the PDA can make:

1. If a non-terminal A is at the top of the stack, pop it and push the right-hand side of a production $A \rightarrow B_1B_2 \cdots B_n$ from G
2. If a terminal *a* is at the top of the stack, pop it and **match** it with whatever symbol is being read from the input string

Leftmost and Rightmost Derivations

Consider the grammar $G = (\{S, A, B, C\}, \{a, b, c\}, S, P)$ where

$P = \{ S \rightarrow ABC,\ A \rightarrow aA \mid \Lambda,\ B \rightarrow bB \mid \Lambda,\ C \rightarrow cC \mid \Lambda \}$

Leftmost derivation (always expand the leftmost variable first):

$S \Rightarrow ABC \Rightarrow aABC \Rightarrow aBC \Rightarrow abBC \Rightarrow abbBC \Rightarrow abbC$
$\Rightarrow abbcC \Rightarrow abbc$

Rightmost derivation (always expand the rightmost variable first):

$S \Rightarrow ABC \Rightarrow ABcC \Rightarrow ABc \Rightarrow AbBc \Rightarrow AbbBc \Rightarrow Abbc$
$\Rightarrow aAbbc \Rightarrow abbc$

Note:

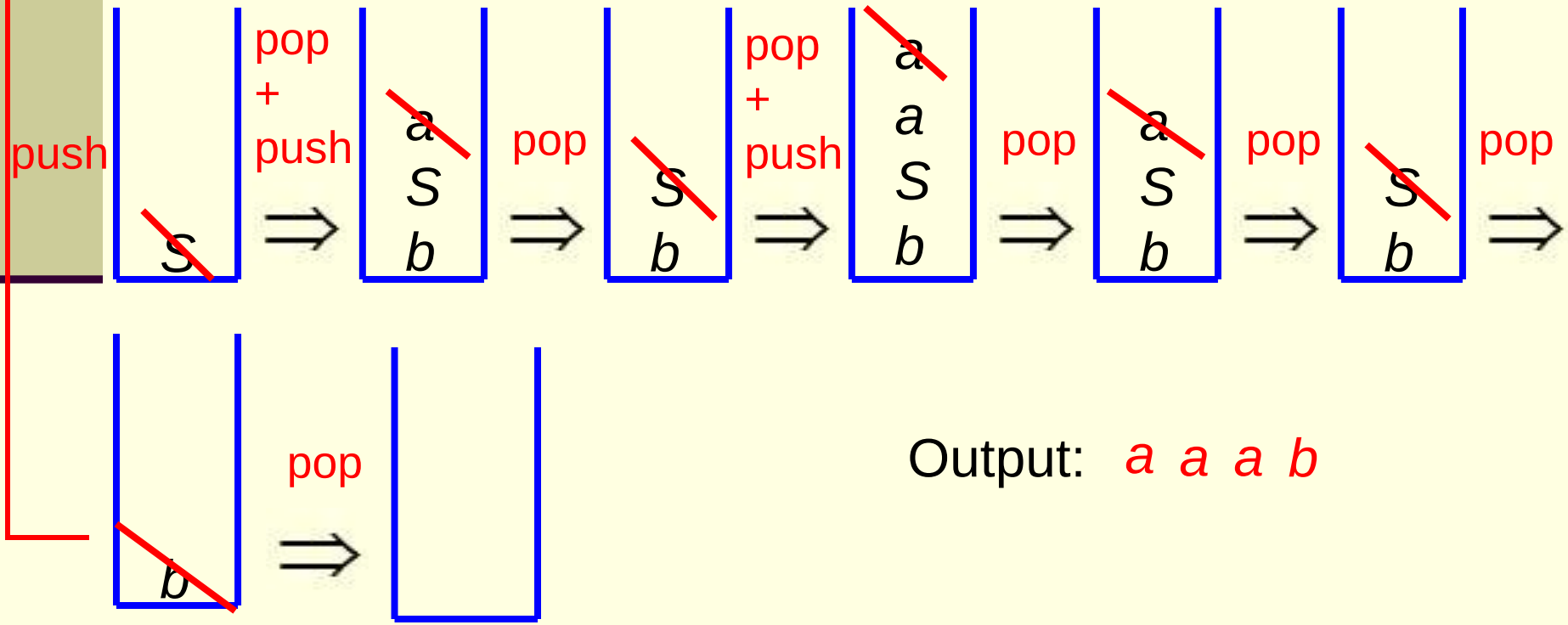
1. Different derivations result in different sentential forms, but
2. For a C-F grammar, it doesn't make difference in what order we expand the variables.

7. Context-Free Languages & Pushdown Automata- Pushdown Automata

How to simulate a leftmost derivation of *aaab* for the grammar

$S \to \Lambda \mid aSb \mid aaS$

$S \Rightarrow aSb \Rightarrow aaaSb \Rightarrow aaab$ $\Leftrightarrow$ $S \to aSb\,;\quad S \to aaS\,;\quad S \to \Lambda$



7. Context-Free Languages & Pushdown Automata- Pushdown Automata

Example. For the grammar: $S \to \Lambda \mid aSb \mid aaS$, do a leftmost derivation of *aaab* and show the IDs for executing it.

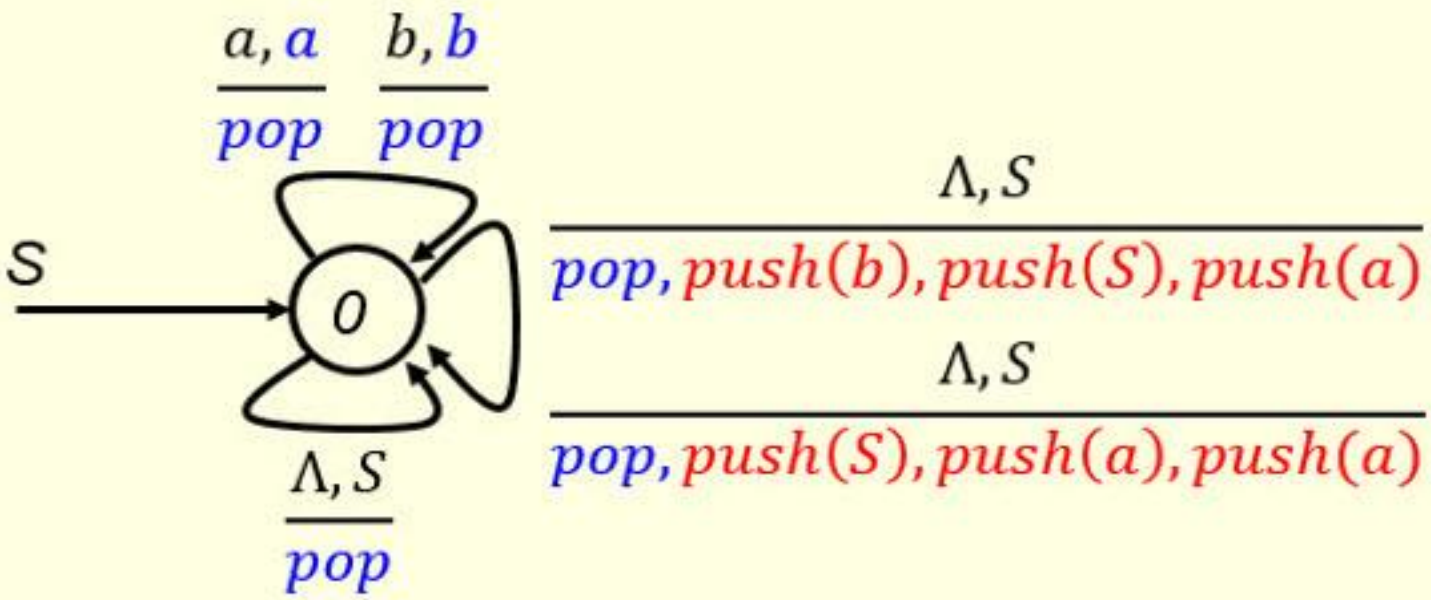
First, get PDA instructions for terminals and productions:

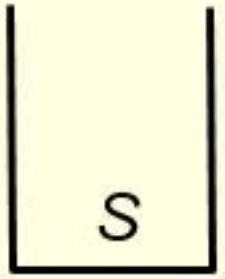
Terminals	Corresponding PDA instructions
a	$(0, a, a, pop, 0)$
b	$(0, b, b, pop, 0)$

Start with this one

Productions	Corresponding PDA instructions
$S \to aSb$	$(0, \Lambda, S, \langle pop, push(b), push(S), push(a)\rangle, 0)$
$S \to aaS$	$(0, \Lambda, S, \langle pop, push(S), push(a), push(a)\rangle, 0)$
$S \to \Lambda$	$(0, \Lambda, S, pop, 0)$

This is the empty-stack PDA:





Why?

First note that:

A leftmost derivation of *aaab:*

ID's for aaab:

$(0, \Lambda aaab, S)$

$(0, aaab, aSb)$

$(0, \Lambda aab, Sb)$

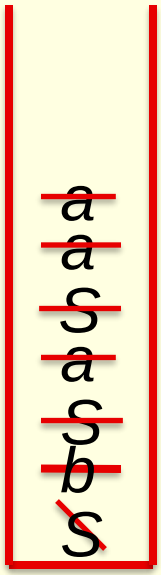
$(0, aab, aaSb)$

$(0, ab, aSb)$

$(0, \Lambda b, Sb)$

$(0, b, b)$

$(0, \Lambda, \Lambda)$



PDA instructions:

$(0, \Lambda, S, \langle pop, push(b), push(S), push(a) \rangle, 0)$

$(0, a, a, pop, 0)$

$(0, \Lambda, S, \langle pop, push(S), push(a), push(a) \rangle, 0)$

$(0, a, a, pop, 0)$

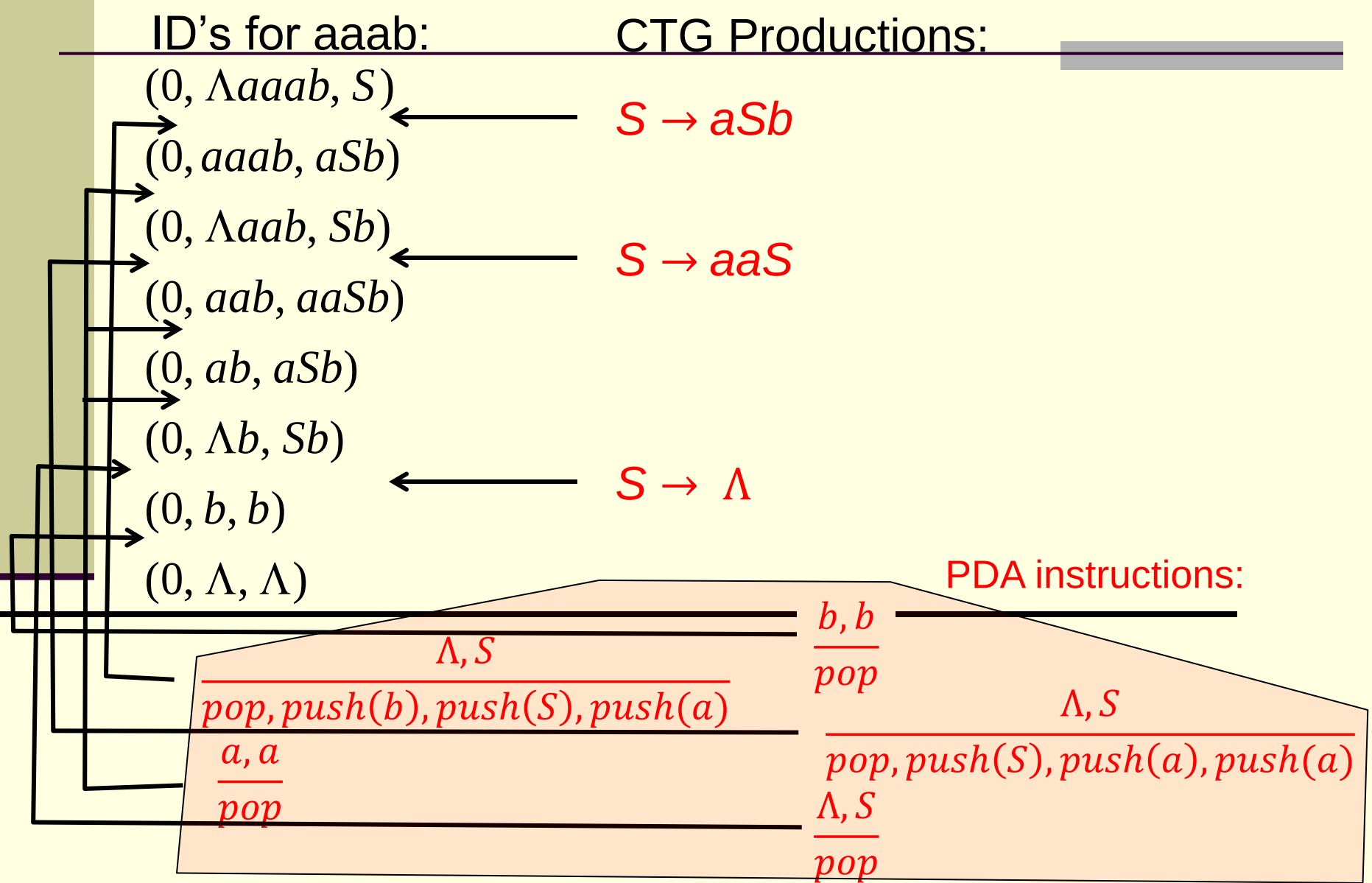
$(0, a, a, pop, 0)$

$(0, \Lambda, S, pop, 0)$

$(0, b, b, pop, 0)$

Accepted

Here is why:


ID's for aaab:
CTG Productions:
$(0, \Lambda aaab, S)$
$S \rightarrow aSb$
$(0, aaab, aSb)$
$(0, \Lambda aab, Sb)$
$S \rightarrow aaS$
$(0, aab, aaSb)$
$(0, ab, aSb)$
$(0, \Lambda b, Sb)$
$S \rightarrow \Lambda$
$(0, b, b)$
$(0, \Lambda, \Lambda)$
PDA instructions:
$\frac{b, b}{pop}$
$\frac{\Lambda, S}{pop, push(b), push(S), push(a)}$
$\frac{\Lambda, S}{pop, push(S), push(a), push(a)}$
$\frac{a, a}{pop}$
$\frac{\Lambda, S}{pop}$

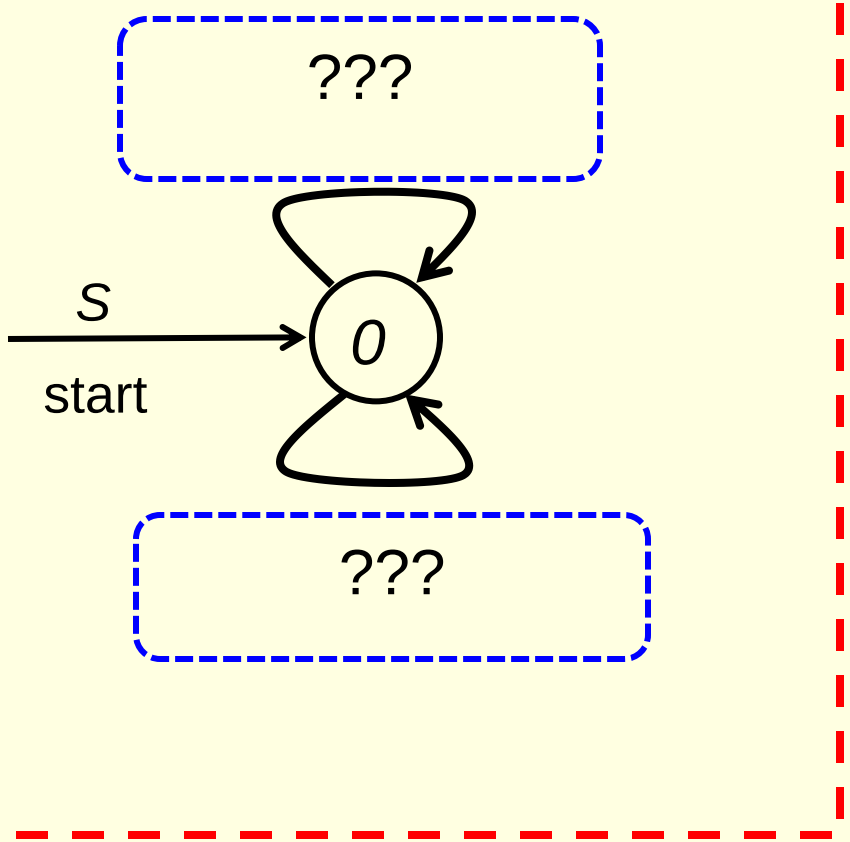
Example: transform the following C-F grammar to an empty-stack PDA

N = { S }

T = { a, b, c, d }

S : start symbol

$$P = \left\{ \begin{array}{rl} S & \rightarrow \ d|ABC \\ A & \rightarrow aA|\Lambda \\ B & \rightarrow bB|\Lambda \\ C & \rightarrow cC|\Lambda \end{array} \right\}$$



$$P = \left\{ \begin{array}{l} S \rightarrow d|ABC \\ A \rightarrow aA|\Lambda \\ B \rightarrow bB|\Lambda \\ C \rightarrow cC|\Lambda \end{array} \right\}$$

d

~~*A d*~~

Input queue

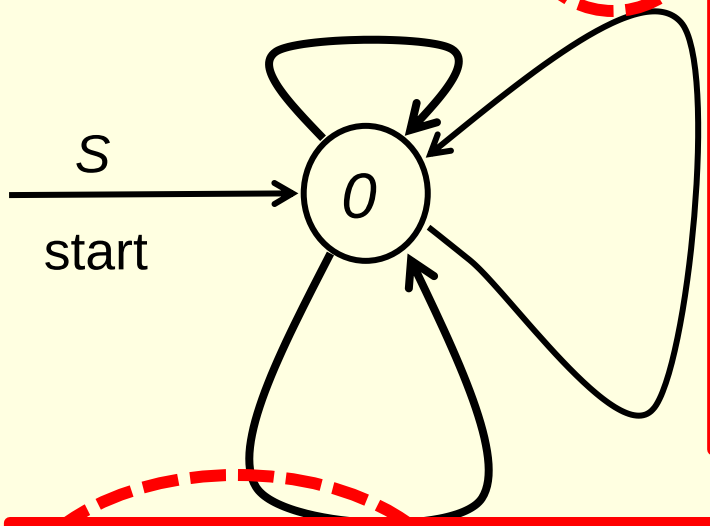
$\frac{a,a}{pop}, \frac{b,b}{pop}, \frac{c,c}{pop}, \frac{d,d}{pop}$

$$\frac{\Lambda, S}{< pop, push(C), push(B), push(A) >}$$

$$\frac{\Lambda, A}{< pop, push(A), push(a) >}$$

$$\frac{\Lambda, B}{< pop, push(B), push(b) >}$$

$$\frac{\Lambda, C}{< pop, push(C), push(c) >}$$



$\frac{\Lambda, S}{< pop, push(d) >}, \frac{\Lambda, A}{pop}, \frac{\Lambda, B}{pop}, \frac{\Lambda, C}{pop}$

~~*d*~~

~~**S**~~

$$P = \left\{ \begin{array}{l} S \rightarrow d|ABC \\ A \rightarrow aA|\Lambda \\ B \rightarrow bB|\Lambda \\ C \rightarrow cC|\Lambda \end{array} \right\}$$

a

Λ a

Input queue

~~*Λ Λ a Λ Λ Λ*~~

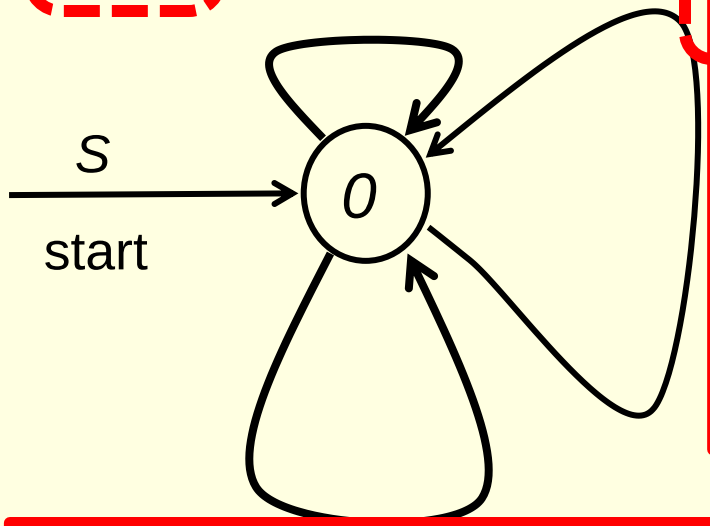
$\frac{a,a}{pop}, \frac{b,b}{pop}, \frac{c,c}{pop}, \frac{d,d}{pop}$

$\frac{\Lambda, S}{< pop, push(C), push(B), push(A) >}$

$\frac{\Lambda, A}{< pop, push(A), push(a) >}$

$\frac{\Lambda, B}{< pop, push(B), push(b) >}$

$\frac{\Lambda, C}{< pop, push(C), push(c) >}$



$\frac{\Lambda, S}{< pop, push(d) >}, \frac{\Lambda, A}{pop}, \frac{\Lambda, B}{pop}, \frac{\Lambda, C}{pop}$

~~*a*~~

A

~~*A*~~

B

C

~~**S**~~

Example: transform the following C-F grammar to an empty-stack PDA

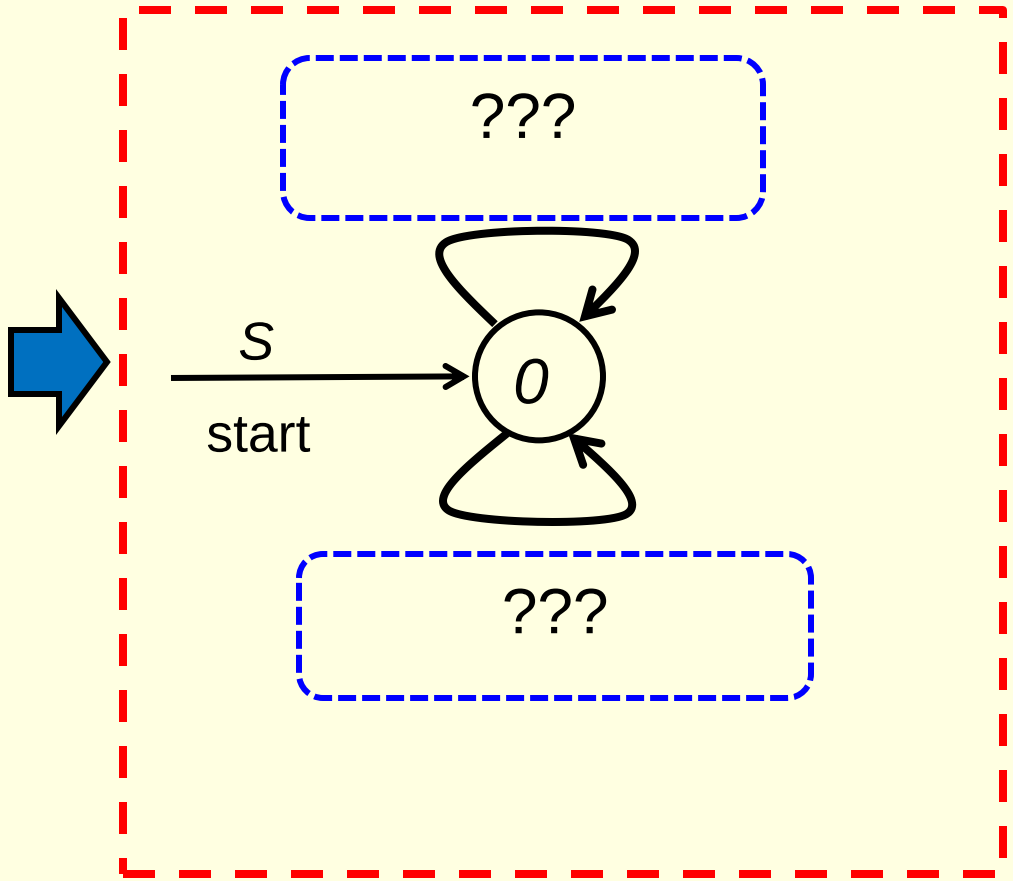
N = { S }

T = { a, b, c, d, e }

S : start symbol

P =

$$\left\{ \begin{array}{rcl} S & \rightarrow & d \mid e \mid ABC \\ A & \rightarrow & aA \mid \Lambda \\ B & \rightarrow & bB \mid \Lambda \\ C & \rightarrow & cC \mid \Lambda \end{array} \right\}$$



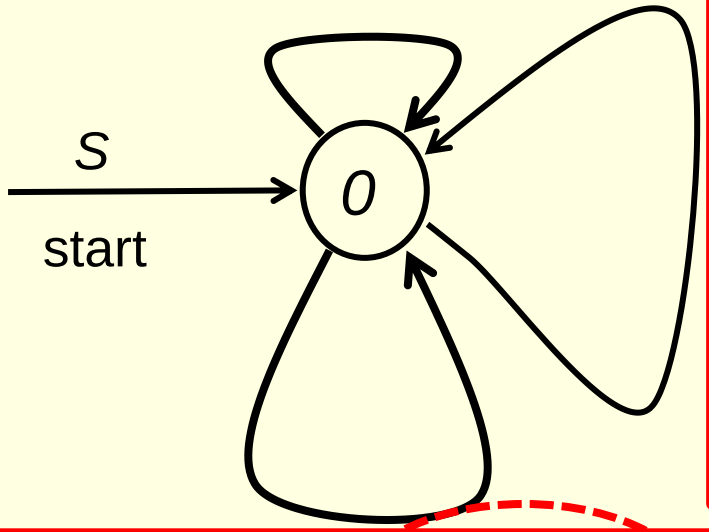
$$P = \left\{ \begin{array}{rcl} S & \rightarrow & d|e|ABC \\ A & \rightarrow & aA|\Lambda \\ B & \rightarrow & bB|\Lambda \\ C & \rightarrow & cC|\Lambda \end{array} \right\}$$

a a b e

Input queue

Do this at home

$$\frac{a,a}{pop}, \frac{b,b}{pop}, \frac{c,c}{pop}, \frac{d,d}{pop}, \frac{e,e}{pop}$$



$$\frac{\Lambda, S}{< pop, push(C), push(B), push(A) >}$$

$$\frac{\Lambda, A}{< pop, push(A), push(a) >}$$

$$\frac{\Lambda, B}{< pop, push(B), push(b) >}$$

$$\frac{\Lambda, C}{< pop, push(C), push(c) >}$$

$$\frac{\Lambda, S}{< pop, push(d) >}, \frac{\Lambda, S}{< pop, push(e) >}, \frac{\Lambda, A}{pop}, \frac{\Lambda, B}{pop}, \frac{\Lambda, C}{pop}$$

S

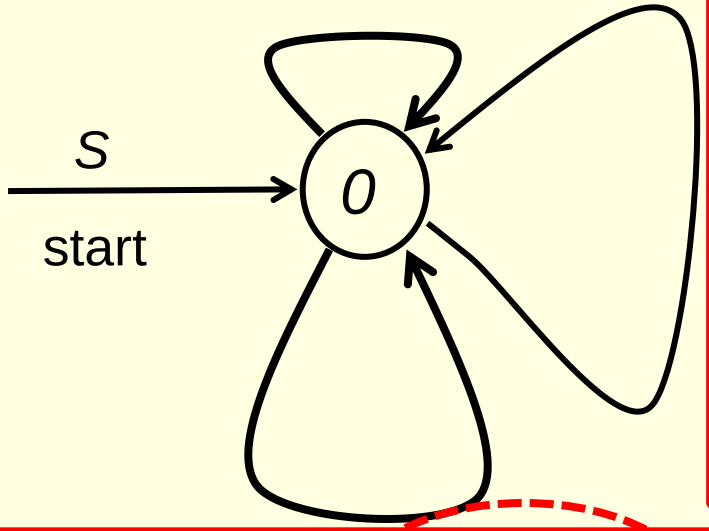
$$P = \left\{ \begin{array}{rcl} S & \rightarrow & d|e|ABC \\ A & \rightarrow & aA|\Lambda \\ B & \rightarrow & bB|\Lambda \\ C & \rightarrow & cC|\Lambda \end{array} \right\}$$

Input queue

a a b e

rejected

$$\frac{a,a}{pop}, \frac{b,b}{pop}, \frac{c,c}{pop}, \frac{d,d}{pop}, \frac{e,e}{pop}$$



$$\frac{\Lambda, S}{< pop, push(C), push(B), push(A) >}$$

$$\frac{\Lambda, A}{< pop, push(A), push(a) >}$$

$$\frac{\Lambda, B}{< pop, push(B), push(b) >}$$

$$\frac{\Lambda, C}{< pop, push(C), push(c) >}$$

$$\frac{\Lambda, S}{< pop, push(d) >}, \frac{\Lambda, S}{< pop, push(e) >}, \frac{\Lambda, A}{pop}, \frac{\Lambda, B}{pop}, \frac{\Lambda, C}{pop}$$

S

End of Context-Free Languages and Pushdown Automata I