CS375:
Logic and Theory of Computing

Fuhua (Frank) Cheng

Department of Computer Science

University of Kentucky

Table of Contents:

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- Pushdown Automata

Transform an empty-stack PDA to a C-F grammar

such that

language accepted by the PDA is the same as

language generated by the C-F grammar

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

We know how to transform a C-F grammar to an empty-stack PDA

Idea:

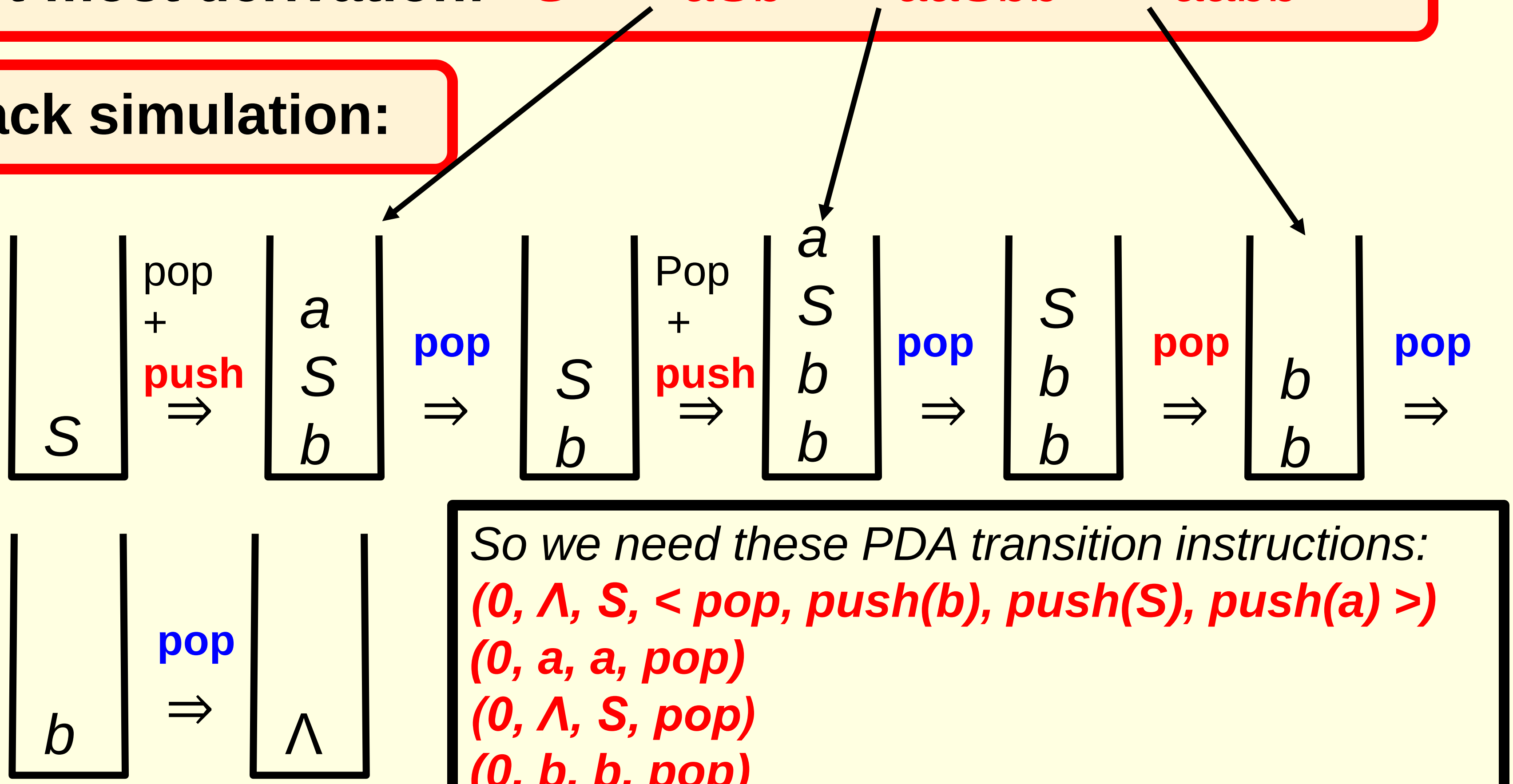
use stack to simulate the (left-most) derivation of a string

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

Example: Given S → aSb | Λ consider aabb

Left-most derivation: S ⇒ aSb ⇒ aaSbb ⇒ aabb

Stack simulation:



S

pop + **push** ⇒

a
S
b

pop ⇒

S
b

Pop + **push** ⇒

a
S
b
b

pop ⇒

S
b
b

pop ⇒

b
b

pop ⇒

b

pop ⇒

Λ

So we need these PDA transition instructions:

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

7. Con ag

Automata- ıdown Automata

Why should the stack operation be a ‘pop’?

b/c in this case ‘a’ will be output (accepted)

To transform an empty-stack PDA to a C-F grammar, we need to know the relationship between accepting a string and generating a string

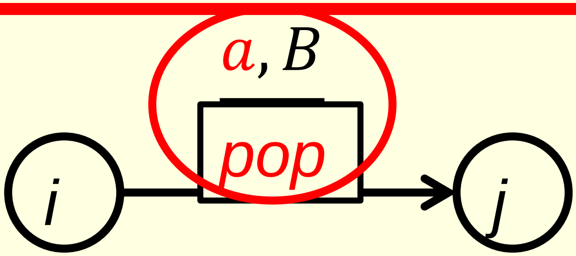
- ☐ Accepting a symbol ‘a’ means there is an instruction “$\frac{a,?}{pop}$” to execute when the input symbol ‘a’ is read
- ☐ Generating a symbol ‘a’ means a production of the form “B → a(w_1)” can be executed in the (leftmost) derivation process.

Relationship between **accepting** and **generating**:

How would a PDA accept a symbol ‘a’ in an input string?

$w = \cdots a \cdots$

Must have



(State j could be the same as state i, and B can not be Λ)

Why B can not be Λ ?

If B is Λ, it means the stack is empty, and a string has already been accepted, so no need to process anything any more.

Why the stack operation must be a ‘pop’?

b/c in this case ‘a’ will be output (accepted).

Relationship between **accepting** and **generating**:

How would a CFG generate a string with a symbol ‘a’ in it?

$w = \cdots a \cdots$

Must have

$B \rightarrow w_0\, a\, w_1$

(B is a non-terminal, w1 is a string of terminals and/or non-terminals, and w1 could be Λ)

Why w1 could be Λ ?

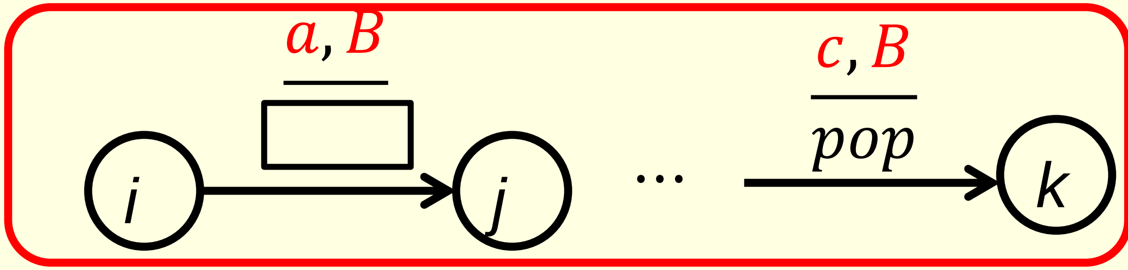
Because a could be the last symbol of w

Relationship between accepting and generating:

How would an empty-stack PDA accept a string ‘a⋯c’ ?

Must have

$w = \boxed{a \cdots c}$



(State k could be the same as state i, and B can not be Λ)

Why B can not be Λ ?

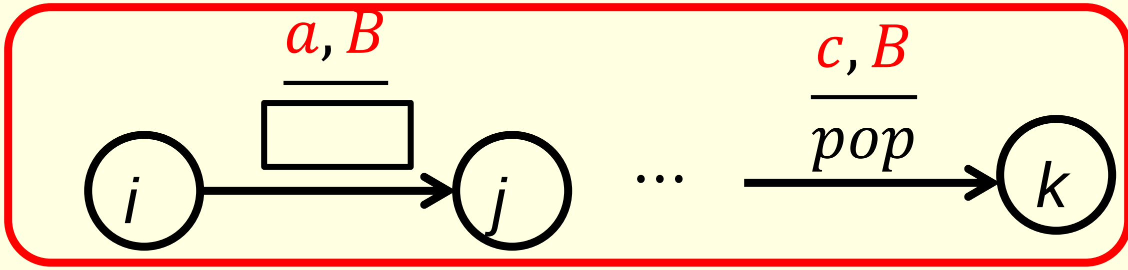
Why the ‘B’ in “c,B/pop” is the same as the ‘B’ in “a,B/▭ ” ?

Relationship between accepting and generating:

How would an empty-stack PDA accept a string ‘a⋯c’ ?

Must have

$$w = \boxed{a \cdots c}$$



(State k could be the same as state i, and B can not be Λ)

Why the ‘B’ in “c,B/pop” is the same as the ‘B’ in “a,B/▭ ” ?

For w=a…c to be accepted by the PDA, B must be the start symbol of the stack, it must also be the last symbol to remove from the stack.

Transform an empty-stack PDA to a C-F grammar

Skip slides 12-14

Basic idea:

Define non-terminals Bij for the CFG that can generate all strings w that:

- upon reading w on the input tape the PDA will take you from state i to state j in the PDA and have a "net result" of popping B off the stack

- In essence, B is "eventually" replaced by w
- It may take many moves to get there.

Transform an empty-stack PDA to a C-F grammar

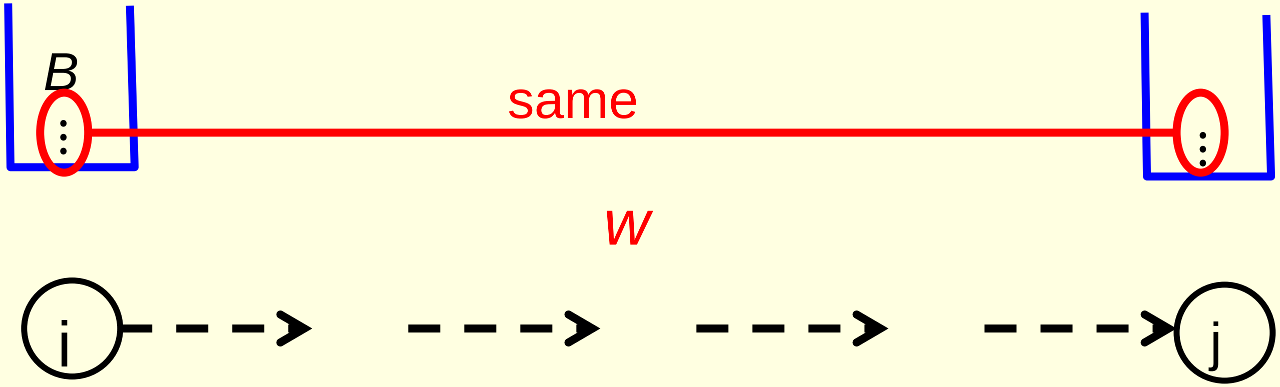
Three questions have to be answered:

1. *When would only a terminal be involved on the right hand side of a production?*

2. *When would both a terminal and a non-terminal be involved on the right hand side of a production?*

3. *When would the start symbol be involved in a production?*

Transform an empty-stack PDA to a C-F grammar



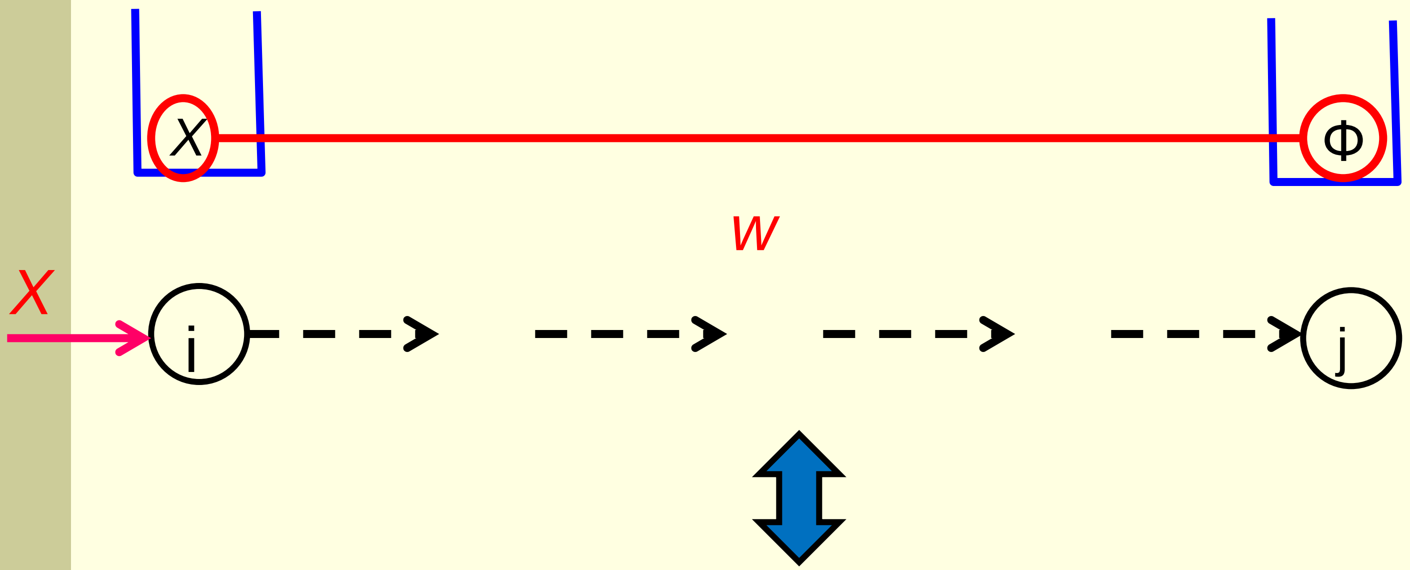
(B is removed)

$$B_{ij} \Rightarrow \boxed{} \Rightarrow \boxed{} \Rightarrow \cdots \Rightarrow \boxed{w}$$

(leftmost derivation)

Transform an empty-stack PDA to a C-F grammar

Case 1:



(X is removed)

(stack is empty)

(w is accepted)

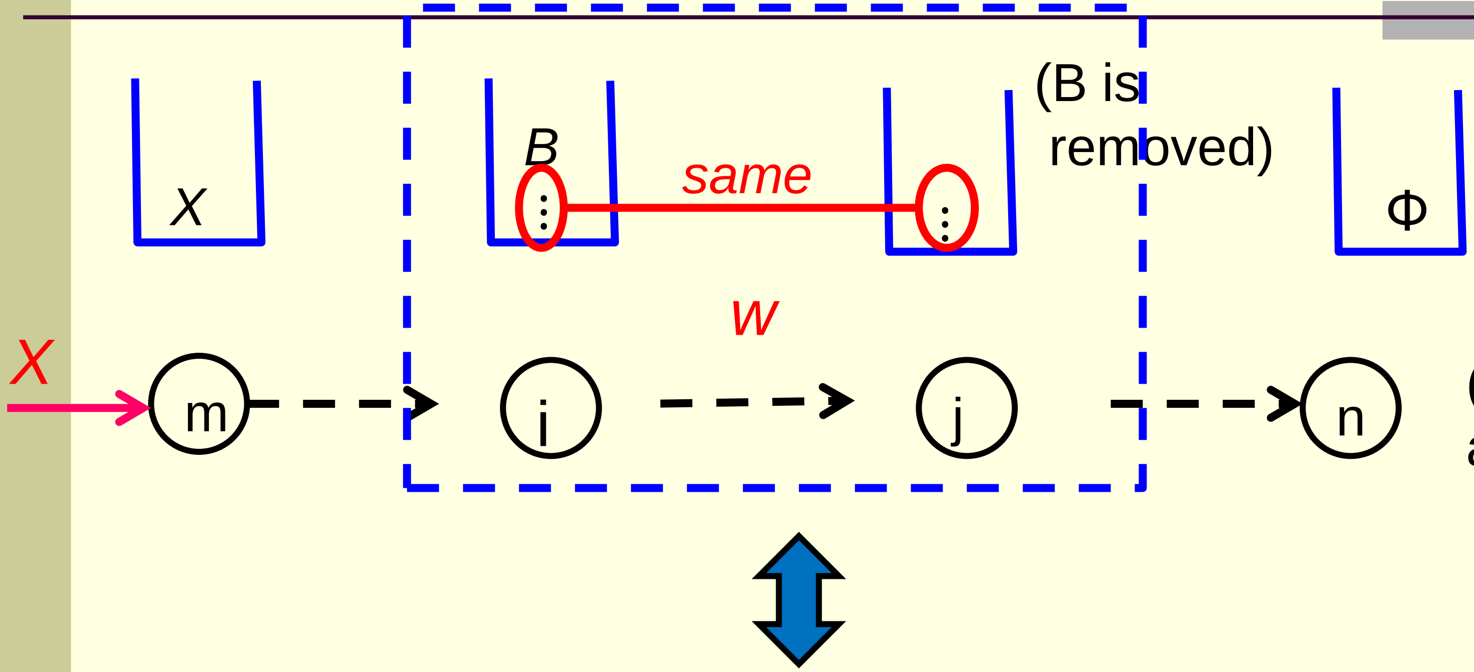
$$B_{ij} \Rightarrow \boxed{} \Rightarrow \boxed{} \Rightarrow \cdots \Rightarrow \boxed{w}$$

(w is generated)

(leftmost derivation)

Transform an empty-stack PDA to a C-F grammar

Case 2:



(w is part of an accepted string)

$$B_{mn} \Rightarrow \cdots w_1 B_{ij} \Rightarrow \boxed{} \cdots \Rightarrow \boxed{w} \quad B_{jn}$$

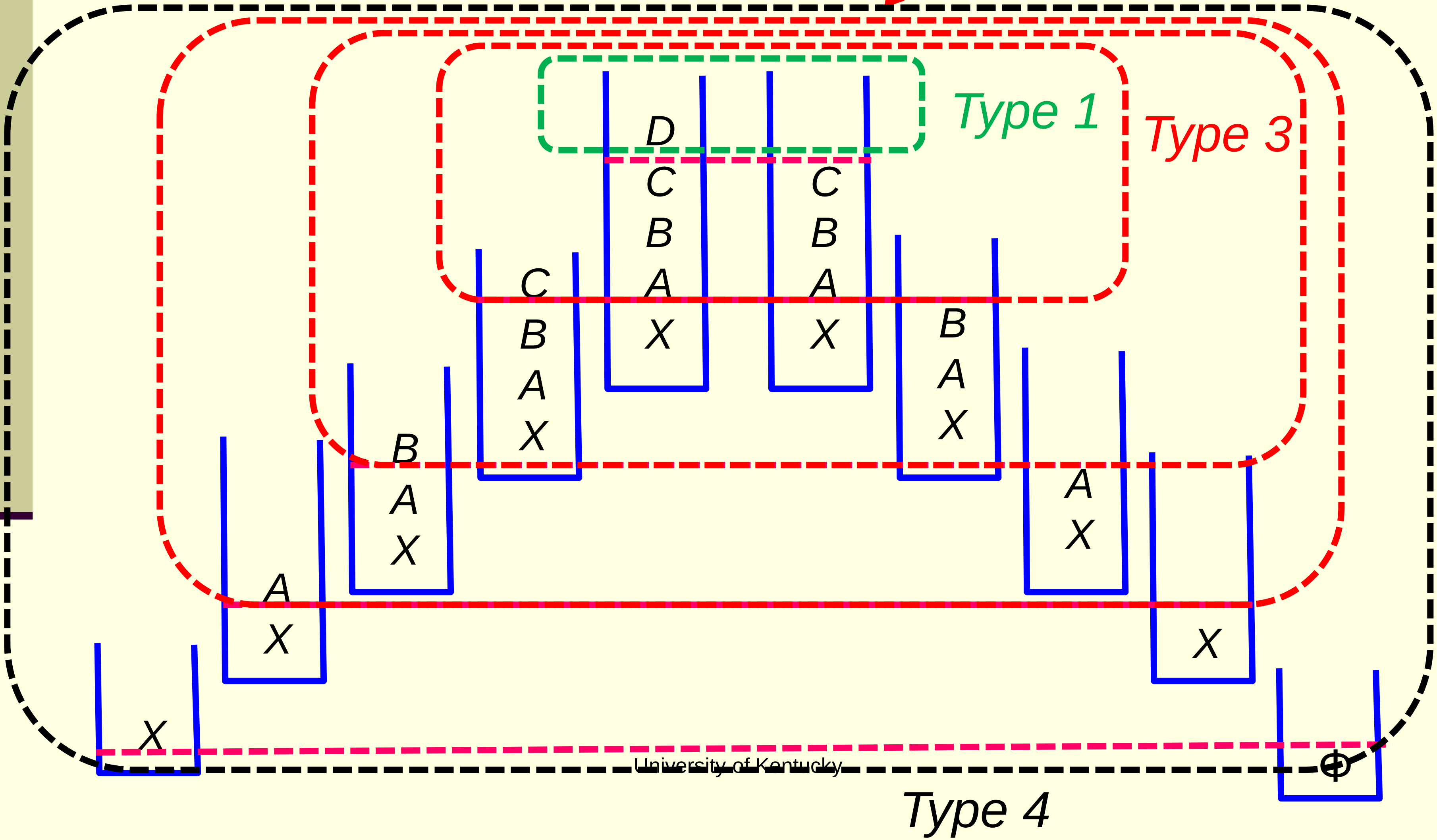
(w is part of a generated string)

(leftmost derivation)

What is the point?

A *Pascal Triangle*-like chart

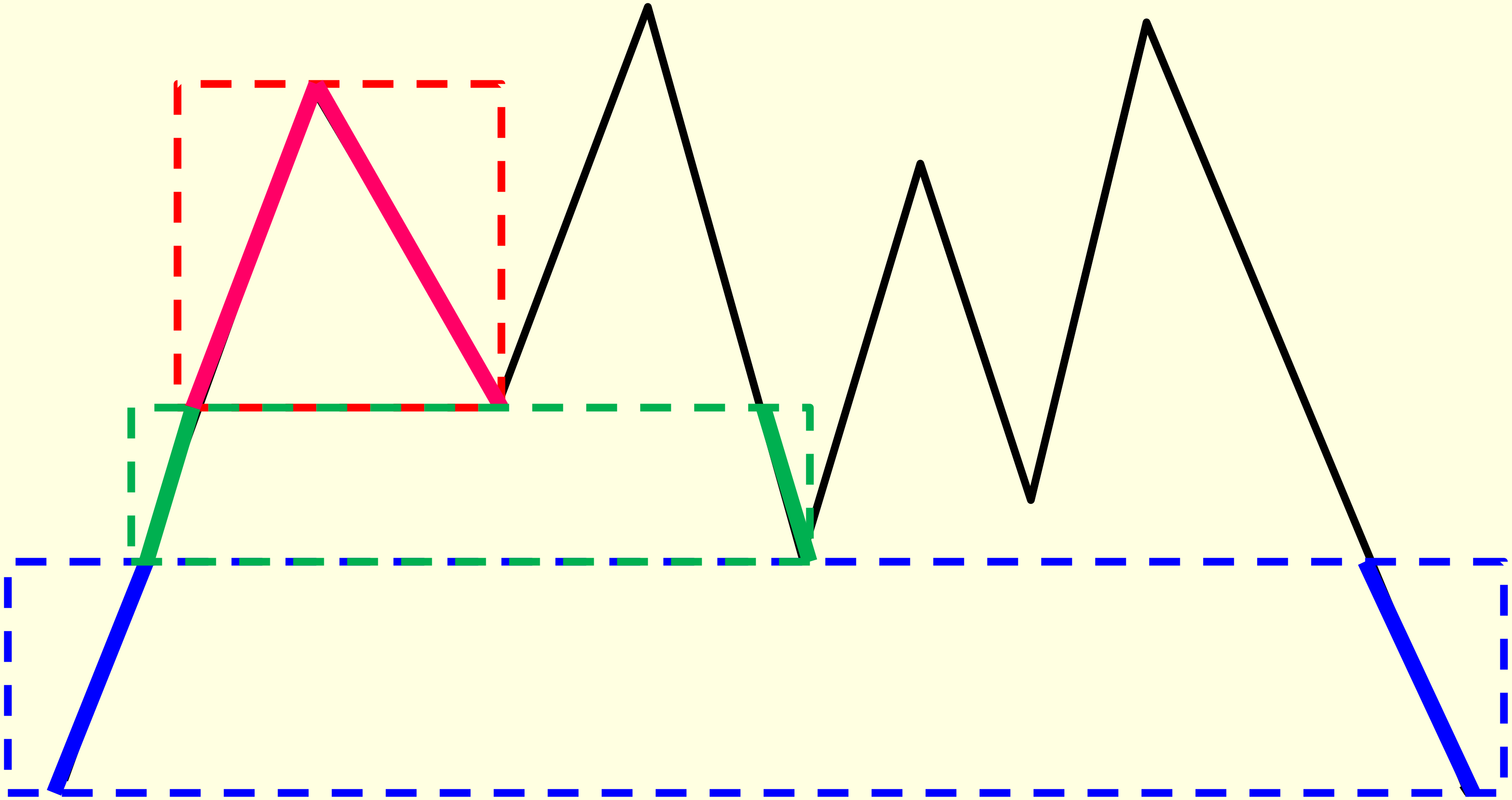
The acceptance process can be decomposed as a sequence of net-loss-of one-stack-symbol's



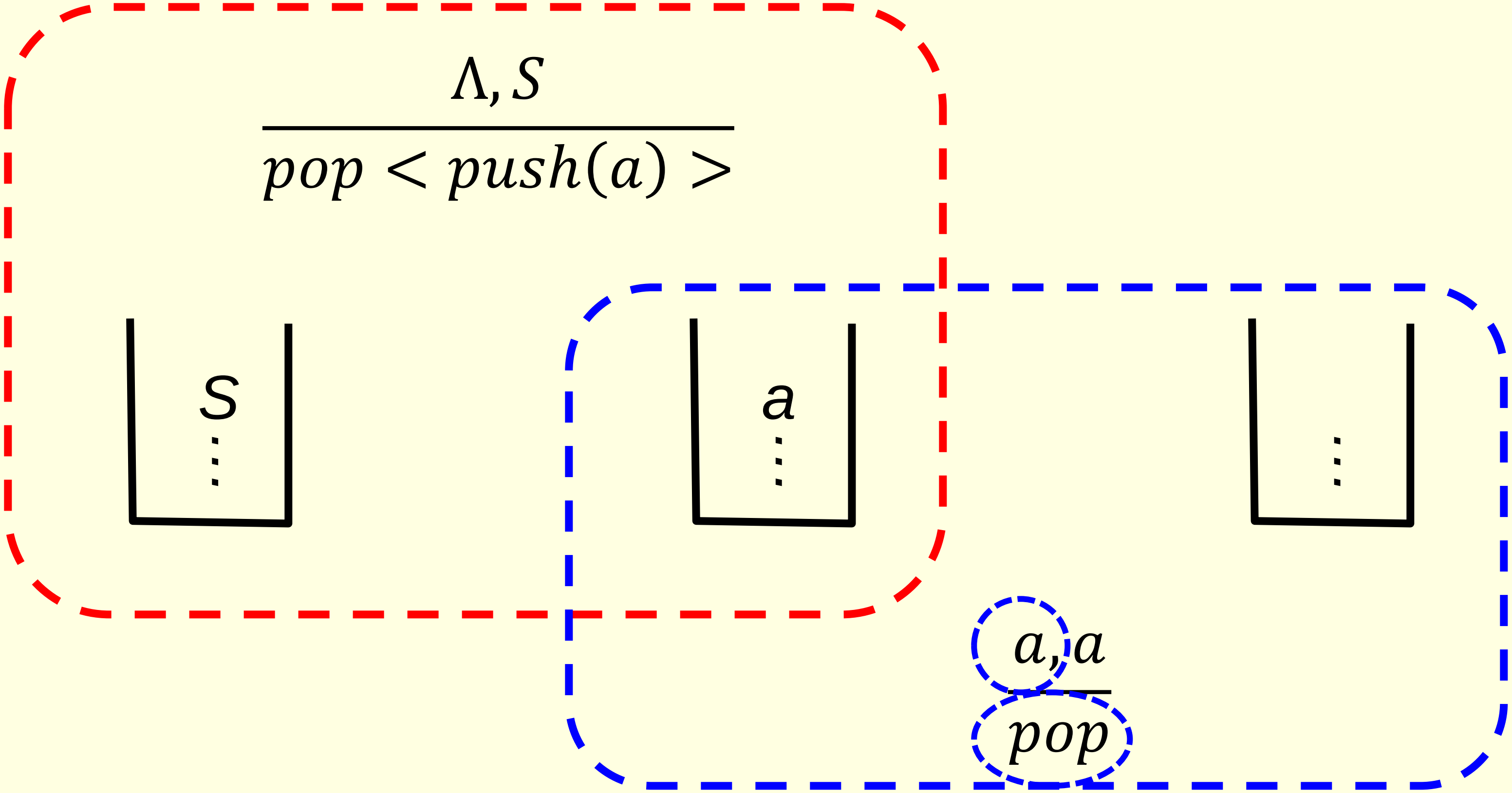
For each push there must be a pop.

Otherwise, the stack wouldn’t be empty eventually.

There could be several peaks



How is $S \rightarrow a$ implemented?



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

$$\frac{a, a}{pop}$$

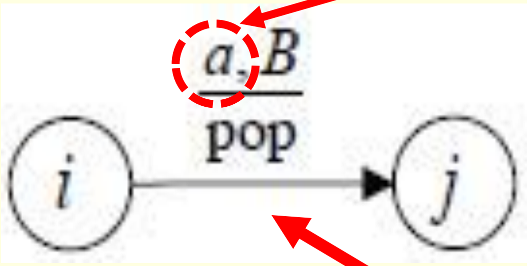
Transform an empty-stack PDA to a C-F grammar

'a' *is the letter/symbol accepted/output*

Type 1:

PDA instruction

Grammar Production



$$B_{ij} \to a$$

can reach an empty-stack status

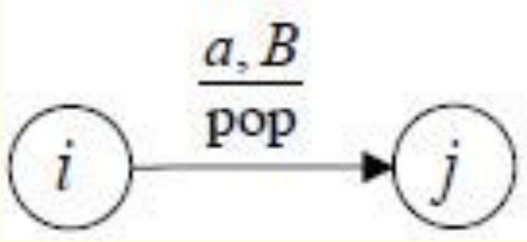
Transform an empty-stack PDA to a C-F grammar

Type 1:

The string accepted

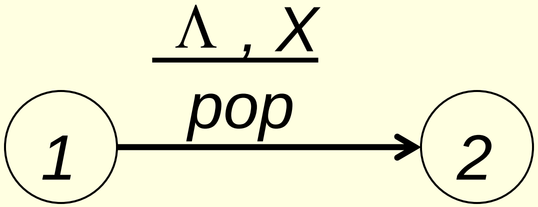
PDA instruction

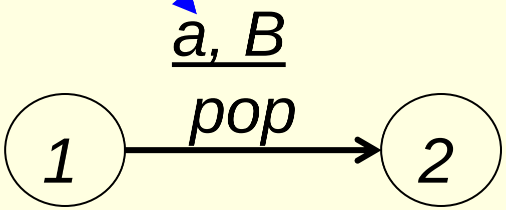
Grammar Production



$$B_{ij} \to a$$

Example:





$$X_{12} \to \Lambda$$

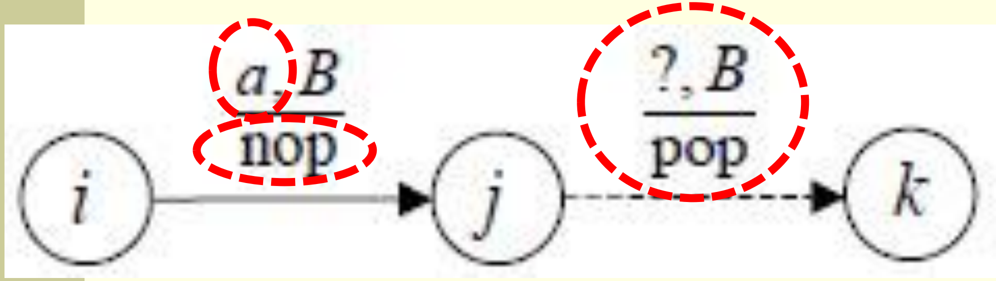
$$B_{12} \to a$$

Transform an empty-stack PDA to a C-F grammar

Type 2:

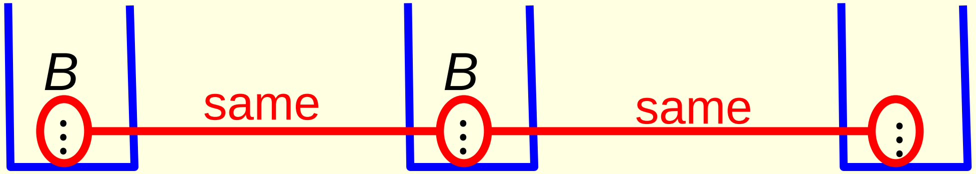
PDA instruction

Grammar Production



$$B_{ik} \to a\ B_{jk}$$

for each state k

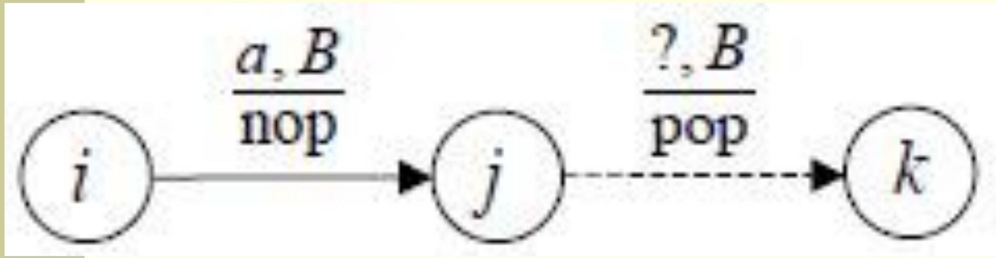


accepted string is ‘a’ followed by whatever is accepted between state j and state k

Transform an empty-stack PDA to a C-F grammar

PDA instruction

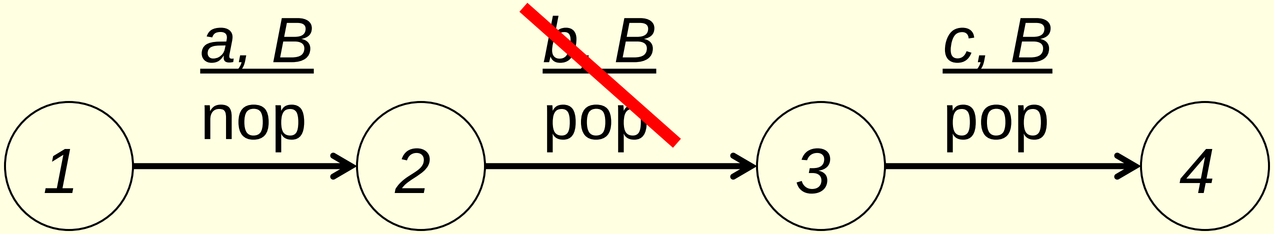
Grammar Production



$$B_{ik} \to aB_{jk}$$

$$\text{for each state } k$$

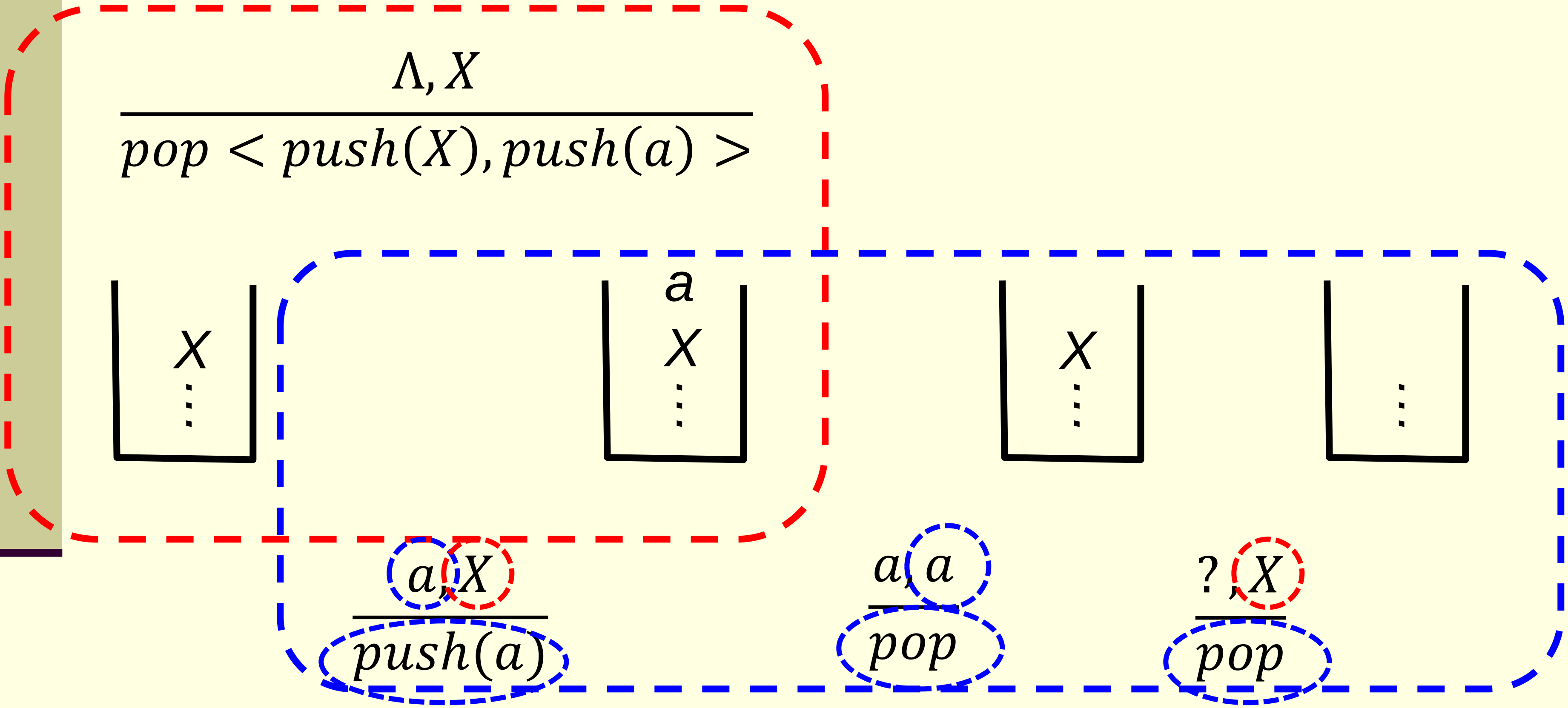
Example:



$$\boxed{B_{13} \to aB_{23}}$$

$$\boxed{B_{14} \to aB_{24}}$$

How is $X \rightarrow aX$ implemented?



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

$$\frac{a, X}{push(a)}$$

$$\frac{a, a}{pop}$$

$$\frac{?, X}{pop}$$

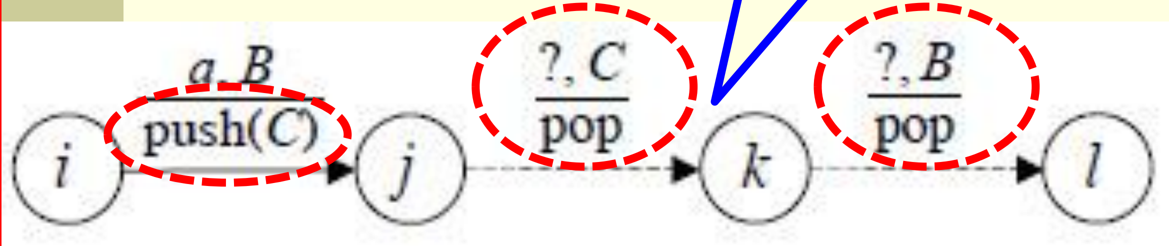
Transform an empty-stack PDA to a C-F grammar

Type 3:

The path has to be a workable path

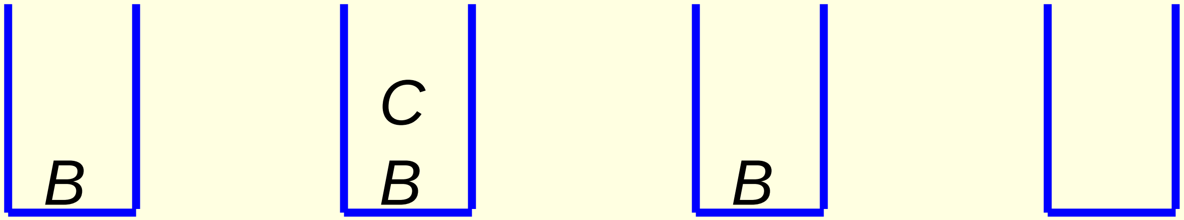
PDA instruction

Grammar Production



$$B_{il} \to aC_{jk}B_{kl}$$

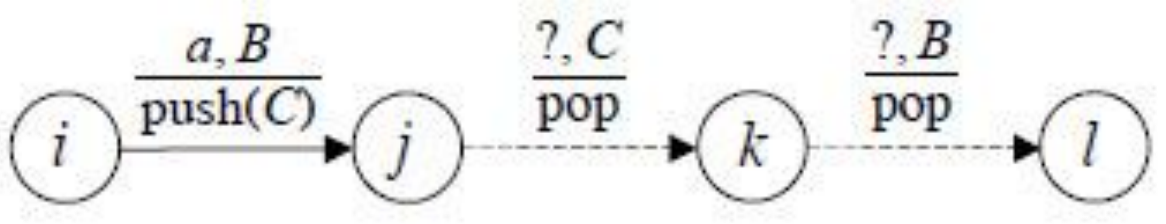
for each state k and l



Transform an empty-stack PDA to a C-F grammar

PDA instruction

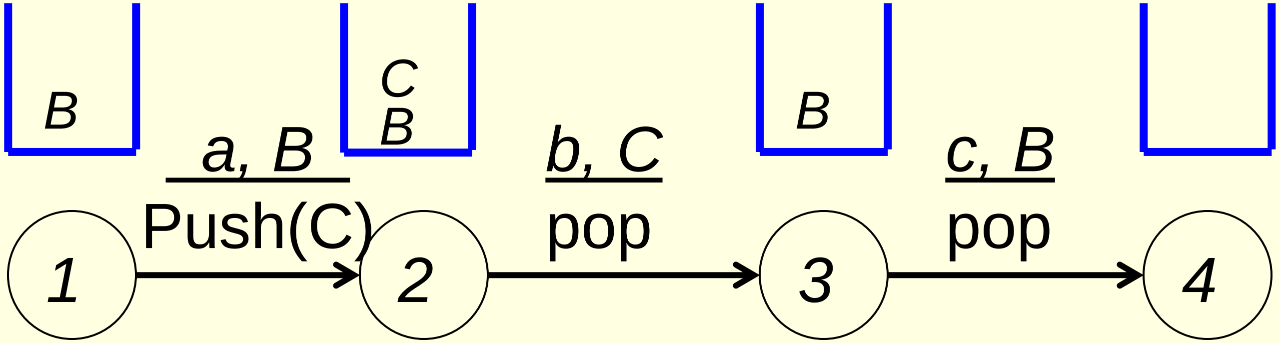
Grammar Production



$$B_{il} \to a C_{jk} B_{kl}$$

for each state k and l

Example:



$$\boxed{B_{14} \to a\, C_{23} B_{34}}$$

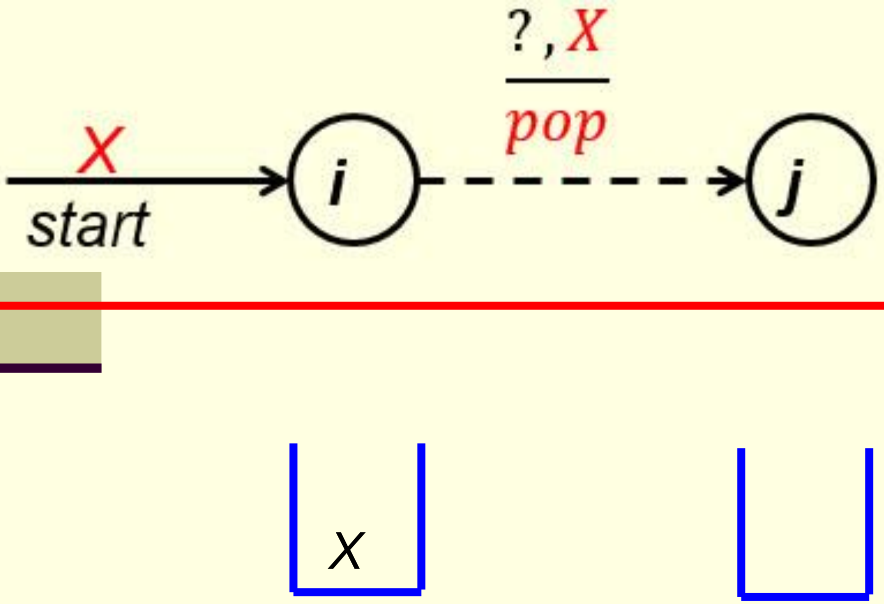
Transform an empty-stack PDA to a C-F grammar

The production that will generate the string accepted by the PDA between state i and state j

Type 4:

PDA instruction

Grammar Production



$$S \rightarrow X_{ij}$$

for each state j

(S is start symbol)

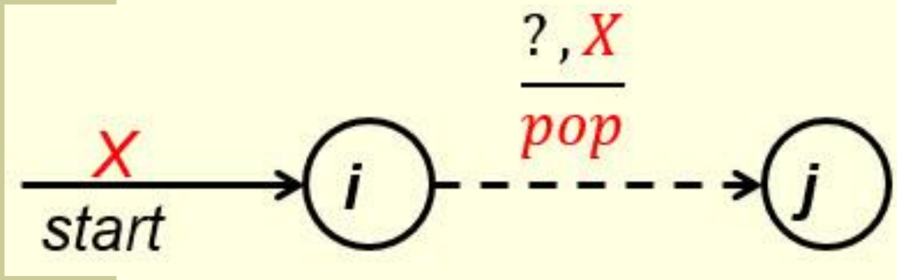
Could lead to an empty-stack status

Transform an empty-stack PDA to a C-F grammar

Type 4:

PDA instruction

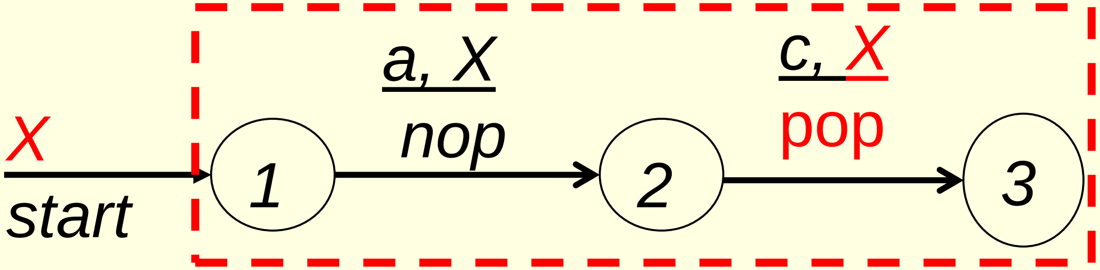
Grammar Production



$$S \to X_{ij}$$

for each state j

Example:



$S \to X_{13}$

$X_{13} \to aX_{23}$ **(Type 2)**

Transform an empty-stack PDA to a C-F grammar

The order CFG productions are constructed:

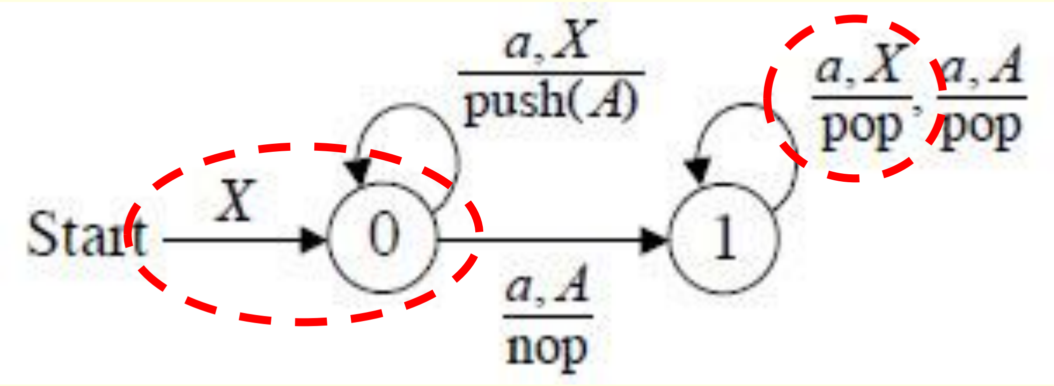
Type 4

Type 1

Type 2 (might not exist)

Type 3

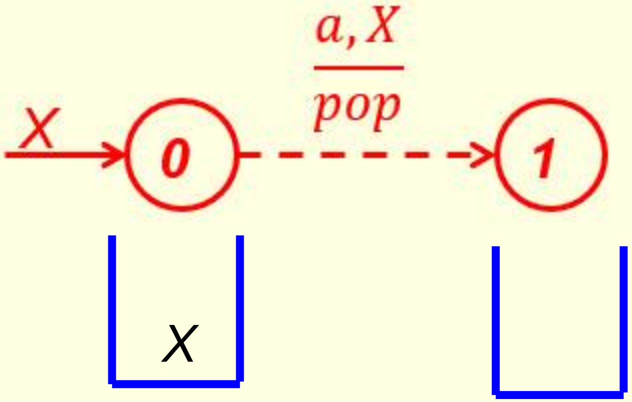
Example. Transform the following empty-stack PDA into a C-F grammar.



Solution:

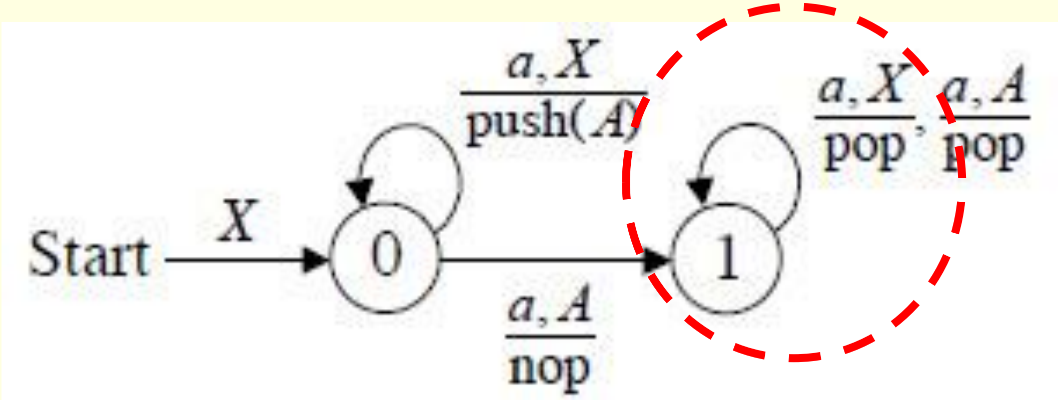
Type 4

The start state 0 and $\frac{a,X}{pop}$ give:



$$\boxed{S \to X_{01}}$$

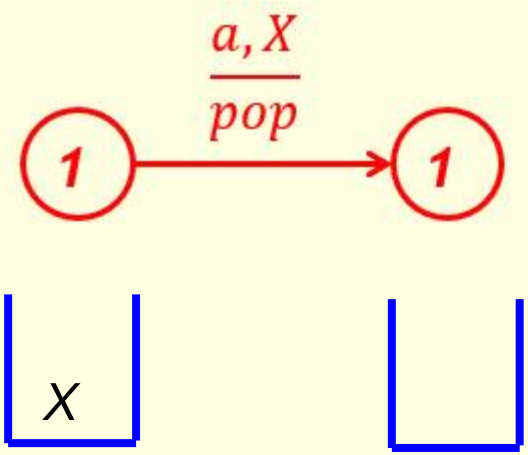
Example. Transform the following empty-stack PDA into a C-F grammar.



Solution:

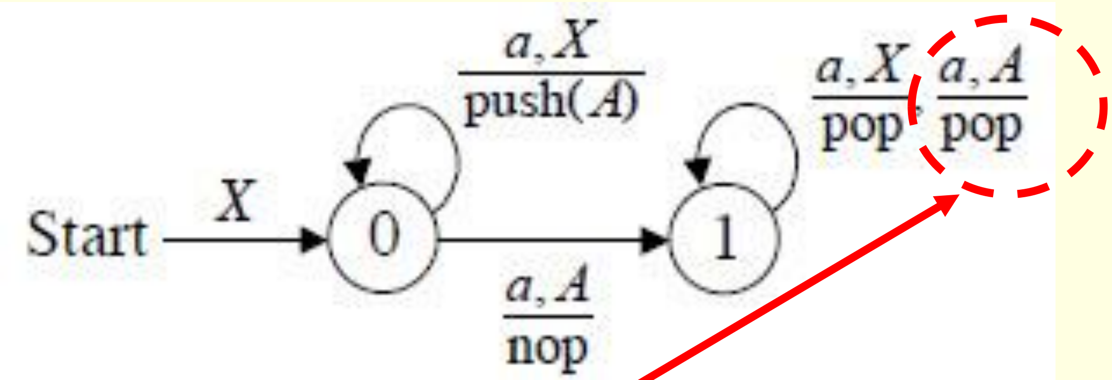
Type 1

The pop operation (1, *a, X, pop, 1)* gives



$$\boxed{X_{11} \to a}$$

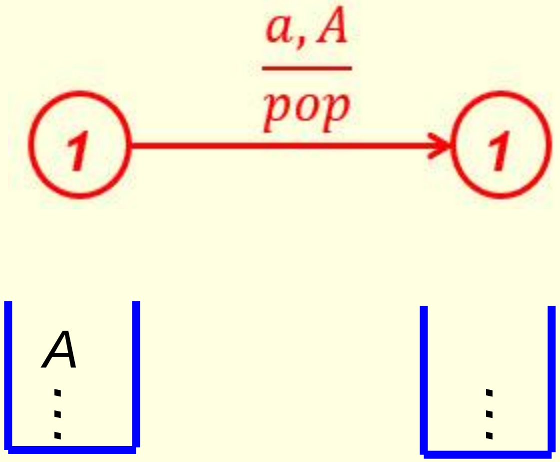
Example. Transform the following empty-stack PDA into a C-F grammar.



Solution:

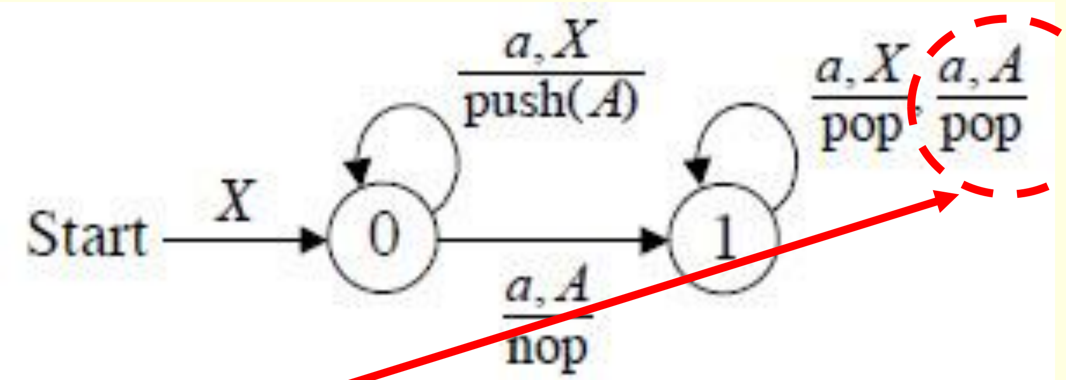
Type 1

The pop operation (1, *a, A, pop, 1)* gives



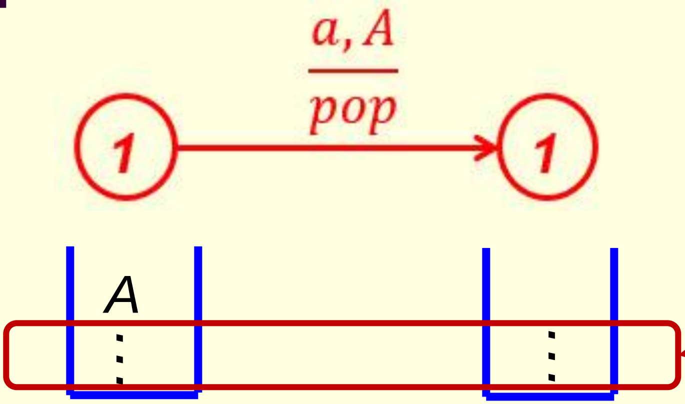
$$\boxed{A_{11} \to a}$$

Example. Transform the following empty-stack PDA into a C-F grammar.



Type 1

Question: a,A/pop is not the last step of an acceptance path, why the right hand side of the production has only a terminal?



$$A_{11} \to a$$

Because this part will be handled by non-terminals contained in previously defined production steps

Here is why:

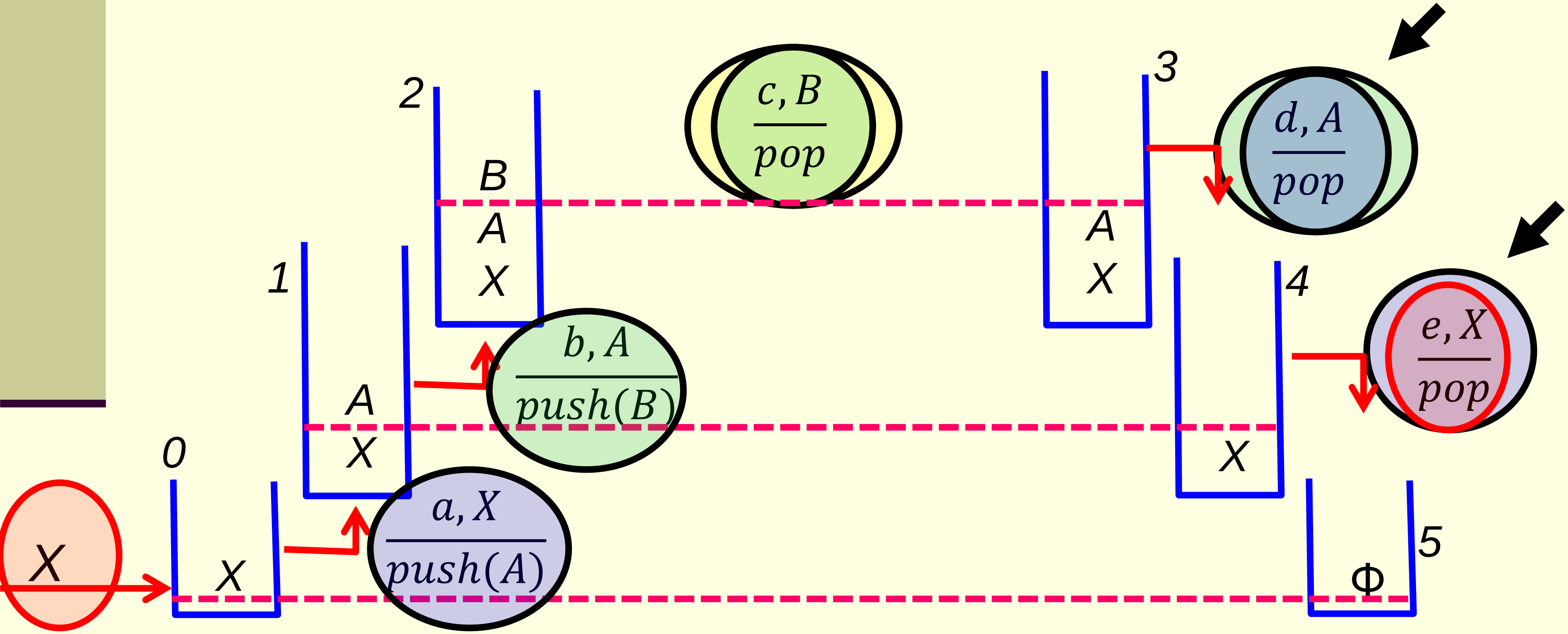
$$S \rightarrow X_{05} \rightarrow aA_{14}X_{45} \rightarrow abcdX_{45} \rightarrow abcde$$

$$A_{14} \rightarrow bB_{23}A_{34} \rightarrow bcd$$

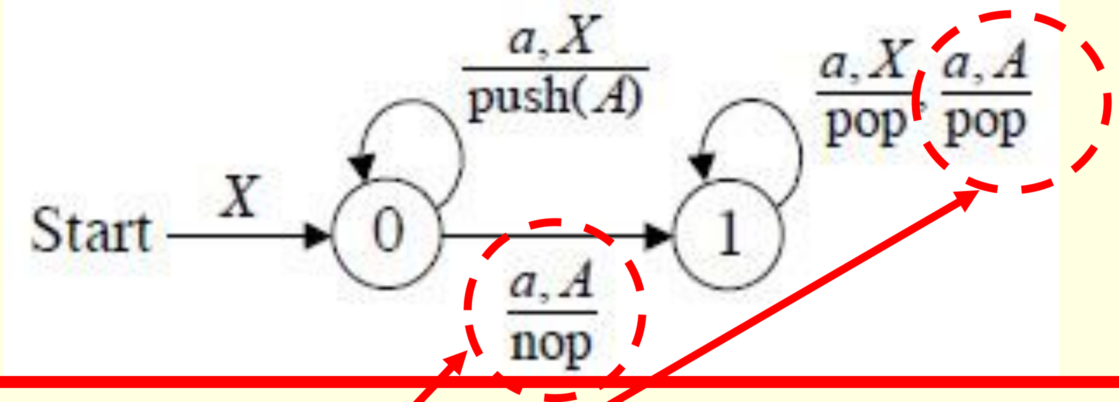
$$B_{23} \rightarrow c$$

$$A_{34} \rightarrow d$$

$$X_{45} \rightarrow e$$



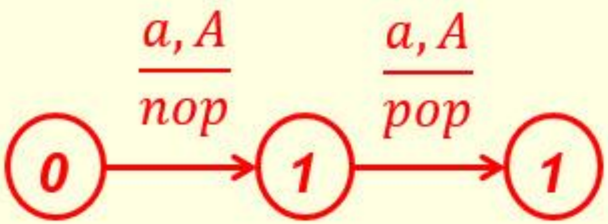
Example. Transform the following empty-stack PDA into a C-F grammar.



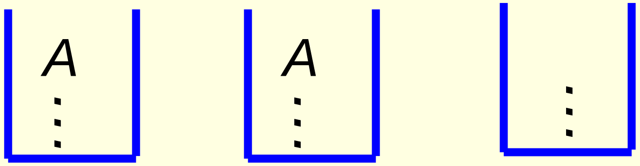
Solution:

Type 2

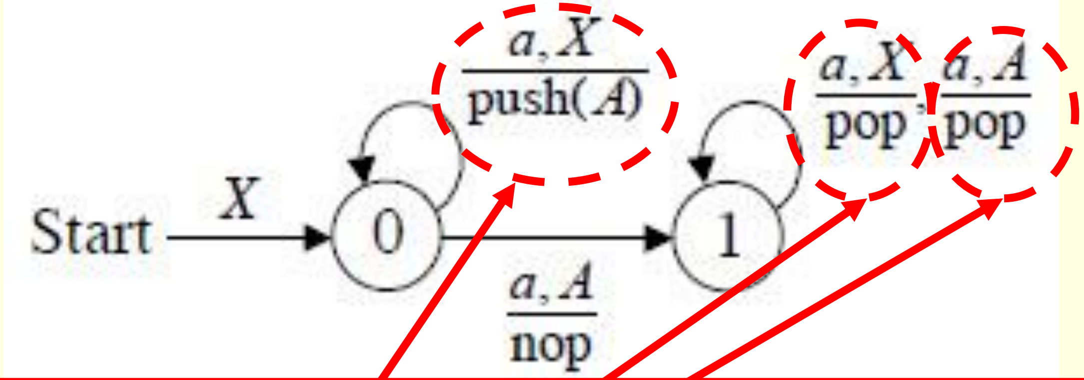
The nop operation (0, *a, A, nop, 1)* gives



$$\boxed{A_{01} \to aA_{11}}$$



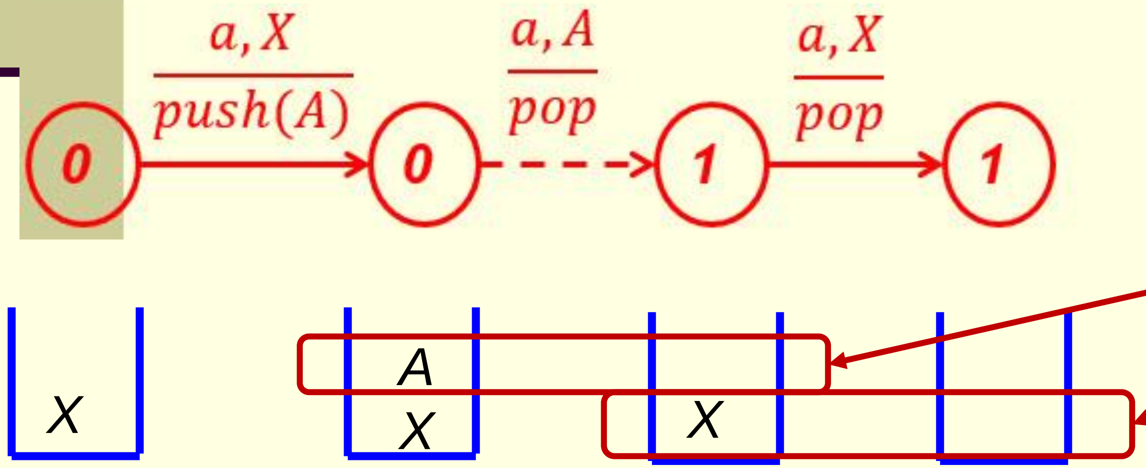
Example. Transform the following empty-stack PDA into a C-F grammar.



Solution:

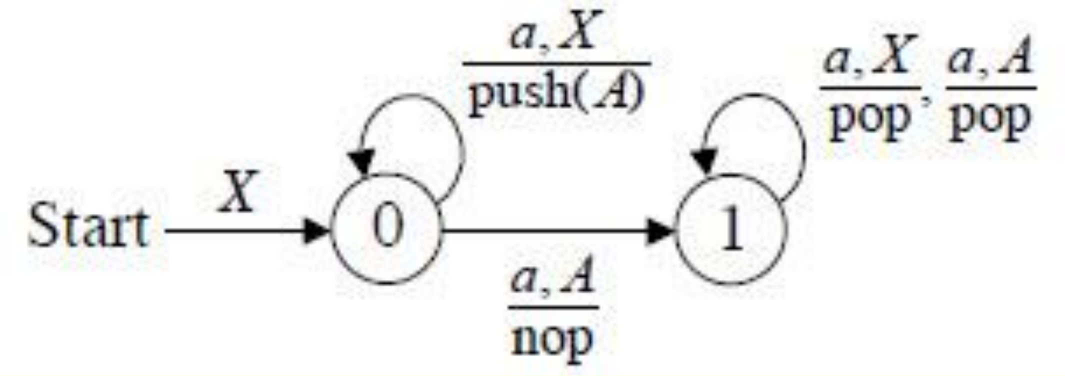
Type 3

The push operation (0, *a*, *X*, *push(A)*, *0)* gives



$$\boxed{X_{01} \rightarrow aA_{01}X_{11}}$$

Empty-stack PDA:



C-F Grammar:

$$S \to X_{01}$$
$$X_{11} \to a$$
$$A_{11} \to a$$
$$A_{01} \to aA_{11}$$
$$X_{01} \to aA_{01}X_{11}$$

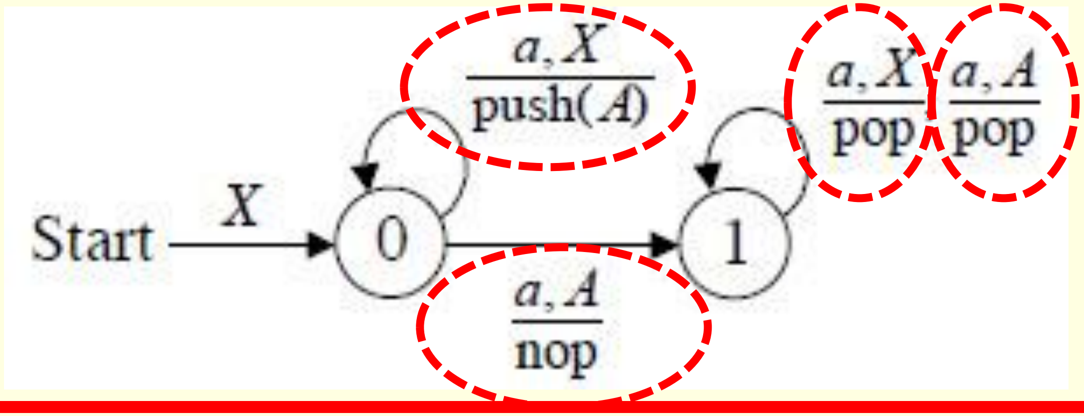
Leftmost

derivation

$$S \Rightarrow X_{01}$$
$$\Rightarrow aA_{01}X_{11}$$
$$\Rightarrow aaA_{11}X_{11}$$
$$\Rightarrow aaaX_{11}$$
$$\Rightarrow aaaa$$

The language accepted by this PDA has only one element : aaaa

Empty-stack PDA:



C-F Grammar:

$$
\left.
\begin{aligned}
S &\rightarrow X_{01} \\
X_{11} &\rightarrow a \\
A_{11} &\rightarrow a \\
A_{01} &\rightarrow aA_{11} \\
X_{01} &\rightarrow aA_{01}X_{11}
\end{aligned}
\right]
$$

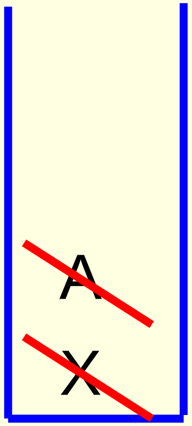
(0, aaaa, X)

(0, aaa, AX)

(1, aa, AX)

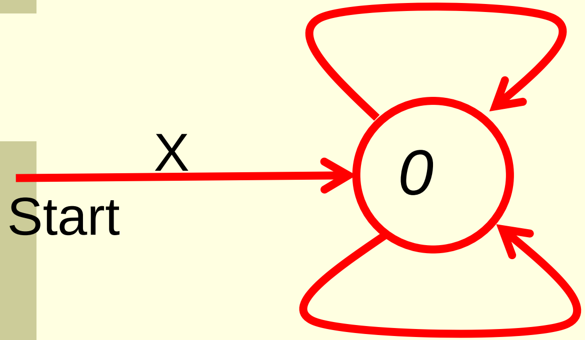
(1, a, X)

Accepted



This PDA accepts only one string: aaaa

How to handle an empty-stack PDA of the following type:



$$\frac{a,\ a}{pop} \qquad \frac{b,\ b}{pop} \qquad \frac{\Lambda,\ X}{pop}$$

$$\frac{\Lambda,\ X}{pop, push\ (a, X, b)} \qquad \frac{\Lambda,\ X}{pop, push\ (a, a, X)}$$

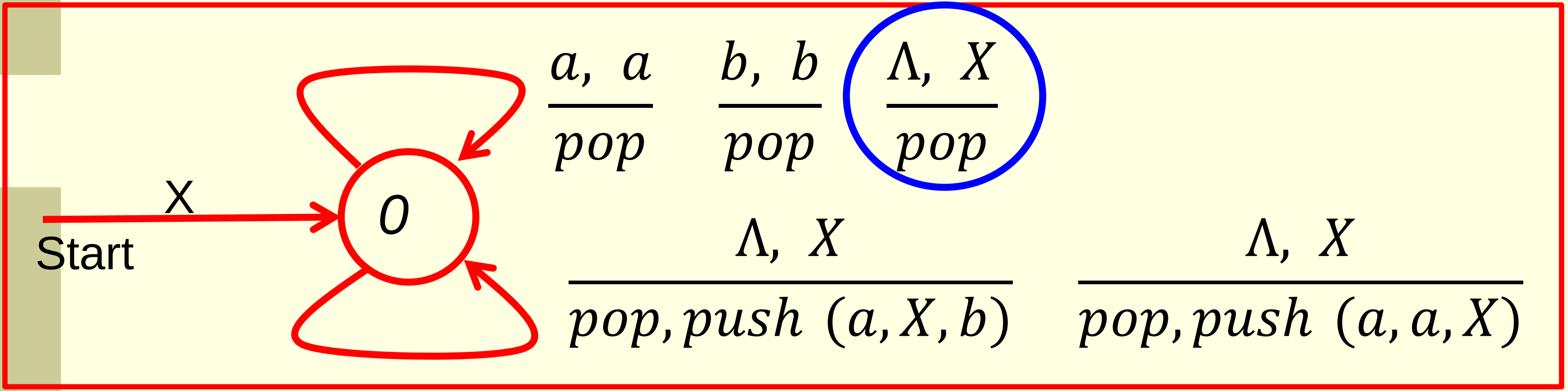
This is the one-state empty-stack acceptance PDA we got for the CFG

$$\begin{aligned} S &\rightarrow \Lambda \\ S &\rightarrow aSb \\ S &\rightarrow aaS \end{aligned}$$

How should a PDA of this form be transformed to a CFG?

Would we be able to transform it back to the original CFG?

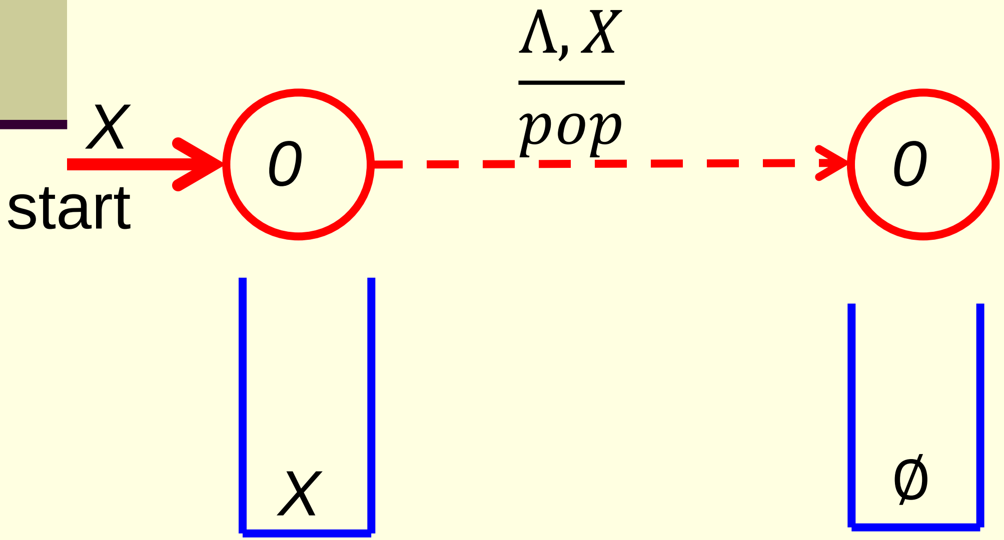
How to handle an empty-stack PDA of the following type:



$\frac{a,\ a}{pop}$ $\frac{b,\ b}{pop}$ $\boxed{\frac{\Lambda,\ X}{pop}}$

$\frac{\Lambda,\ X}{pop, push\ (a, X, b)}$ $\frac{\Lambda,\ X}{pop, push\ (a, a, X)}$

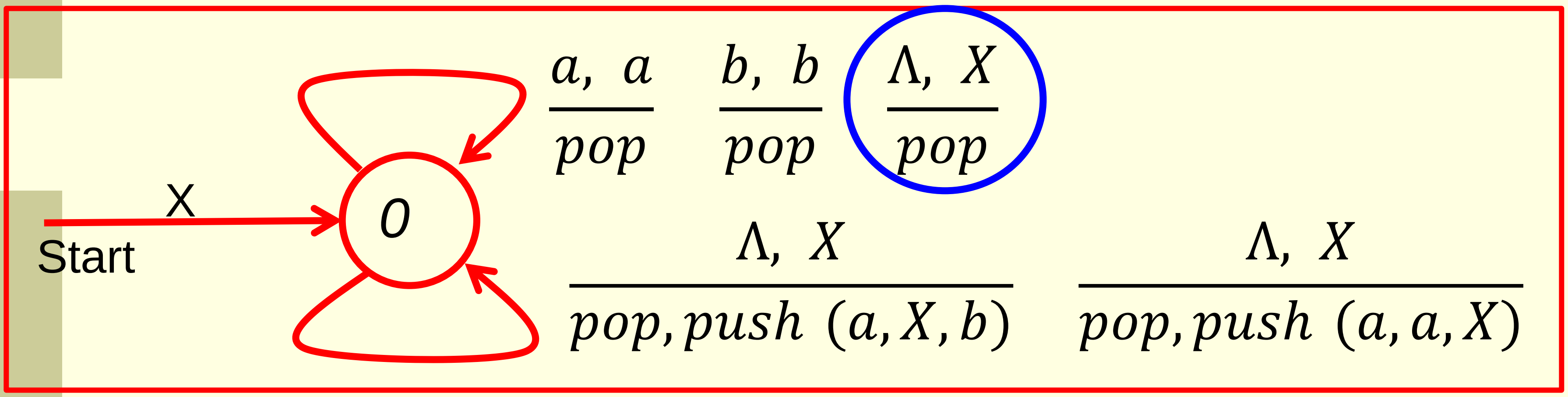
Type 4:



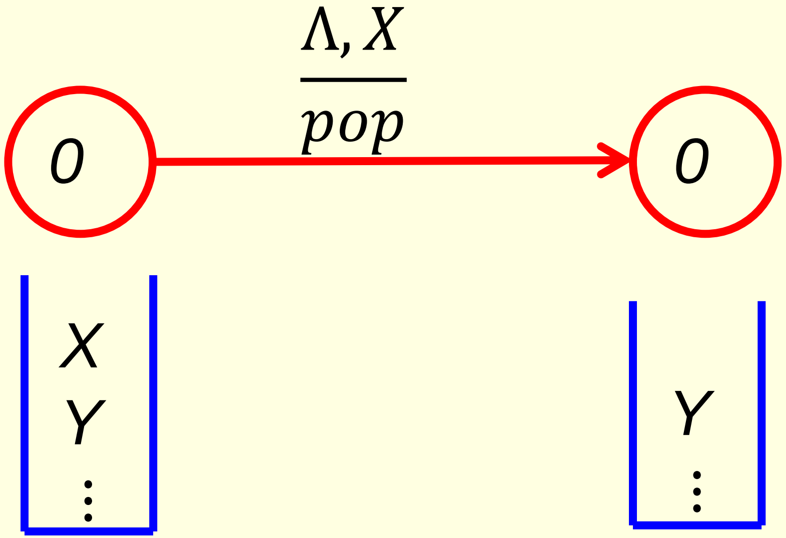
$\frac{\Lambda, X}{pop}$

$$\boxed{S \to X_{00}}$$

What if the given empty-stack PDA is of the following type?

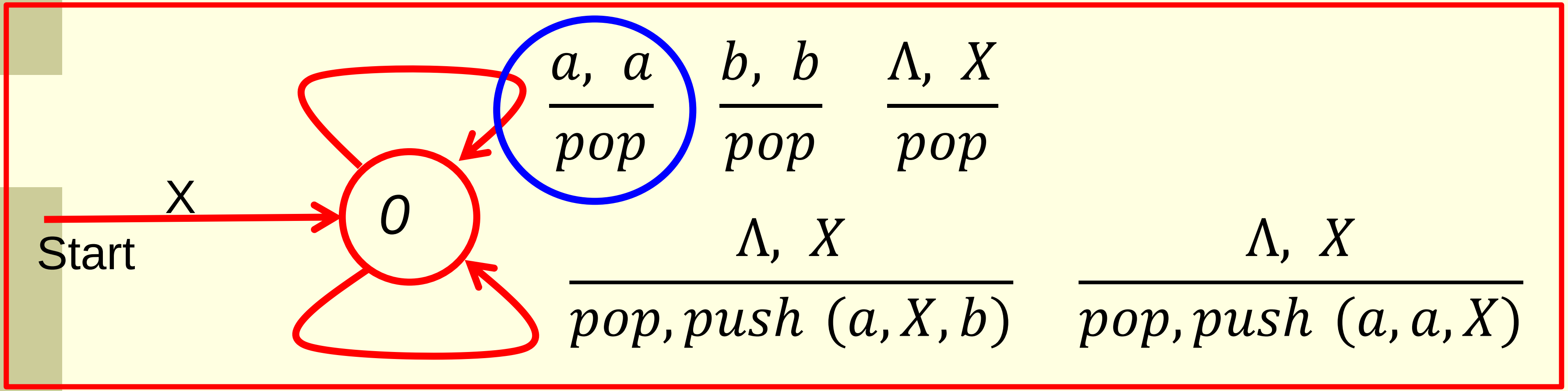


Type 1:

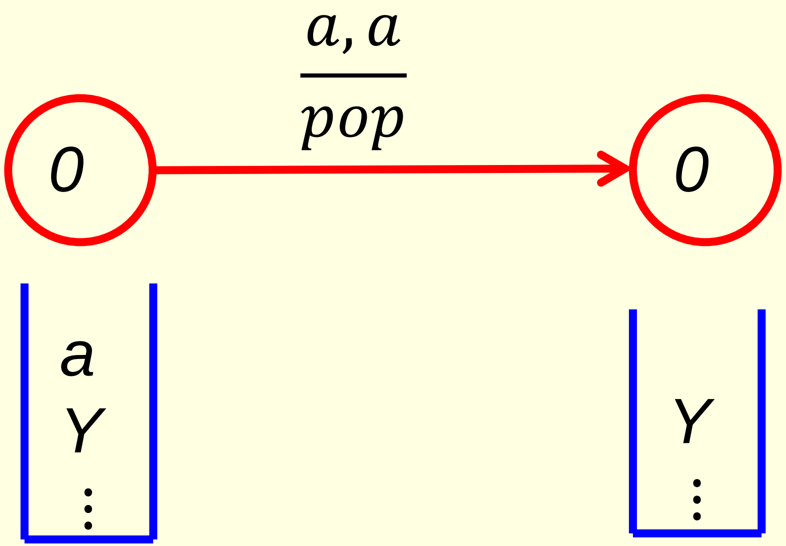


$$\boxed{X_{00} \to \Lambda}$$

What if the given empty-stack PDA is of the following type?

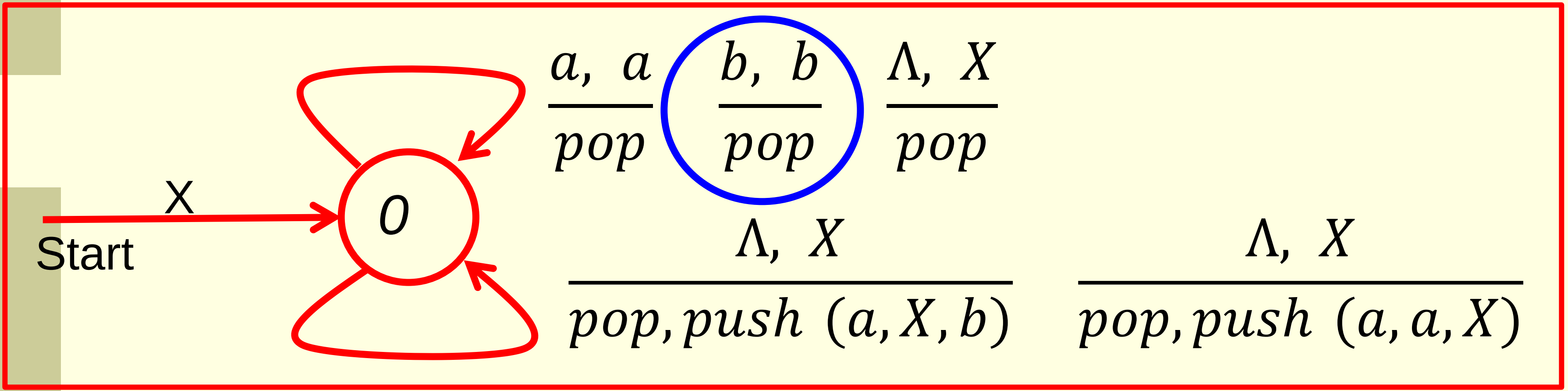


Type 1:



$$\boxed{A_{00} \rightarrow a}$$

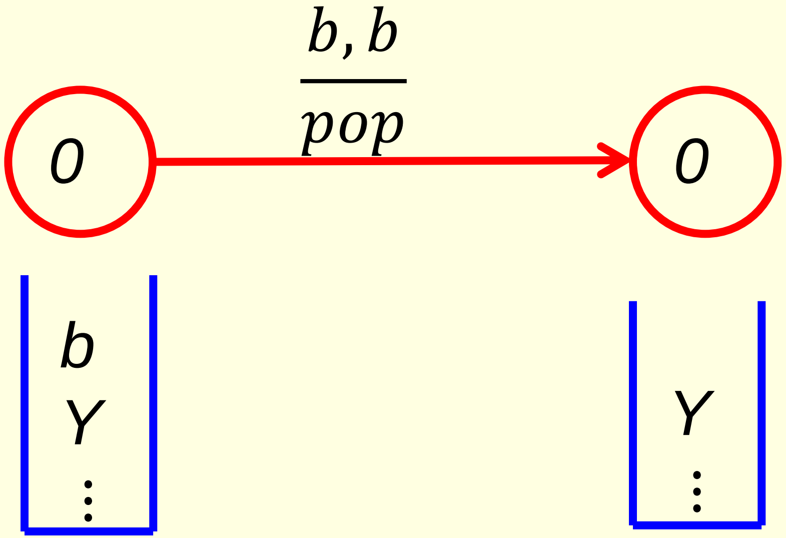
What if the given empty-stack PDA is of the following type?



$$\frac{a,\ a}{pop} \quad \frac{b,\ b}{pop} \quad \frac{\Lambda,\ X}{pop}$$

$$\frac{\Lambda,\ X}{pop, push\ (a, X, b)} \quad \frac{\Lambda,\ X}{pop, push\ (a, a, X)}$$

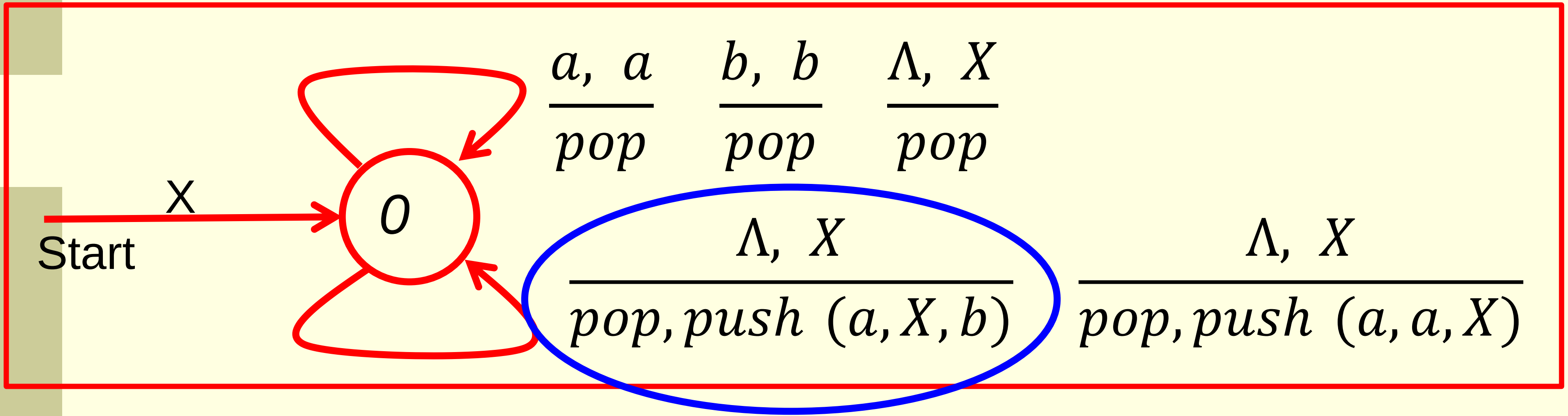
Type 1:



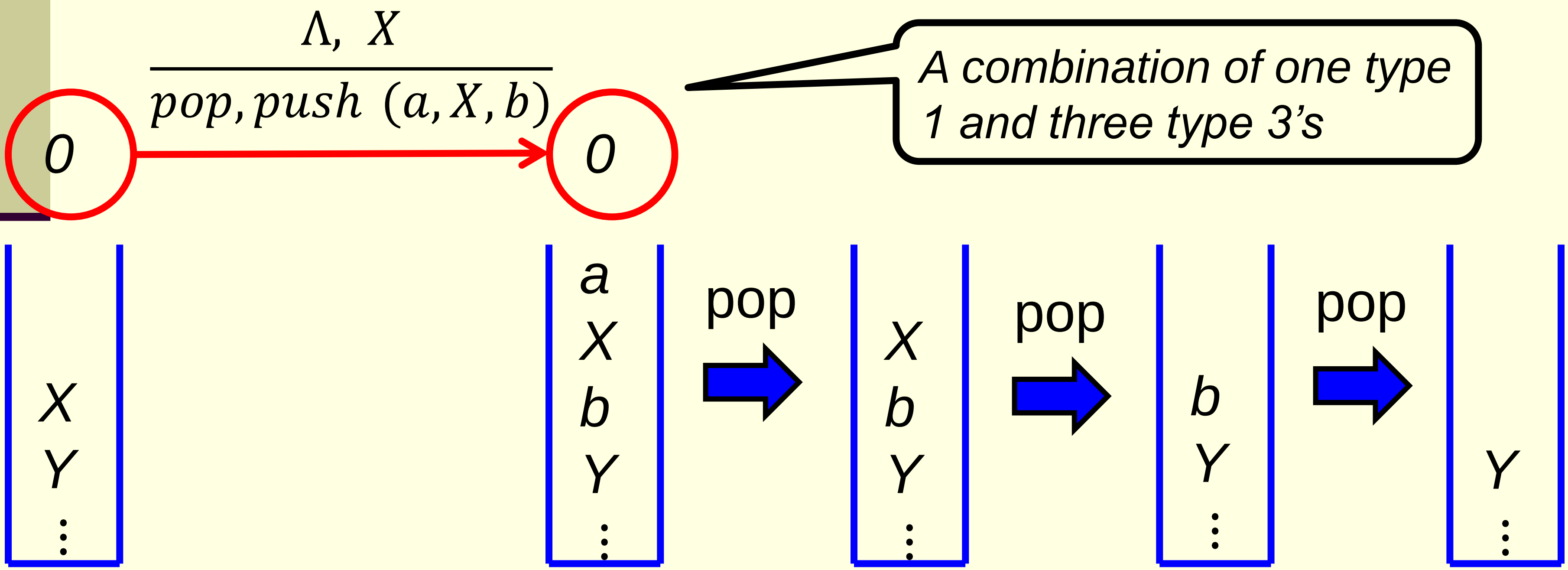
$$\frac{b, b}{pop}$$

$$\boxed{B_{00} \to b}$$

What if the given empty-stack PDA is of the following type?

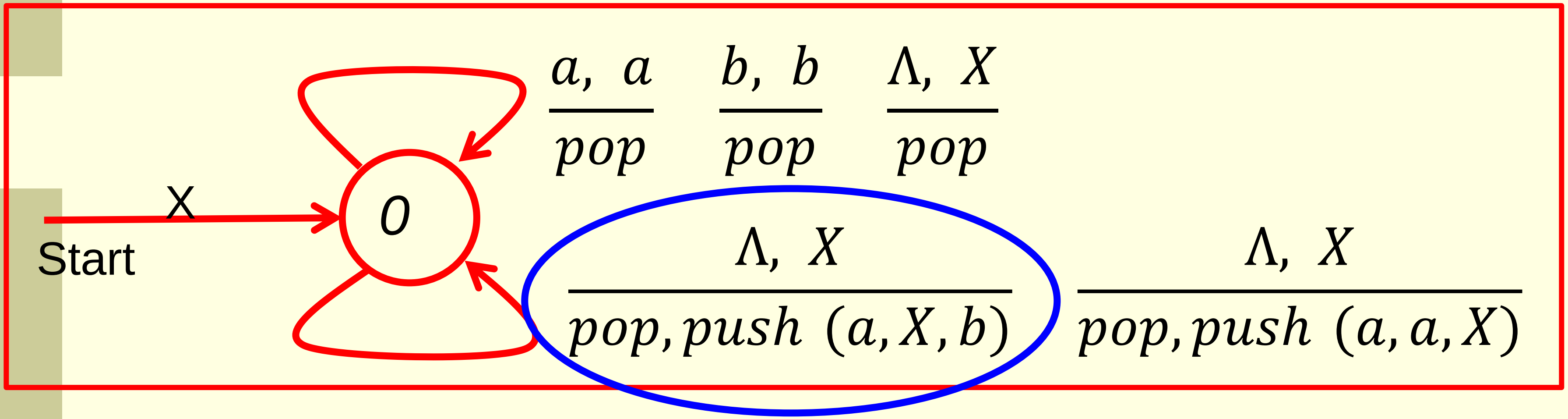


Consider the following situation (**General Type 3**):

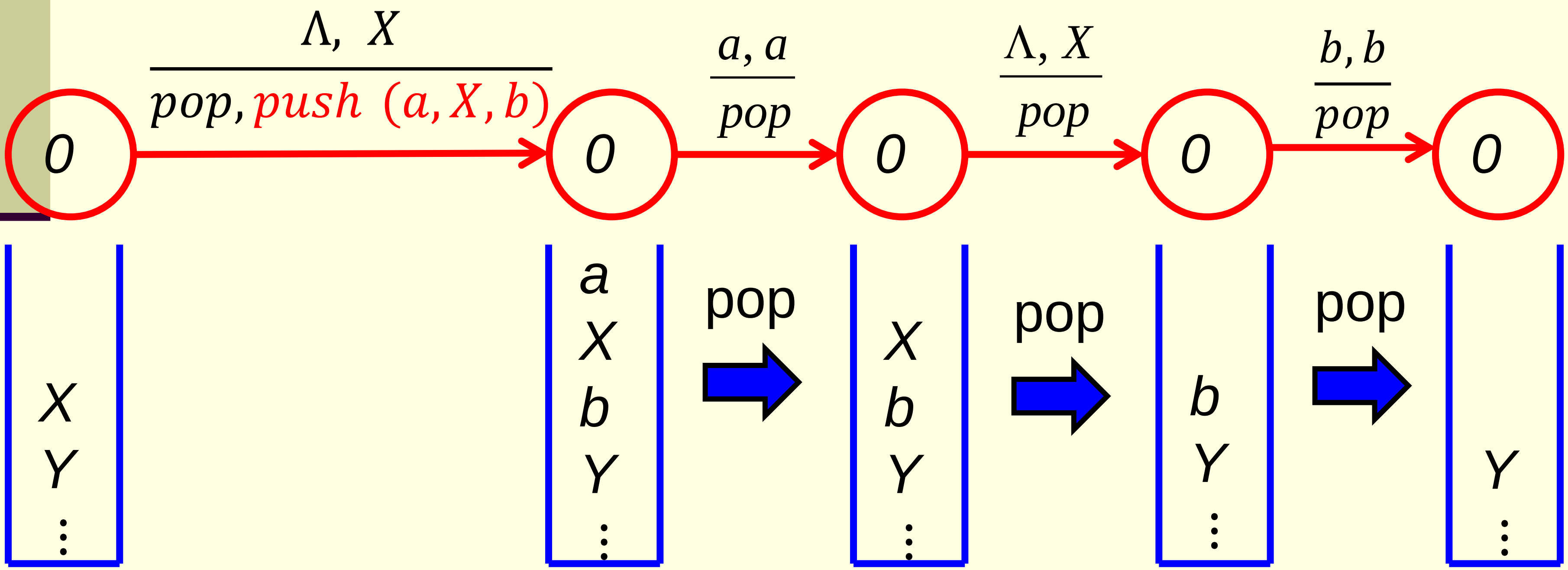


A combination of one type 1 and three type 3's

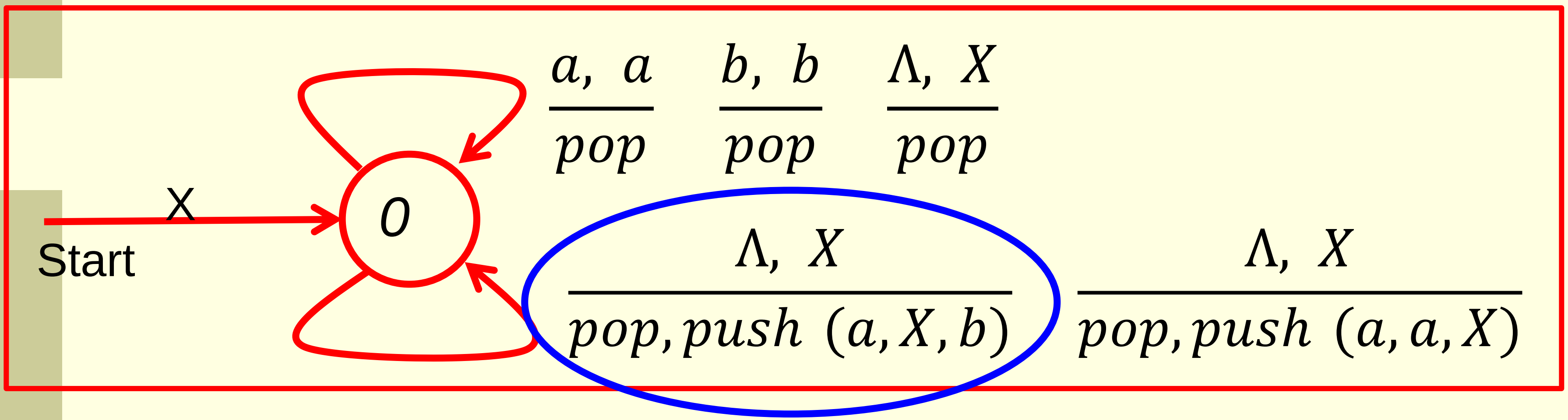
What if the given empty-stack PDA is of the following type?



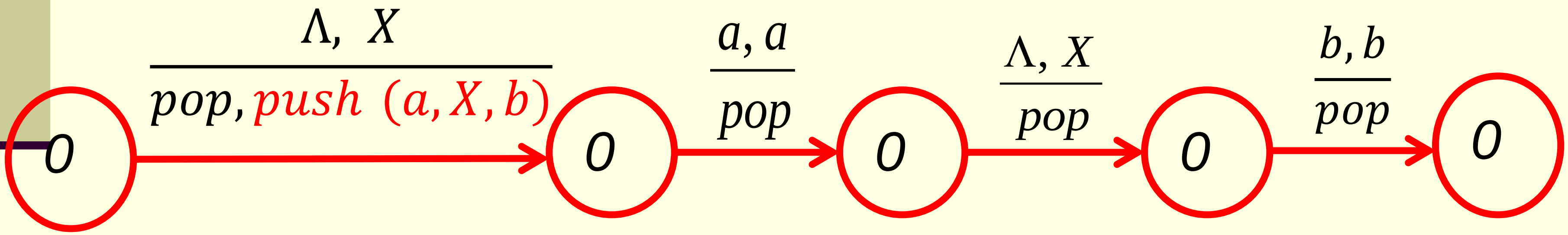
Consider the following situation (**General Type 3**):



What if the given empty-stack PDA is of the following type?

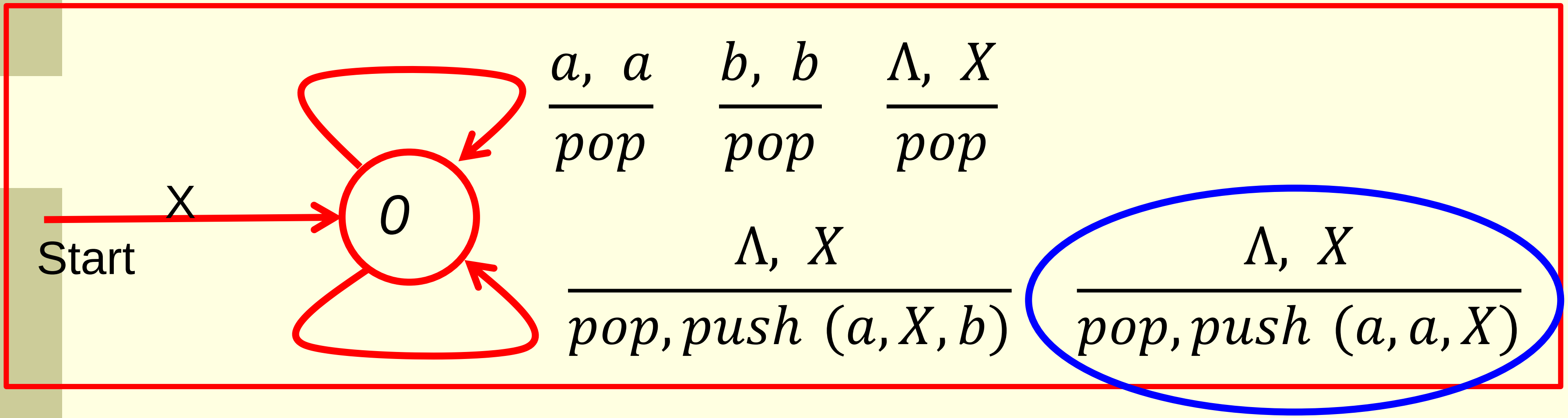


Consider the following situation (**General Type 3**):

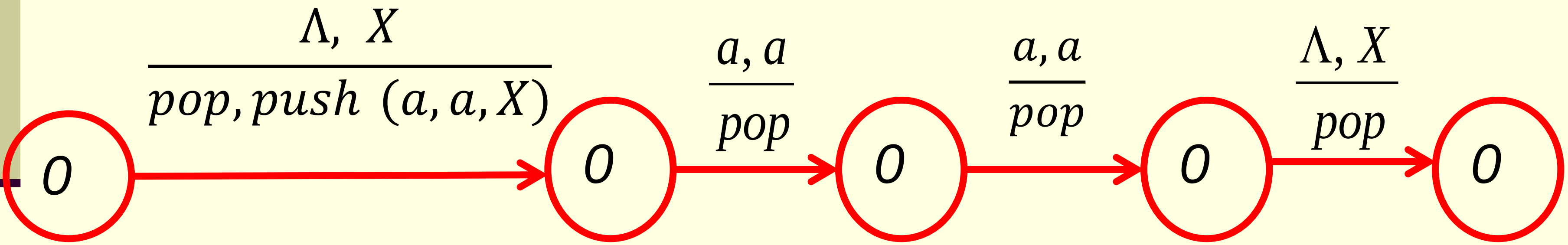


$$\boxed{X_{00} \rightarrow A_{00} X_{00} B_{00}}$$

What if the given empty-stack PDA is of the following type?

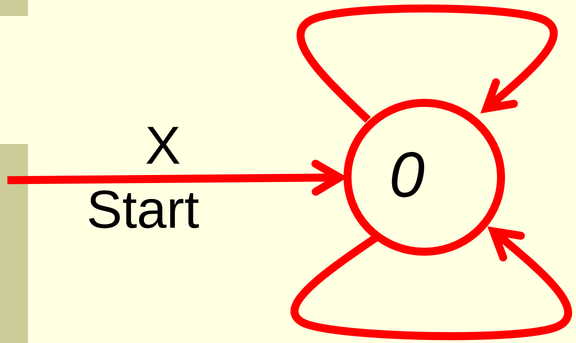


Similarly:



$$\boxed{X_{00} \rightarrow A_{00}A_{00}X_{00}}$$

What if the given empty-stack PDA is of the following type?



$$\frac{a,\ a}{pop} \qquad \frac{b,\ b}{pop} \qquad \frac{\Lambda,\ X}{pop}$$

$$\frac{\Lambda,\ X}{pop, push\ (a, X, b)} \qquad \frac{\Lambda,\ X}{pop, push\ (a, a, X)}$$

So, collectively, we have:

$$S \rightarrow X_{00}$$
$$X_{00} \rightarrow \Lambda$$
$$A_{00} \rightarrow a$$
$$B_{00} \rightarrow b$$
$$X_{00} \rightarrow A_{00}X_{00}B_{00}$$
$$X_{00} \rightarrow A_{00}A_{00}X_{00}$$

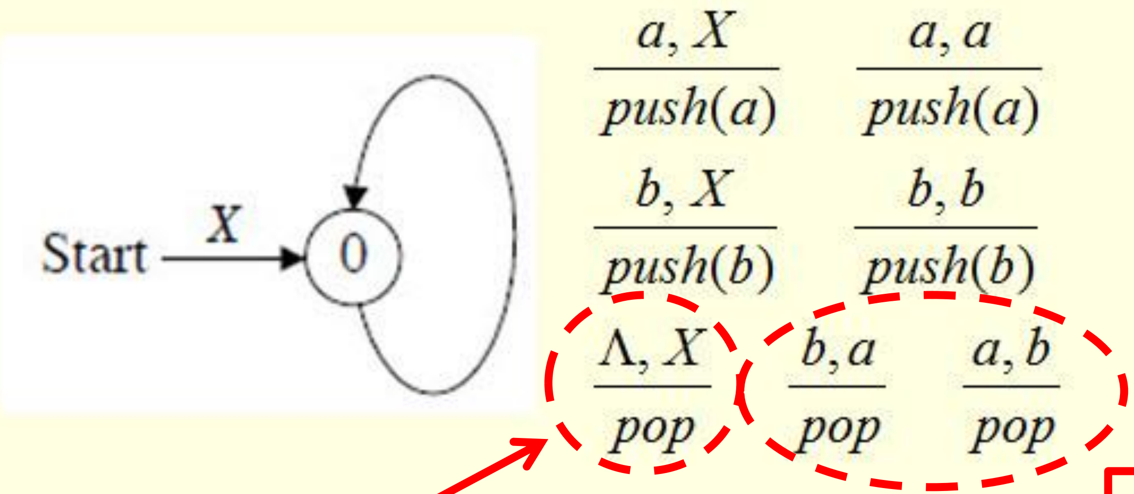
$$S \rightarrow \Lambda$$
$$S \rightarrow aSb$$
$$S \rightarrow aaS$$

Example. Find a grammar for the language

$$L = \{w \in \{a, b\}^* \mid n_a(w) = n_b(w)\}$$

by (1) constructing an empty-stack PDA to accept *L* and then (2) transforming it to a C-F grammar.

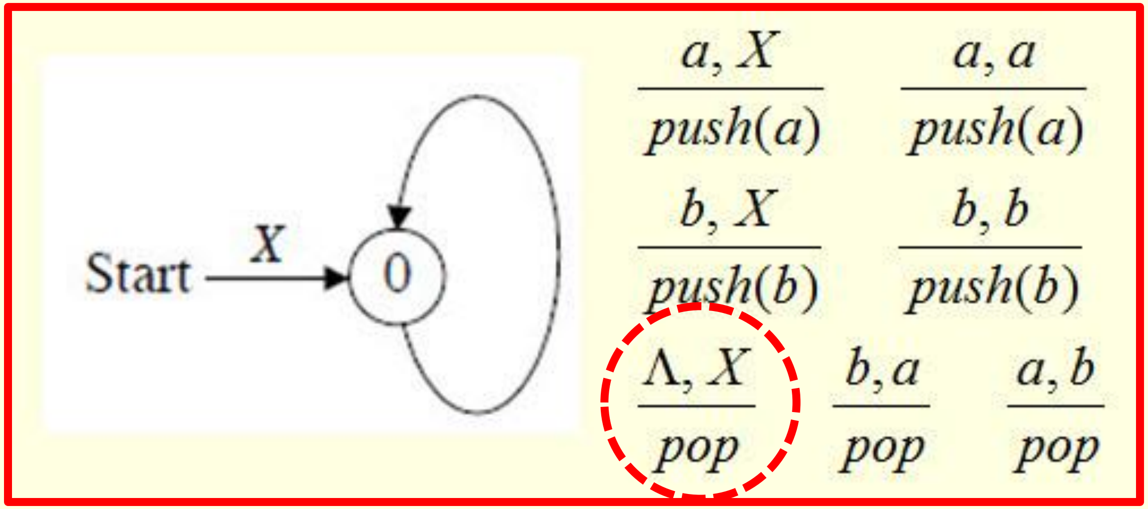
Solution: (1)

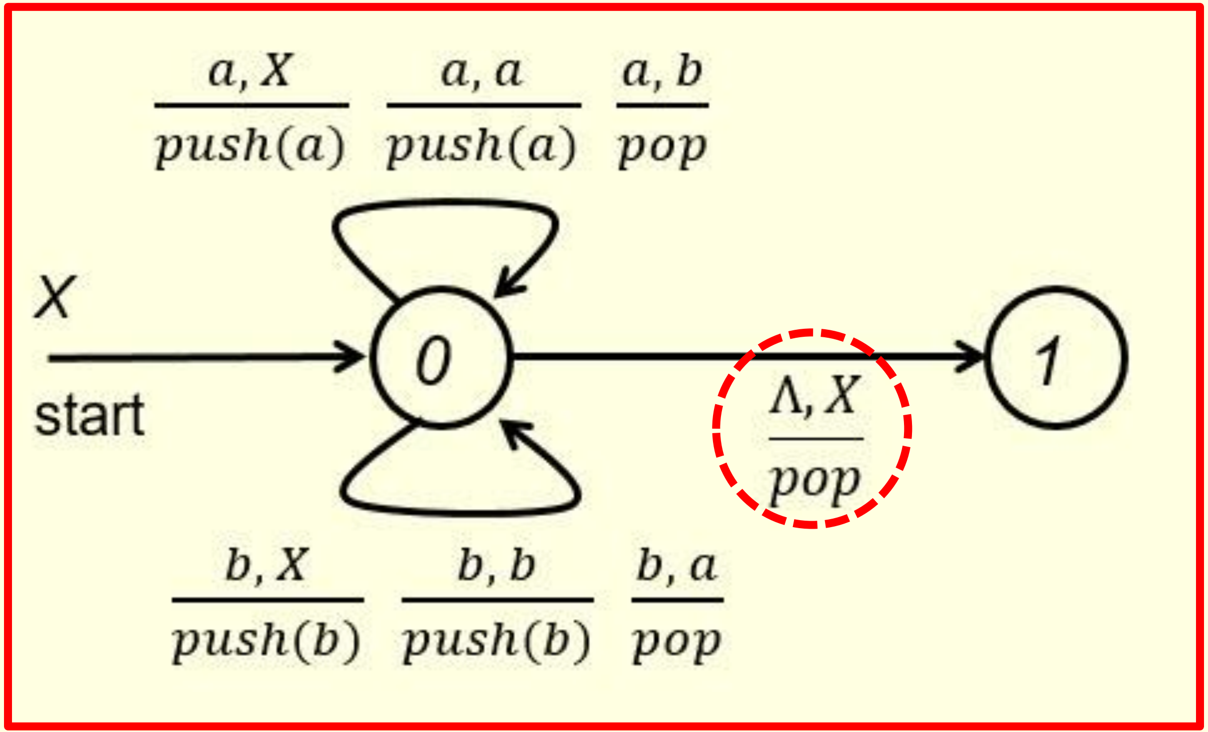


To accept Λ and to reach empty-stack status

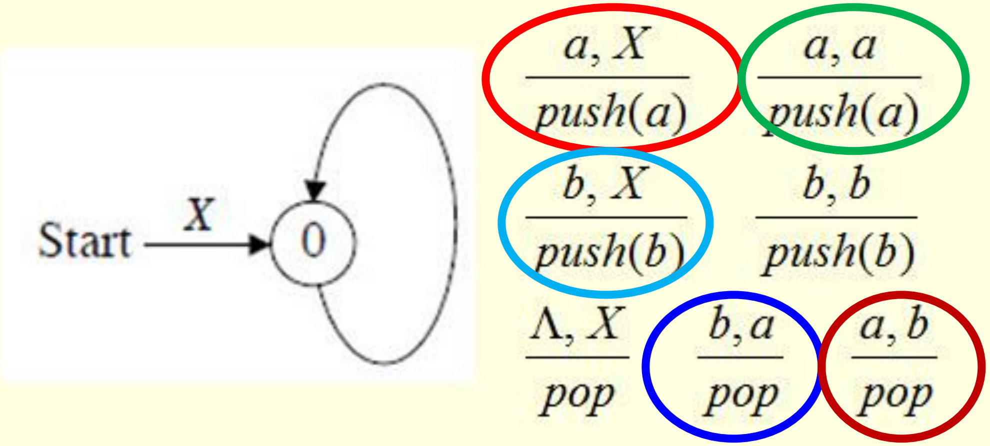
To ensure # of a's and # of b's are the same

Note the following two PDAs are equivalent:





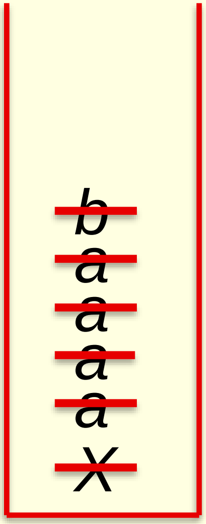
Solution: (1)



Consider: $aaabbabbba\Lambda$

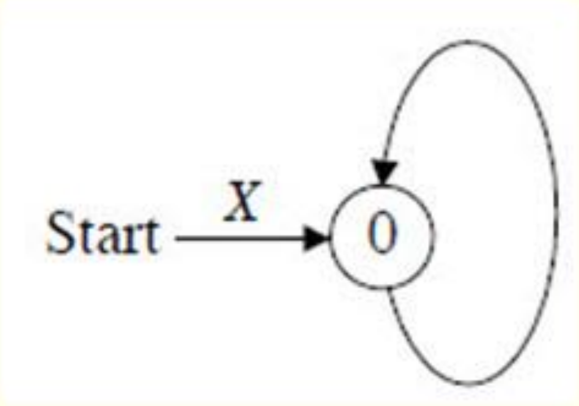
Accepted

Hence, the above PDA accepts L



Stack is empty

Solution: (2)



$\frac{a,\,X}{push(a)}$ $\frac{a,\,a}{push(a)}$

$\frac{b,\,X}{push(b)}$ $\frac{b,\,b}{push(b)}$

$\frac{\Lambda,\,X}{pop}$ $\frac{b,\,a}{pop}$ $\frac{a,\,b}{pop}$

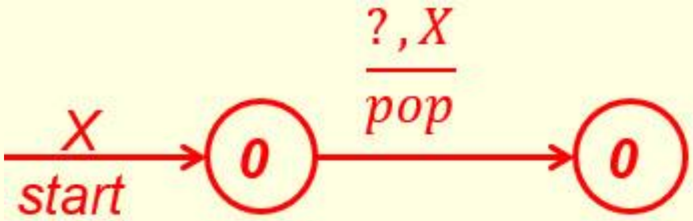
PDA Transformed into C - F grammar :

$$\underline{S \to X_{00}}$$

$$X_{00} \to \Lambda \mid aA_{00}X_{00} \mid bB_{00}X_{00}$$

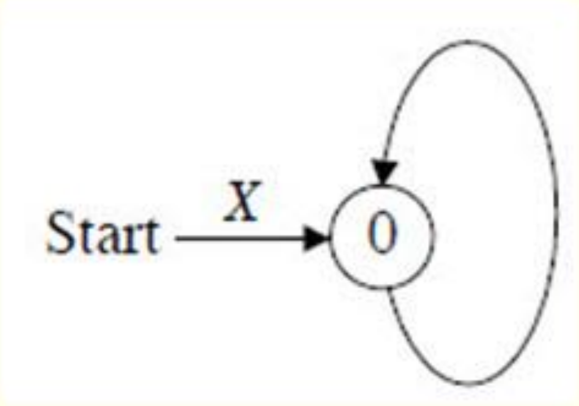
$$A_{00} \to b \mid aA_{00}A_{00}$$

$$B_{00} \to a \mid bB_{00}B_{00}$$



Type = ?

Solution: (2)



$$\frac{a,X}{push(a)} \quad \frac{a,a}{push(a)}$$

$$\frac{b,X}{push(b)} \quad \frac{b,b}{push(b)}$$

$$\frac{\Lambda,X}{pop} \quad \frac{b,a}{pop} \quad \frac{a,b}{pop}$$

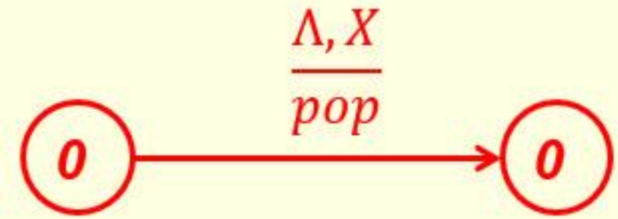
PDA Transformed into C - F grammar :

$$S \to X_{00}$$

$$\underline{X_{00} \to \Lambda} \mid aA_{00}X_{00} \mid bB_{00}X_{00}$$

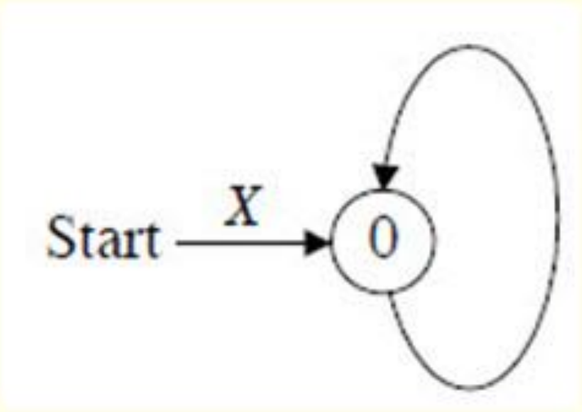
$$A_{00} \to b \mid aA_{00}A_{00}$$

$$B_{00} \to a \mid bB_{00}B_{00}$$



Type = ?

Solution: (2)



$$\frac{a, X}{push(a)} \quad \frac{a, a}{push(a)}$$

$$\frac{b, X}{push(b)} \quad \frac{b, b}{push(b)}$$

$$\frac{\Lambda, X}{pop} \quad \frac{b, a}{pop} \quad \frac{a, b}{pop}$$

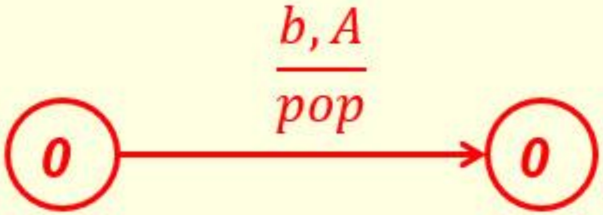
PDA Transformed into C - F grammar :

$$S \to X_{00}$$

$$X_{00} \to \Lambda \mid aA_{00}X_{00} \mid bB_{00}X_{00}$$

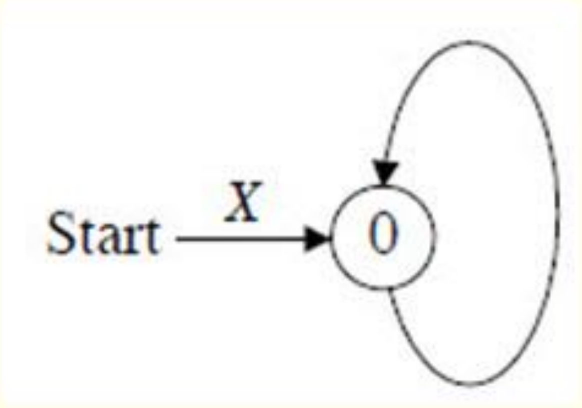
$$\underline{A_{00} \to b} \mid aA_{00}A_{00}$$

$$B_{00} \to a \mid bB_{00}B_{00}$$



Type = ?

Solution: (2)



$\frac{a, X}{push(a)}$ $\frac{a, a}{push(a)}$

$\frac{b, X}{push(b)}$ $\frac{b, b}{push(b)}$

$\frac{\Lambda, X}{pop}$ $\frac{b, a}{pop}$ $\frac{a, b}{pop}$

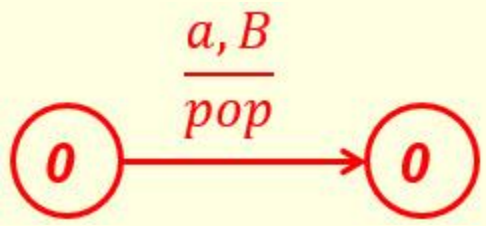
PDA Transformed into C - F grammar :

$$S \to X_{00}$$

$$X_{00} \to \Lambda \mid aA_{00}X_{00} \mid bB_{00}X_{00}$$

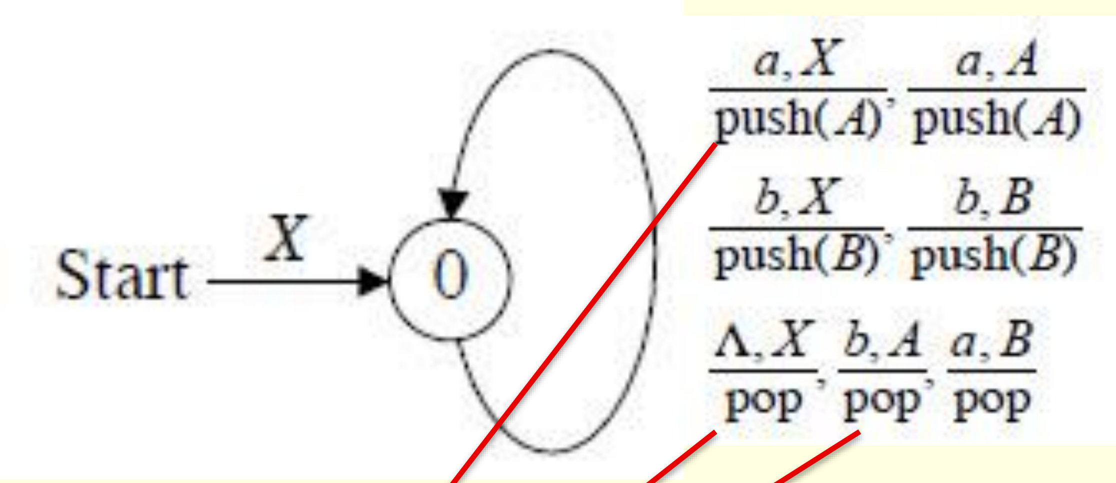
$$A_{00} \to b \mid aA_{00}A_{00}$$

$$\underline{B_{00} \to a} \mid bB_{00}B_{00}$$



Type = ?

Solution: (2)



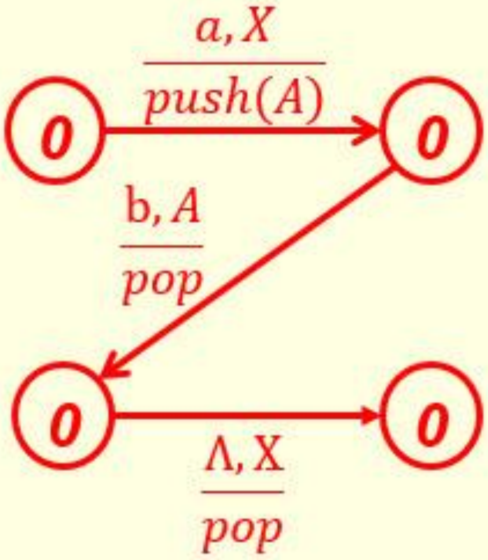
PDA Transformed into C - F grammar :

$$S \rightarrow X_{00}$$

$$\underline{X_{00}} \rightarrow \Lambda \mid \underline{aA_{00}X_{00}} \mid bB_{00}X_{00}$$

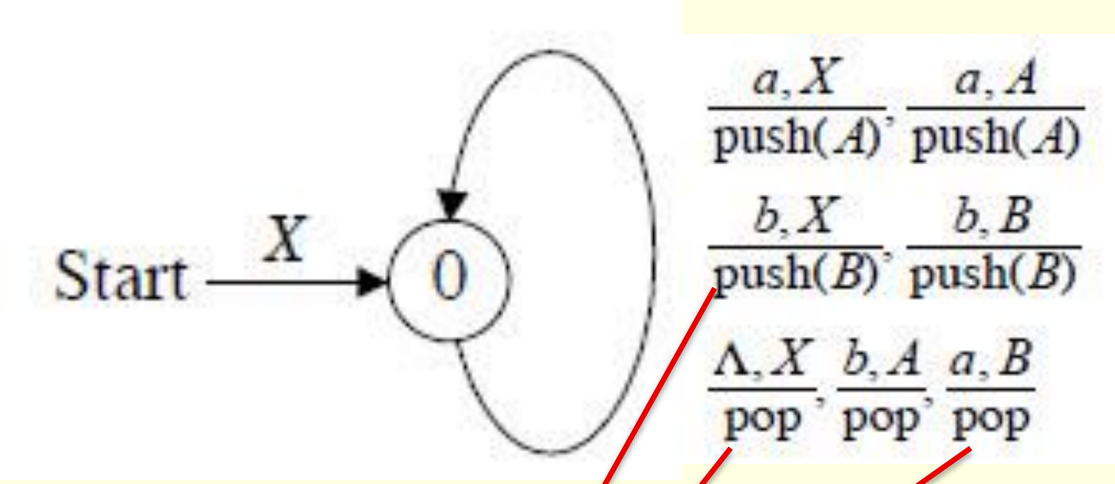
$$A_{00} \rightarrow b \mid aA_{00}A_{00}$$

$$B_{00} \rightarrow a \mid bB_{00}B_{00}$$



Type = ?

Solution: (2)



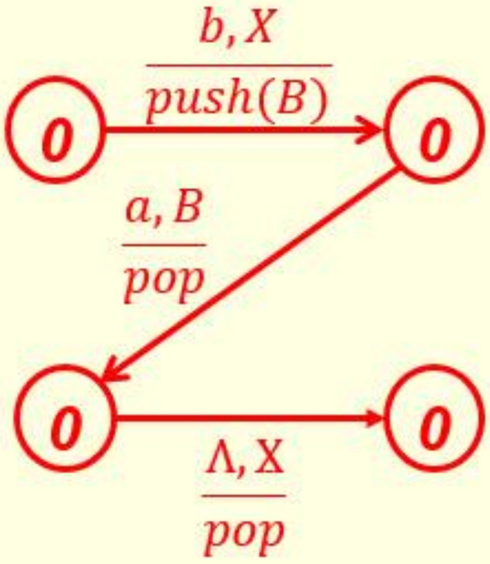
PDA Transformed into C - F grammar :

$$S \to X_{00}$$

$$\underline{X_{00}} \to \Lambda \mid aA_{00}X_{00} \mid \underline{bB_{00}X_{00}}$$

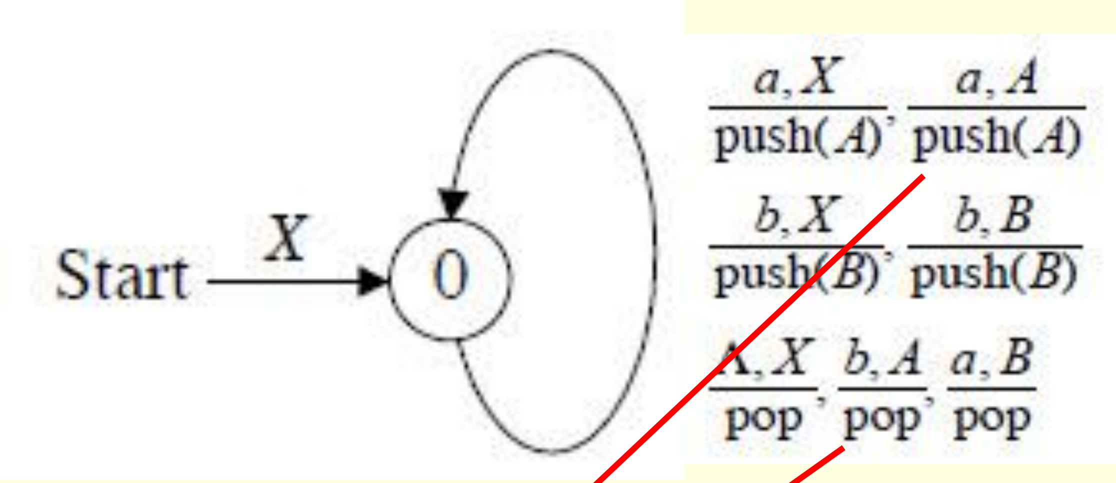
$$A_{00} \to b \mid aA_{00}A_{00}$$

$$B_{00} \to a \mid bB_{00}B_{00}$$



Type = ?

Solution: (2)



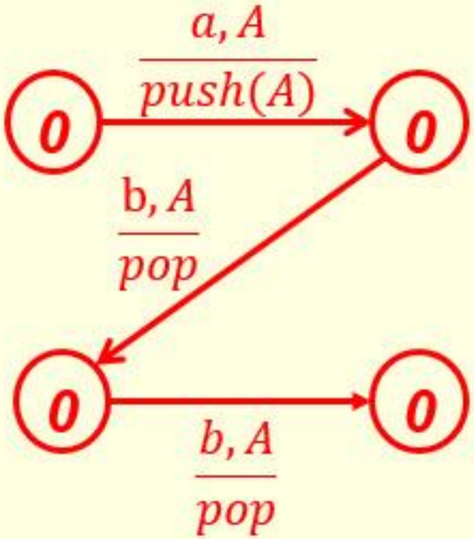
PDA Transformed into C-F grammar :

$$S \to X_{00}$$

$$X_{00} \to \Lambda \mid aA_{00}X_{00} \mid bB_{00}X_{00}$$

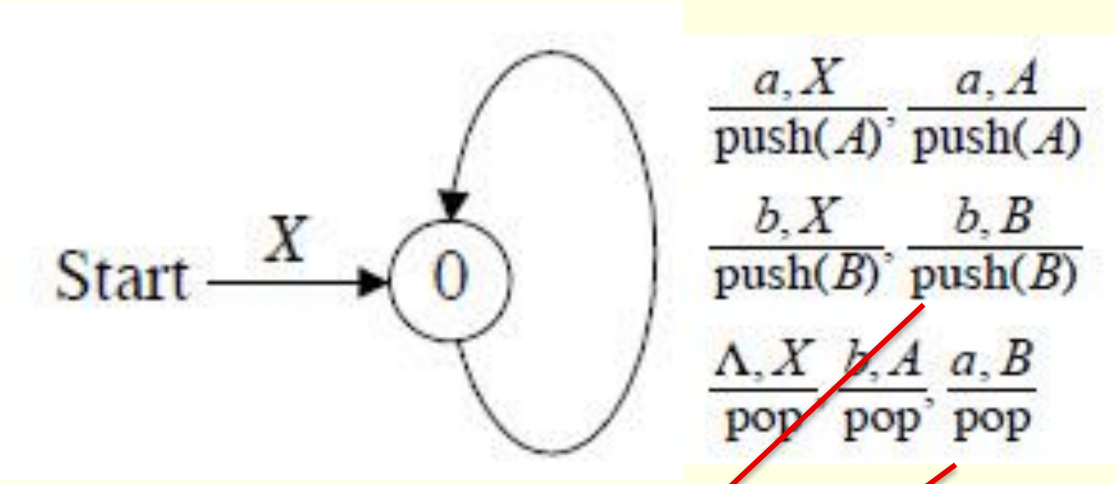
$$\underline{A_{00}} \to b \mid \underline{aA_{00}A_{00}}$$

$$B_{00} \to a \mid bB_{00}B_{00}$$



Type = ?

Solution: (2)



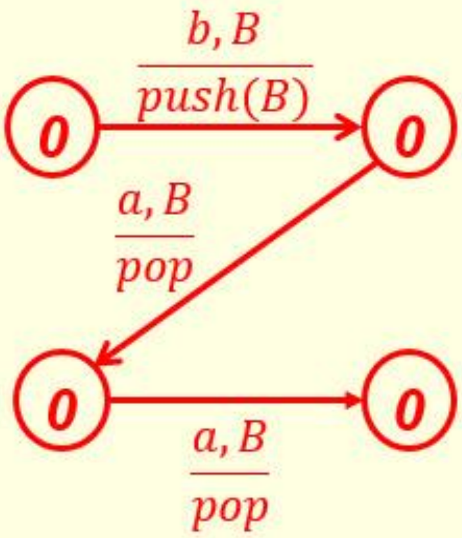
PDA Transformed into C - F grammar :

$$S \rightarrow X_{00}$$

$$X_{00} \rightarrow \Lambda \mid aA_{00}X_{00} \mid bB_{00}X_{00}$$

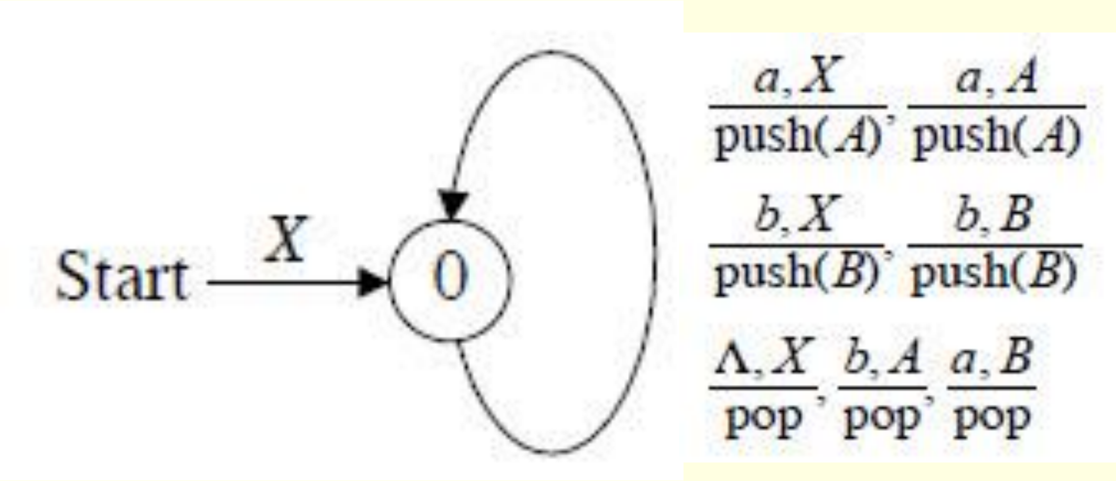
$$A_{00} \rightarrow b \mid aA_{00}A_{00}$$

$$\underline{B_{00}} \rightarrow a \mid \underline{bB_{00}B_{00}}$$



Type = ?

Solution: (2)



PDA Transformed into C - F grammar :

$$S \to X_{00}$$

$$X_{00} \to \Lambda \mid aA_{00}X_{00} \mid bB_{00}X_{00}$$

$$A_{00} \to b \mid aA_{00}A_{00}$$

$$B_{00} \to a \mid bB_{00}B_{00}$$

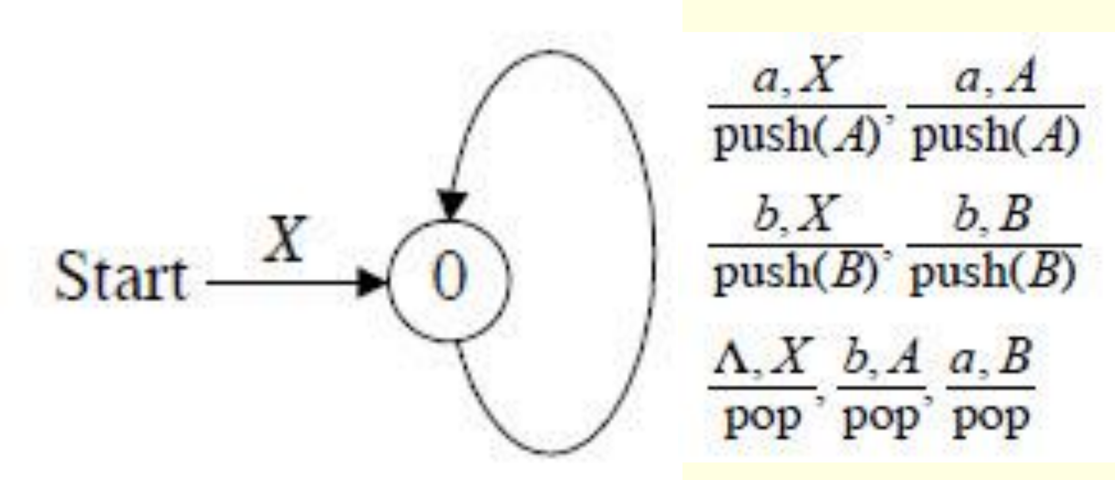
Simplified CFG :

$$S \to \Lambda \mid aAS \mid bBS$$

$$A \to b \mid aAA$$

$$B \to a \mid bBB$$

Solution: (2)



Simplified CFG:

$$S \rightarrow \underline{\Lambda} \,|\, \underline{aAS} \,|\, bBS$$

$$A \rightarrow \underline{b} \,|\, \underline{aAA}$$

$$B \rightarrow a \,|\, bBB$$

Derivation of $\boxed{aababb}$:

$$S \Rightarrow \underline{aAS} \Rightarrow \underline{aaAAS} \Rightarrow \underline{aabAS}$$

$$\Rightarrow \underline{aabaAAS} \Rightarrow \underline{aababAS}$$

$$\Rightarrow \underline{aababbS} \Rightarrow \underline{aababb}$$

Nondeterministic PDAs are **more** powerful than deterministic PDAs.

Left half and right half are symmetric

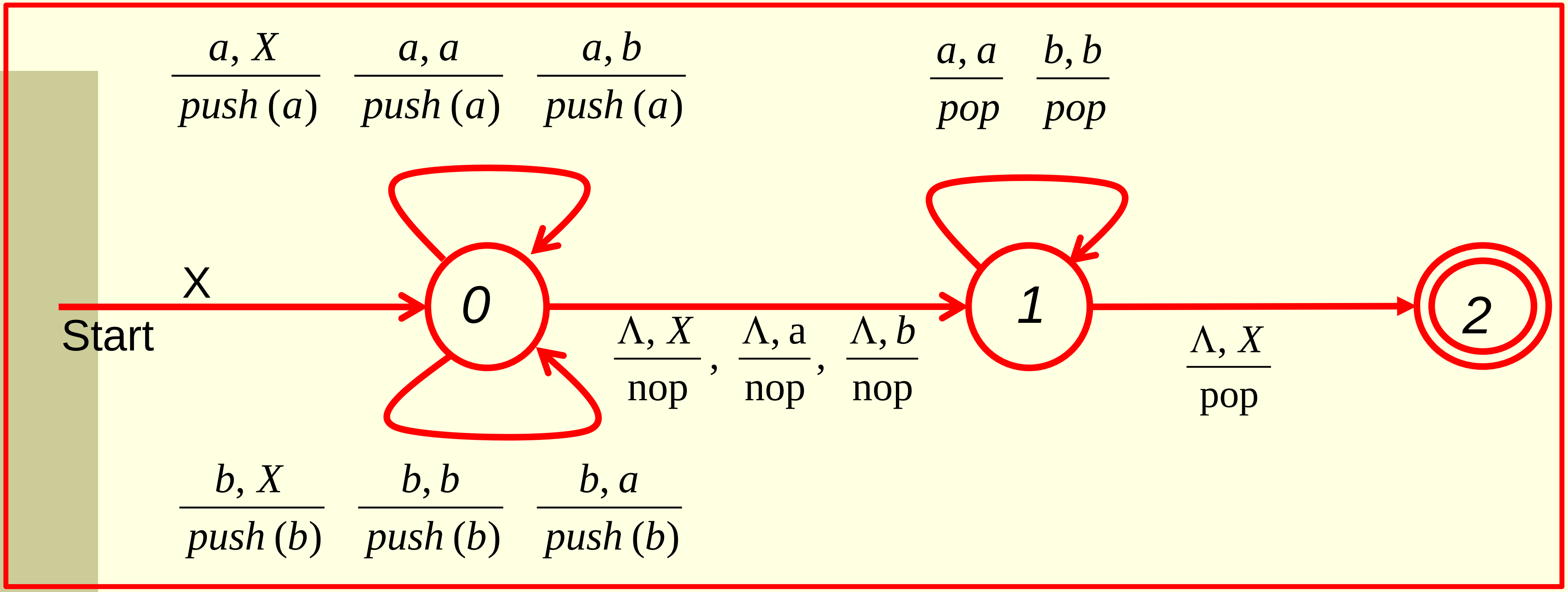
An example is to consider the language of even Palindromes (such as: aababbbbabaa) over {*a*, *b*}.

A context-free grammar for the language is given by

$$S \rightarrow \Lambda \mid aSa \mid bSb$$

Any PDA to accept the language must make a nondeterministic decision to start comparing the 2nd half of a string with the reverse of the first half.

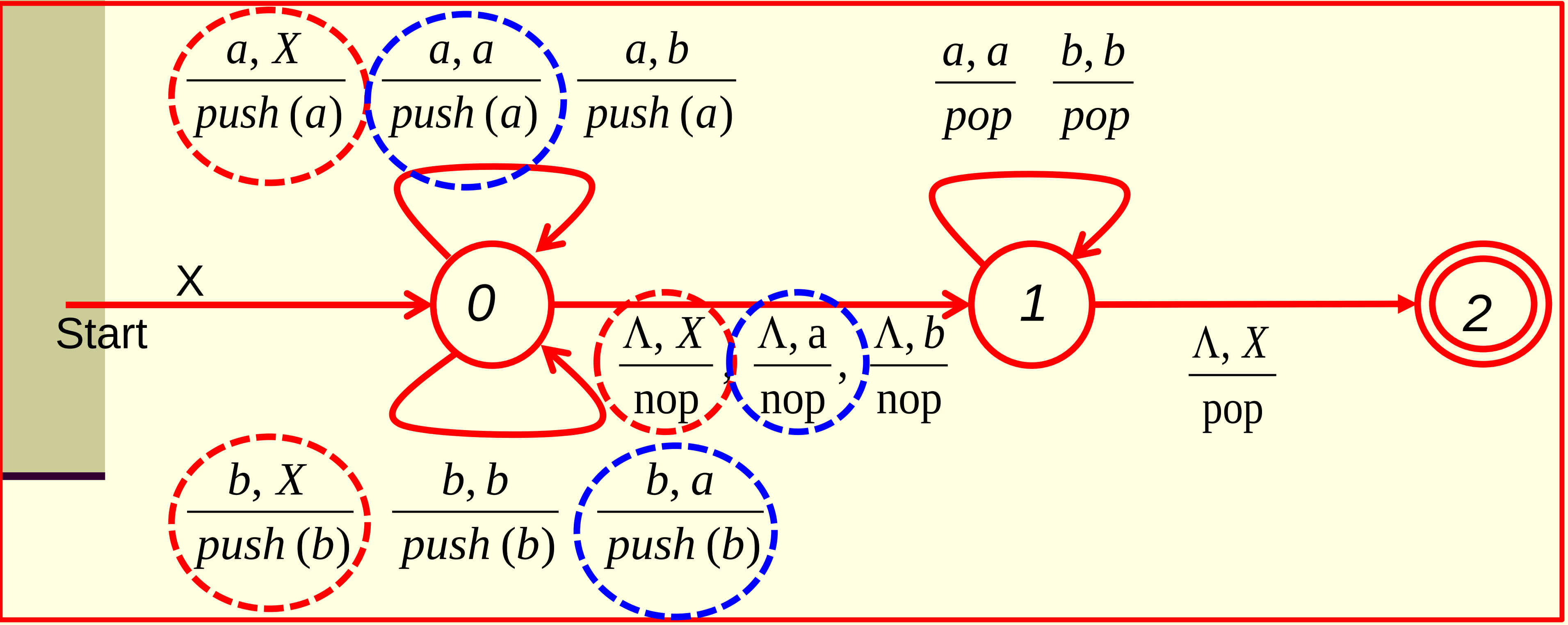
Example: consider the following PDA



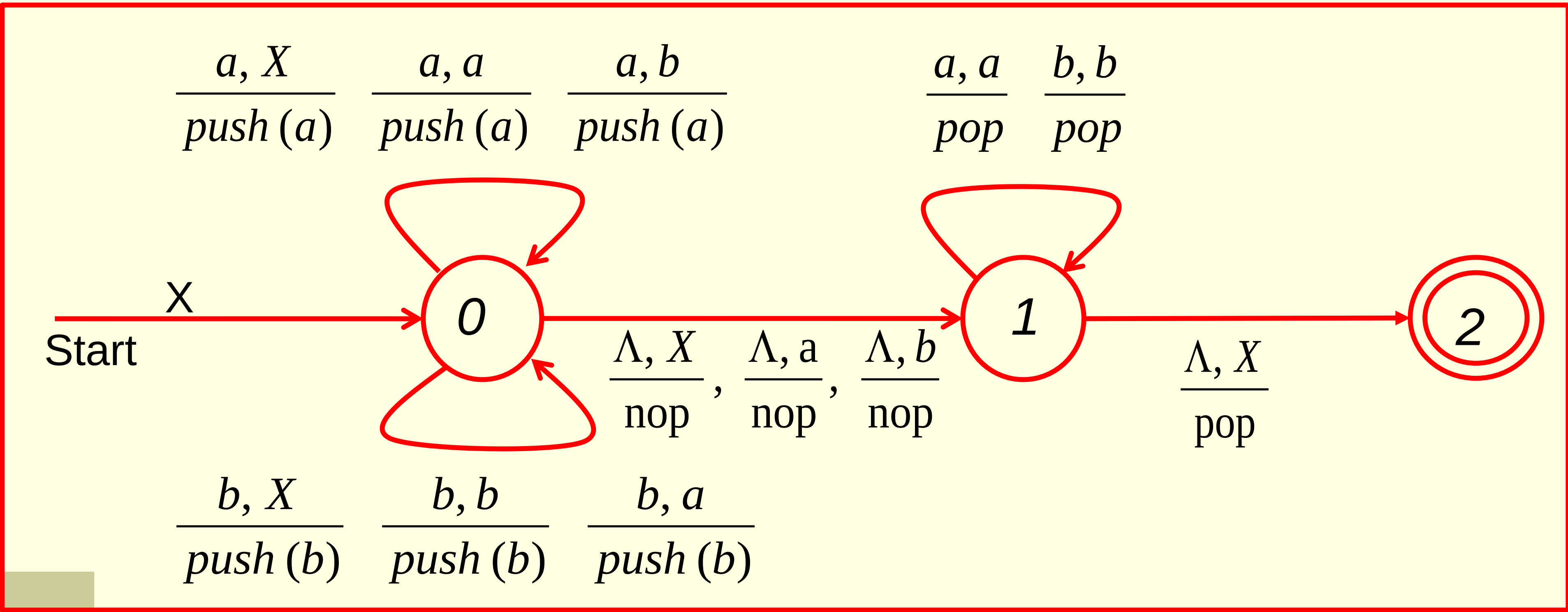
This non-deterministic PDA accepts the language of even palindromes over {a, b}

Example: consider the following PDA

Why is this a non-deterministic PDA ?



Does it accept even palindromes over {a, b} ?

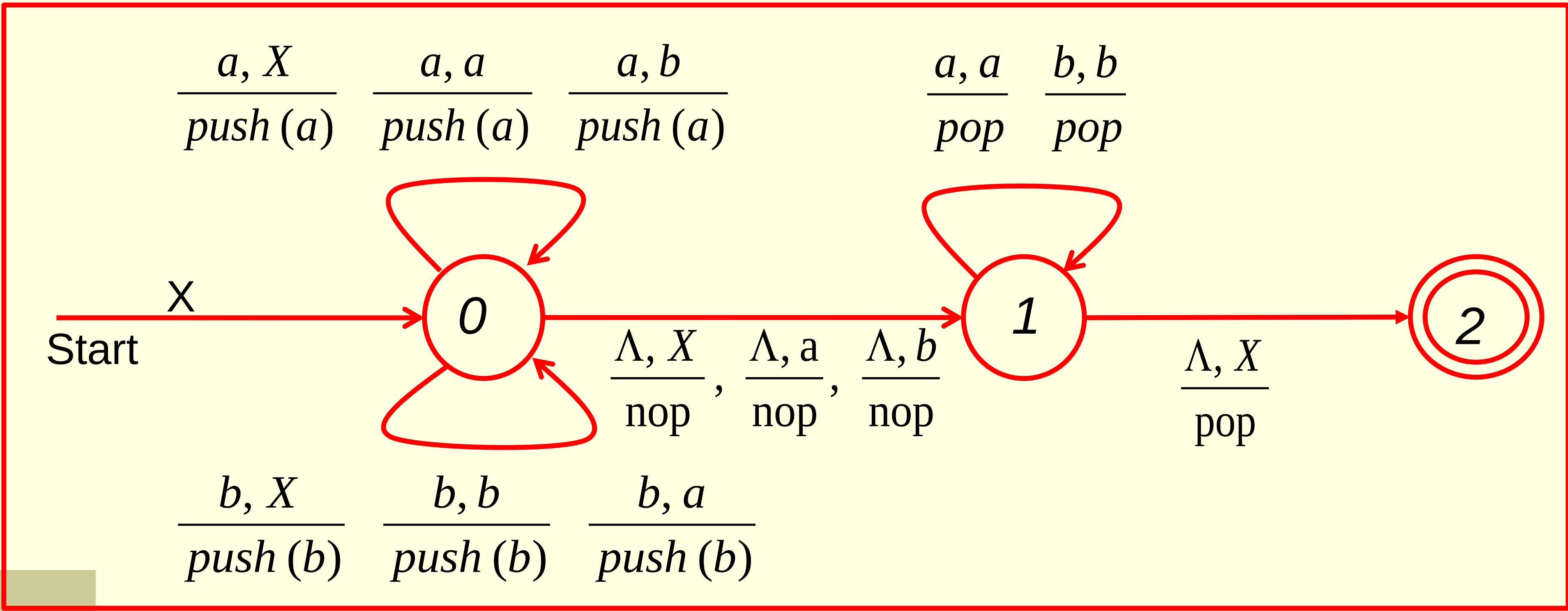


Consider (0, abbbba, X)

⇒ (0, bbbba, aX)
⇒ (0, bbba, baX)
⇒ (0, bba, bbaX)
⇒ (0, Λbba, bbaX)
⇒ (1, bba, bbaX)
⇒ (1, ba, baX) ⇒ (1, a, aX)
⇒ (1, Λ, X) ⇒ (2, Λ, Λ)

If we make a guess here

The PDA may guess the middle wrong:



Consider (0, abbbba, X)

⇒ (0, bbbba, aX)

⇒ (0, bbba, baX)

⇒ (0, Λbbba, baX)

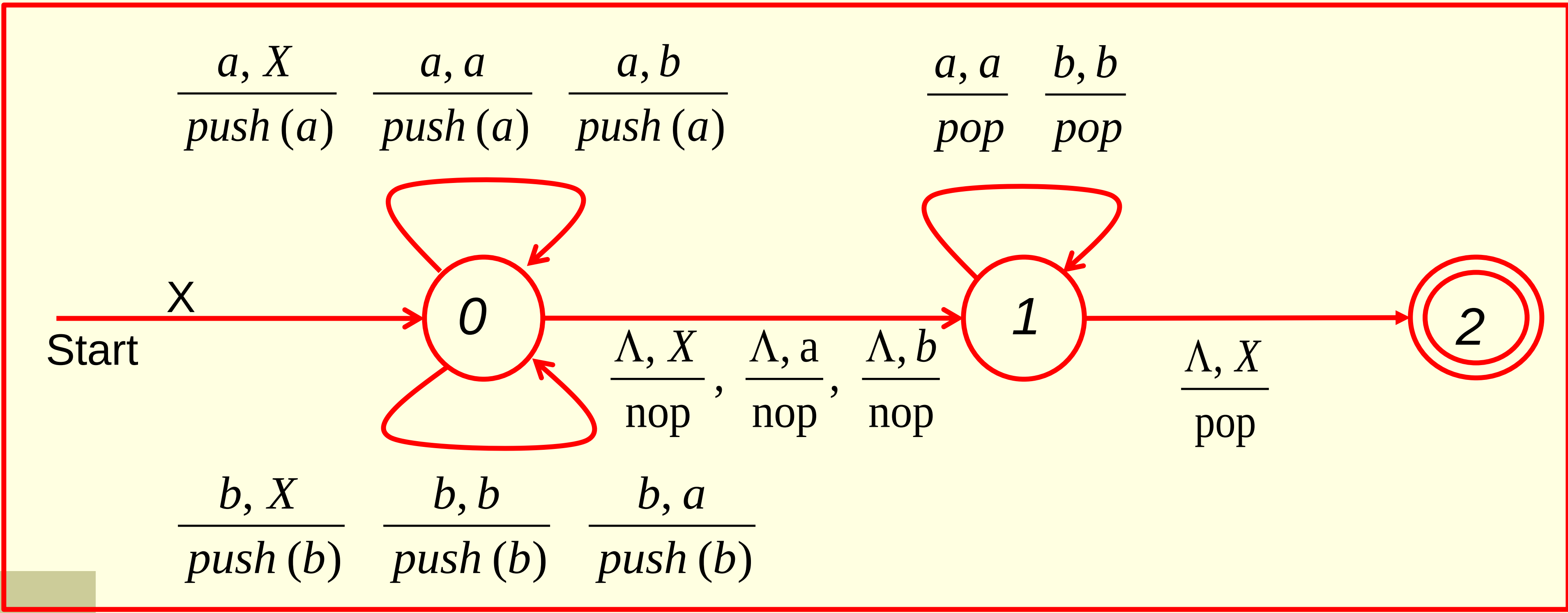
⇒ (1, bbba, baX)

⇒ (1, bba, aX)

If the PDA made a guess here

It gets stuck here!

This PDA can only accept even palindromes:



Consider (0, aabbba, X)

⇒ (0, abbba, aX)
⇒ (0, bbba, aaX)
⇒ (0, bba, baaX)
⇒ (0, ba, bbaaX)
⇒ (0, a, bbbaaX)
⇒ (0, Λ, abbbaaX)
⇒ (1, Λ, abbbaaX)

It gets stuck here!

There is in general no way to translate a non-deterministic PDA (NPDA) into a deterministic one.

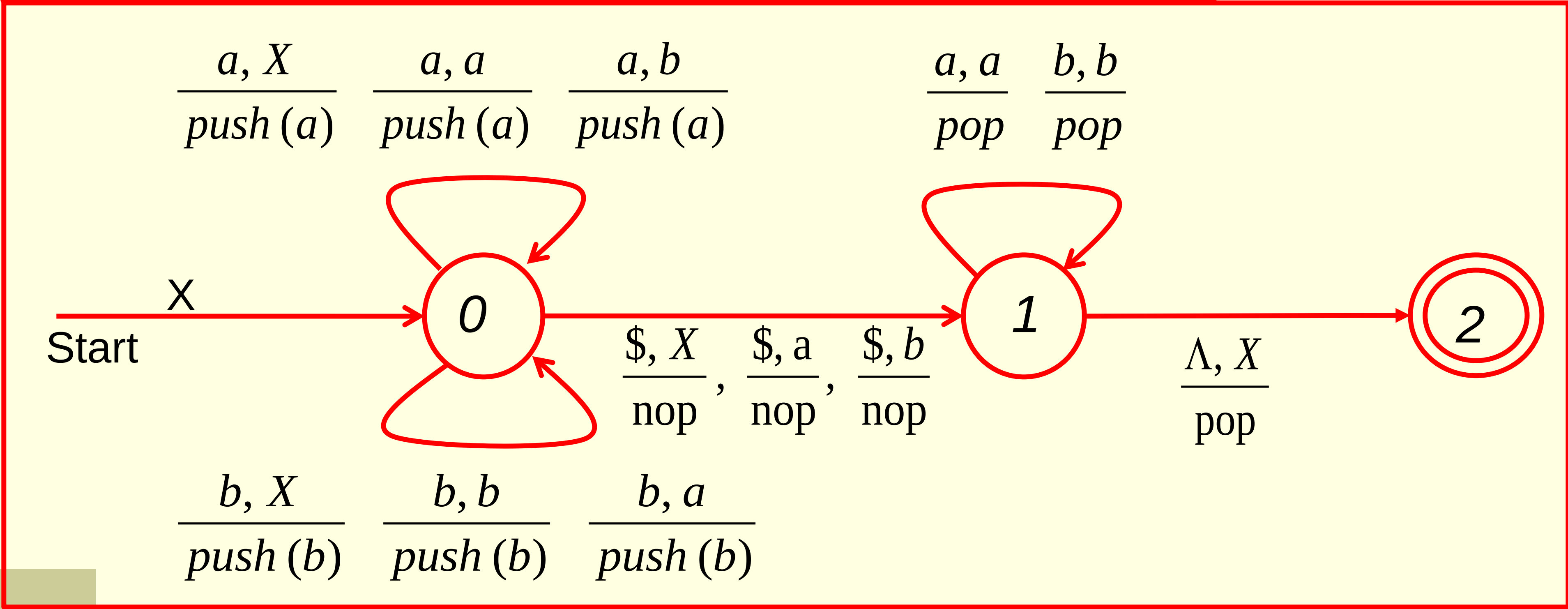
Indeed, there is no DPDA which recognizes the language of even palindromes.

That is why we can say that NPDAs are more powerful than DPDAs.

However, we can define a similar language L_1 over $\{a, b, \$\}$ which can be recognized by a DPDA:

$$L_1 = \{ w\$w^R \mid w \in \{a, b\}^* \}$$

A DPDA for L1:



Consider (0, abb$bba, X)

⇒ (0, bb$bba, aX)
⇒ (0, b$bba, baX)
⇒ (0, $bba, bbaX)
⇒ (1, bba, bbaX)
⇒ (1, ba, baX)
⇒ (1, a, aX)
⇒ (1, Λ, X) ⇒ (2, Λ, Λ)

What do you see here?

Is this PDA indeed deterministic?

Note that

1. Final-state acceptance and empty-stack acceptance are equivalent only for NPDAs

2. Final-state acceptance and empty-stack acceptance are not equivalent for DPDAs. For DPDAs, the class of languages defined by final-state acceptance is bigger.

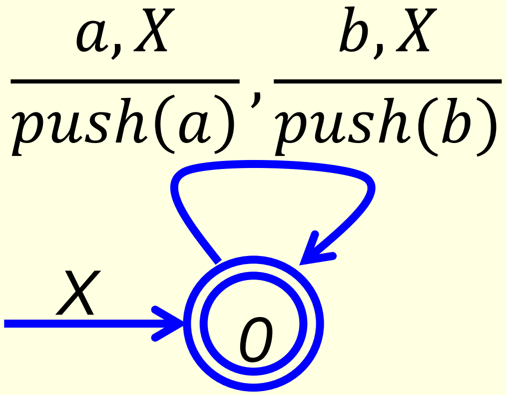
Why?

b/c DPDAs do not have the instruction $\frac{?, X}{pop}$ *in most of the cases*

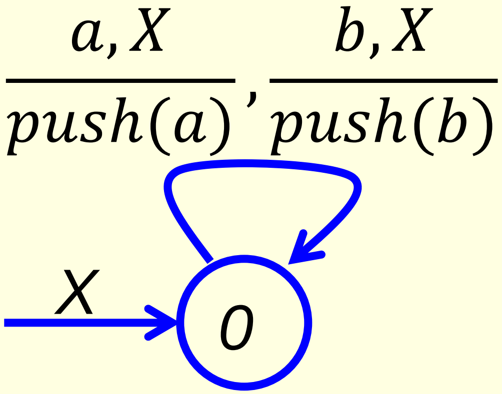
(without the instruction Λ,X/pop,we can still do final state acceptance, but we will not be able to do empty stack acceptance.)

Why?

Consider the following example:



Accepts ? { Λ, a, b}

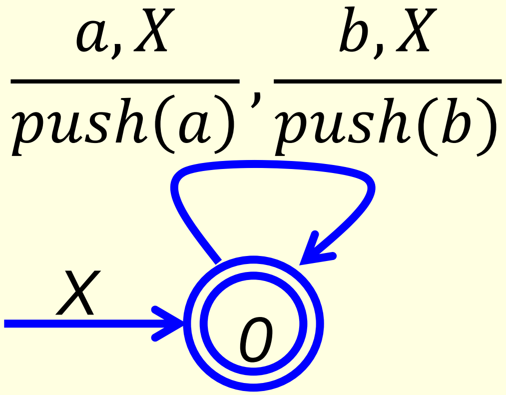


Accepts ? Φ

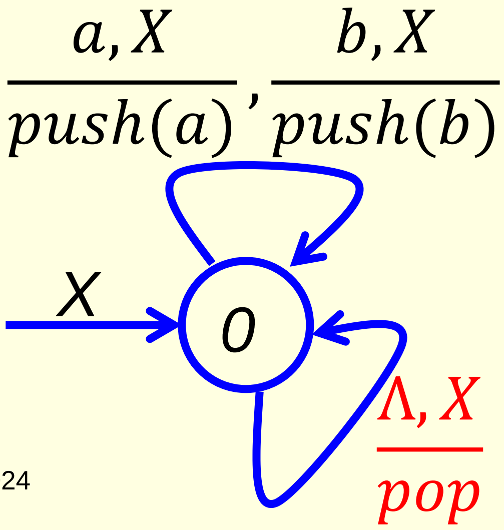
Not even Λ

Why?

Consider the following example:



Accepts ? { Λ, a, b}

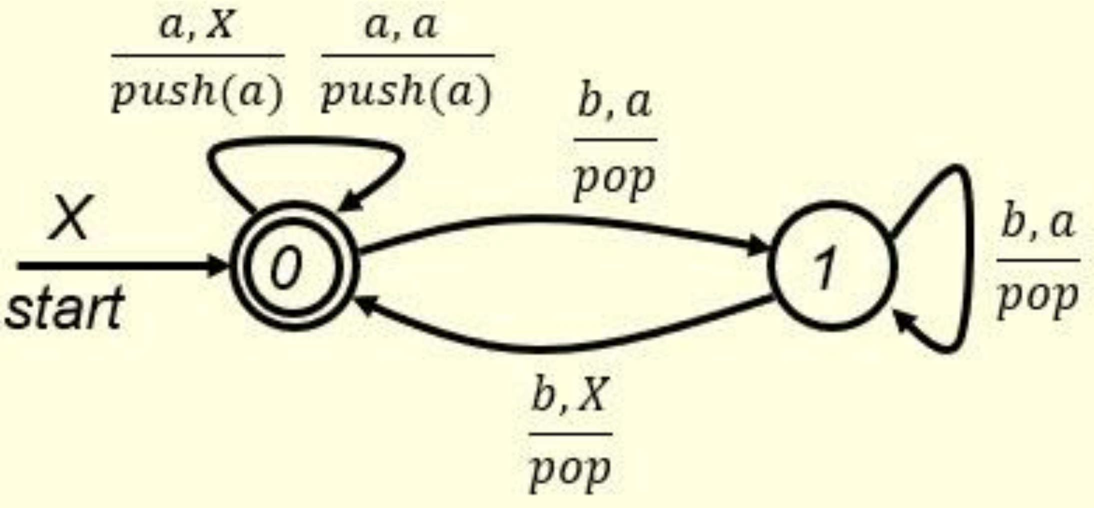


Accepts ? Λ

Not DPDA

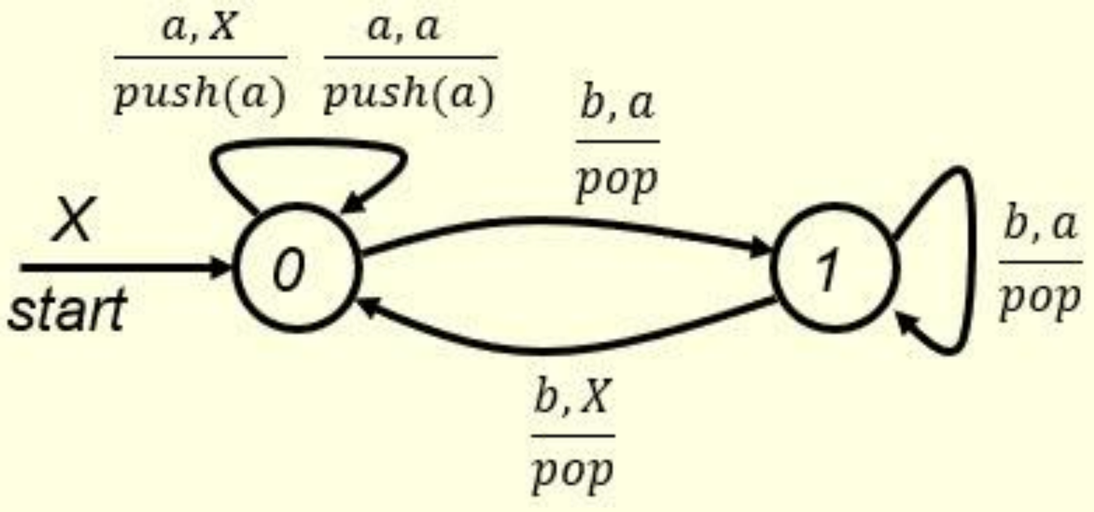
Why?

Or, consider the following example:



Accepts ?

Λ, aa, abb



Accepts ?

abb

Summarize:

- CFGs and PDAs have equivalent expressive powers. More formally, . . .

- Theorem. For every CFG G, there is a PDA P such that L(G) = L(P).

  In addition, for every PDA P, there is a CFG G such that L(P) = L(G).

  Thus, L is CF iff there is a (non-deterministic) PDA P such that L = L(P).

- CF languages are exactly those languages that are accepted by (non-deterministic) PDAs.

L is CF iff there is a (non-deterministic) PDA P such that L = L(P).

CF languages are exactly those languages that are accepted by (non-deterministic) PDAs.

Why?

If a CFL is infinite, it would have a non-trivial grammar (the right hand side of at least one production would contain a non-terminal and often time a recursive non-terminal). For such a grammar, when you convert it to a PDA, you would get a one-state, non-deterministic PDA. For instance, convert the following simple cases and see what would you get.

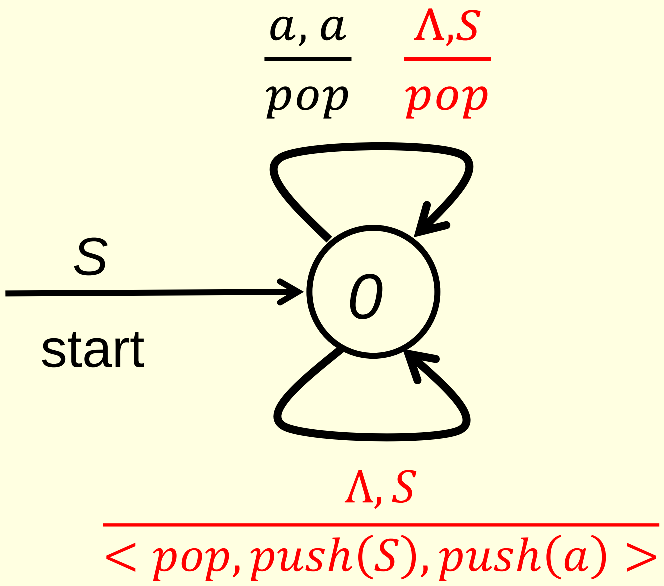
$$S \to aS \mid \Lambda$$

or

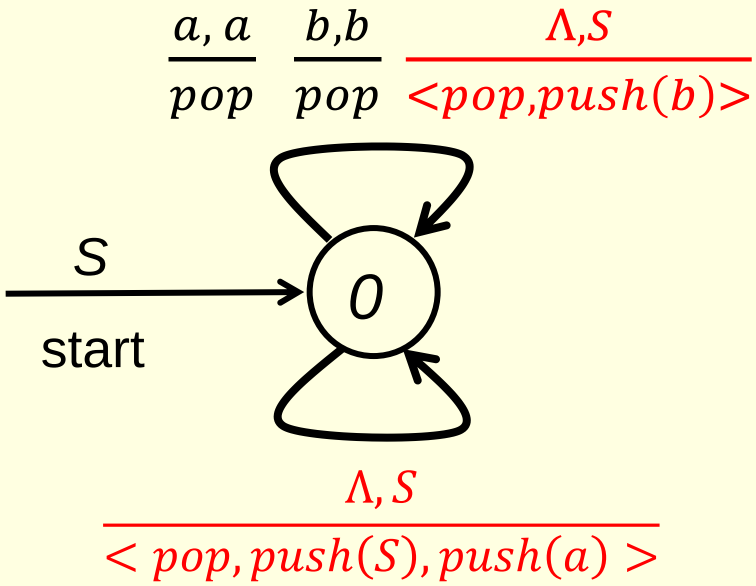
$$S \to aS \mid b$$

Why?

For { S → aS | Λ }



For $\{ S \to aS \mid b \}$



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

S

start

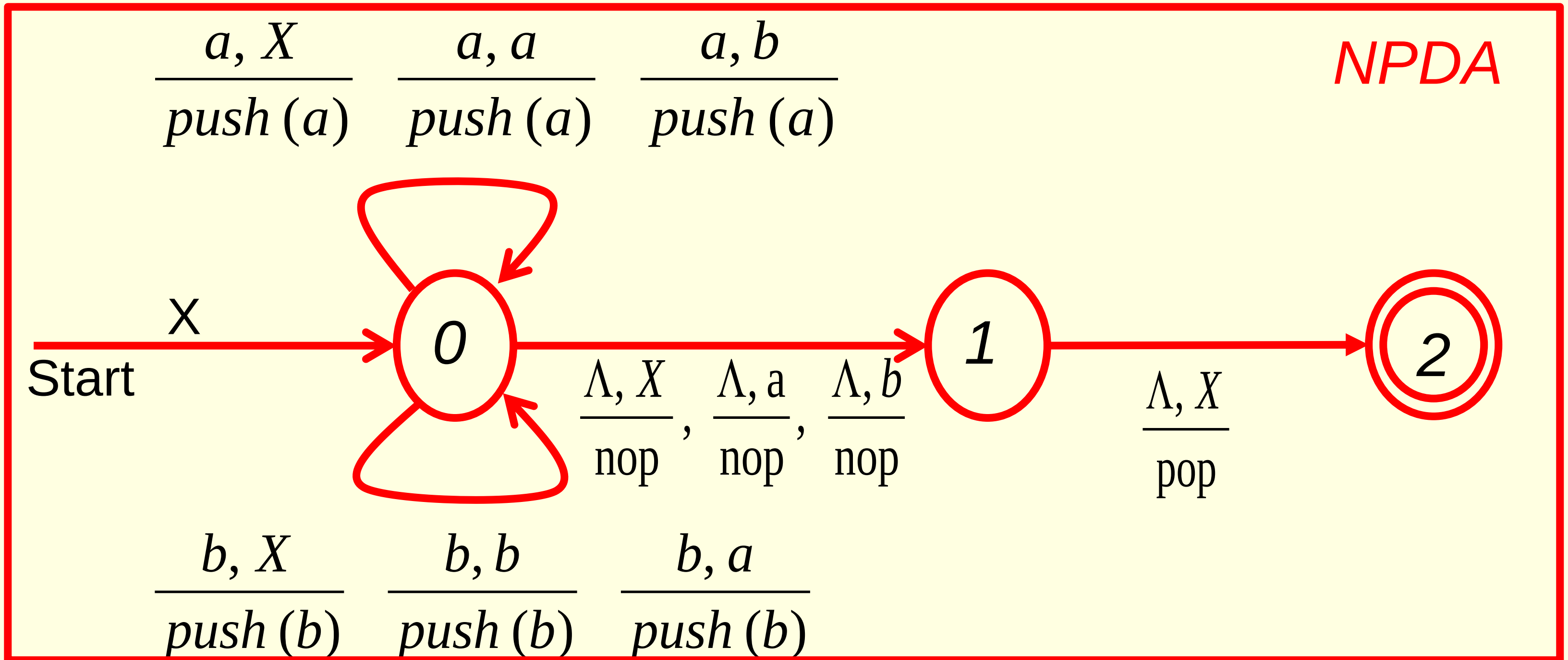
0

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

Summarize:

- A CF language is called a *deterministic final-state CF language* if it can be recognized by a deterministic final-state PDA

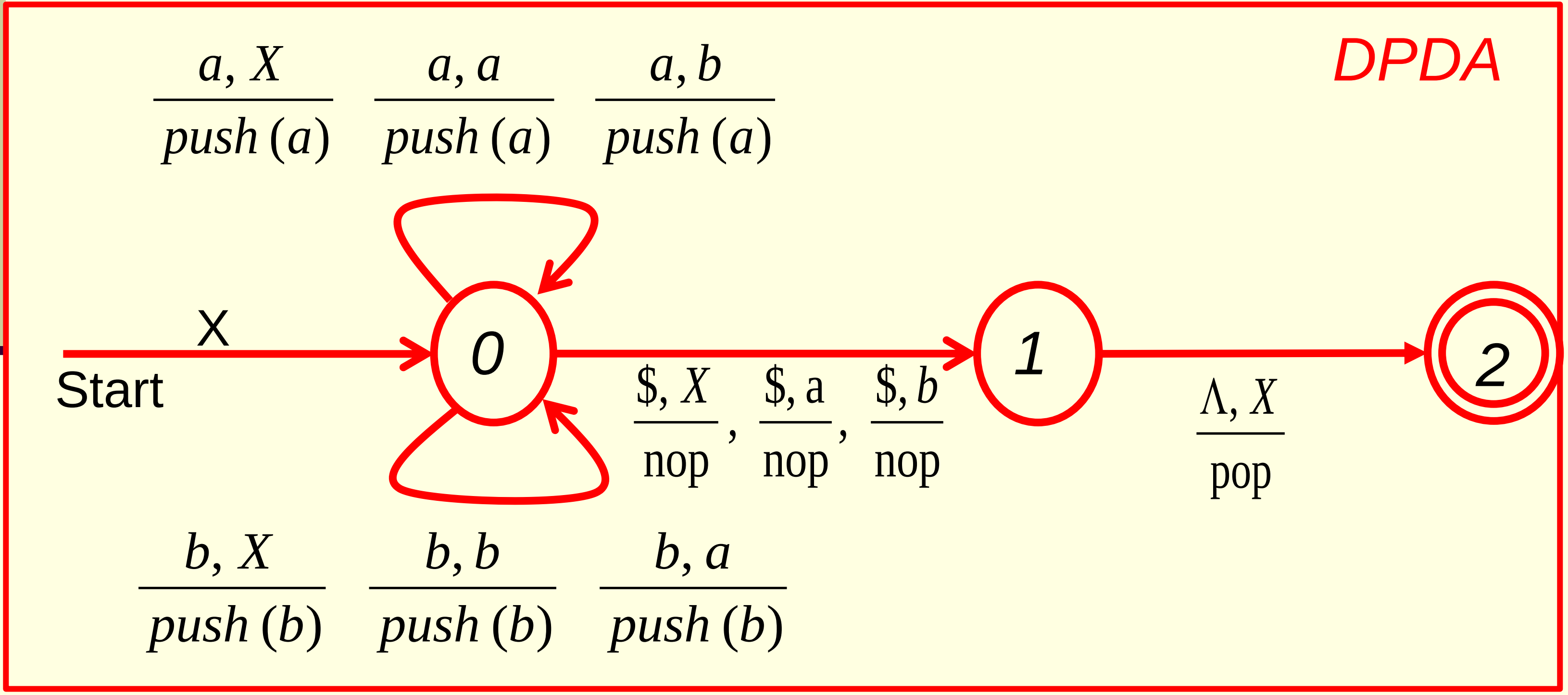
- Even palindromes: not a deterministic final-state CFL



Summarize:

L_1 over $\{a, b, \$\}$ is a deterministic final-state CFL:

$$L_1 = \{\, w\$w^R \mid w \in \{a, b\}^* \,\}$$



End of Context-Free Language and Pushdown Automata II