Intro to Theory of Computation

CS 464

LECTURE 7

Last time:

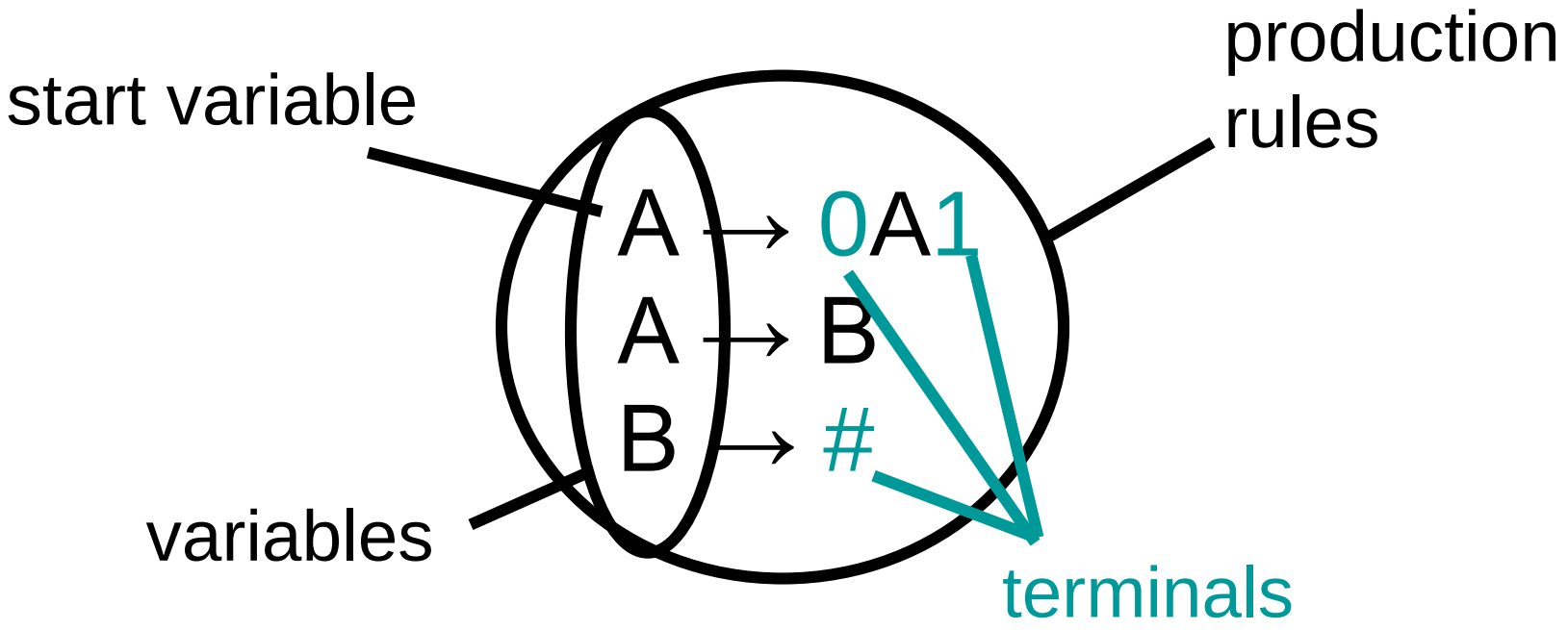
- Proving a language is not regular
- Pushdown automata (PDAs)

Today:

- Context-free grammars (CFG)
- Equivalence of CFGs and PDAs

Sofya Raskhodnikova

CONTEXT-FREE GRAMMARS (CFGs)



$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 00B11 \Rightarrow 00\#11$$

VALLEY GIRL GRAMMAR

<PHRASE> → <FILLER><PHRASE>

<PHRASE> → <START><END>

<FILLER> → LIKE

<FILLER> → UMM

<START> → YOU KNOW

<START> → ε

<END> → GAG ME WITH A SPOON

<END> → AS IF

<END> → WHATEVER

<END> → LOSER

CS 464

VALLEY GIRL GRAMMAR

<PHRASE> → <FILLER><PHRASE> | <START><END>

<FILLER> → LIKE | UMM

<START> → YOU KNOW | ε

<END> → GAG ME WITH A SPOON | AS IF | WHATEVER | LOSER

Formal Definition

A ***CFG*** is a 4-tuple **G = (V, Σ, R, S)**

V is a finite set of variables

Σ is a finite set of terminals (disjoint from V)

R is set of production rules of the form $A \rightarrow W$, where $A \in V$ and $W \in (V \cup \Sigma)^*$

$S \in V$ is the start variable

L(G) is the set of strings generated by G

A language is context-free if it is generated by some CFG.

Formal Definition

A ***CFG*** is a 4-tuple **G = (V, Σ, R, S)**

V is a finite set of variables

Σ is a finite set of terminals (disjoint from V)

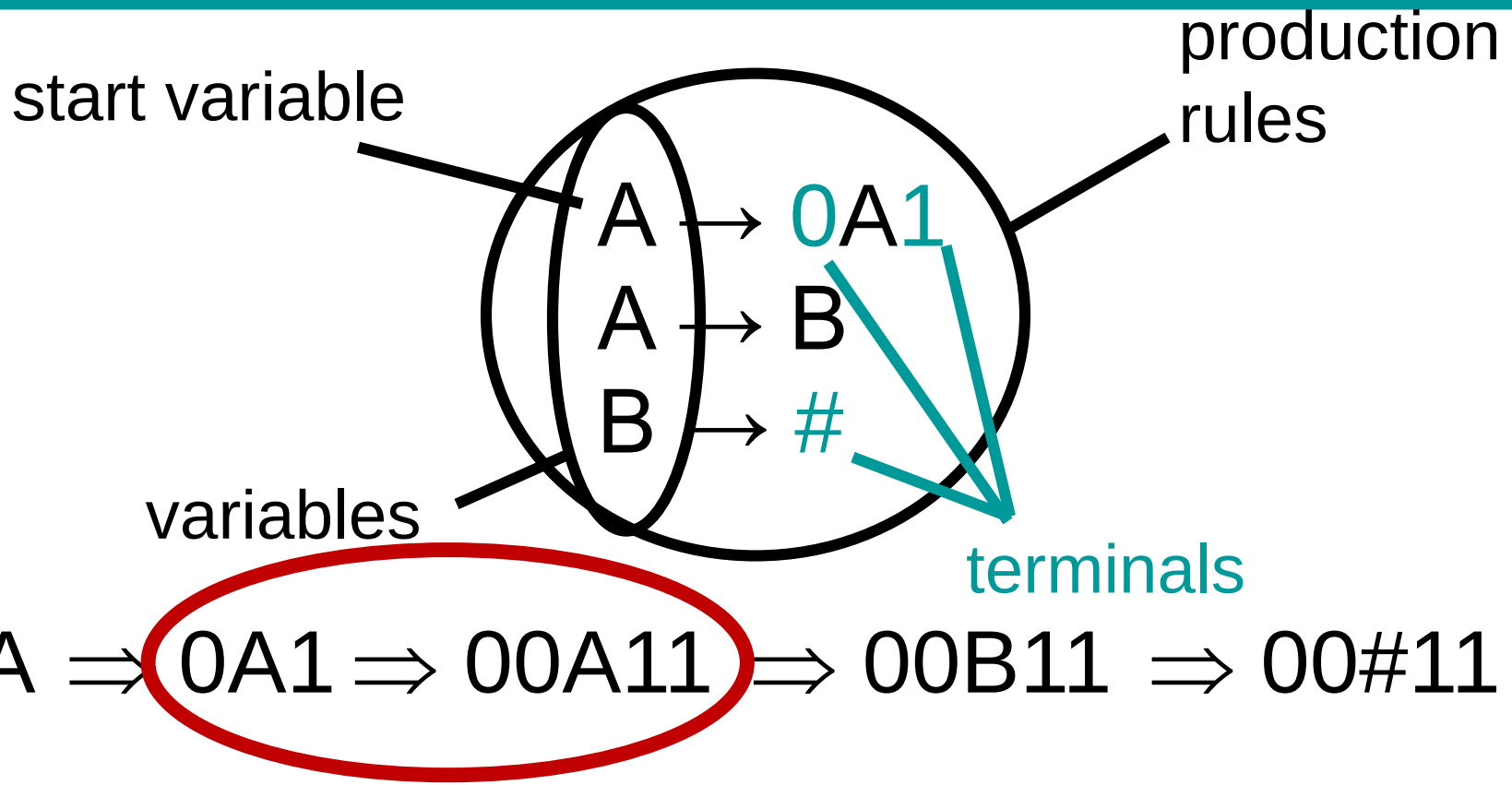
R is set of production rules of the form $A \rightarrow W$, where $A \in V$ and $W \in (V \cup \Sigma)^*$

$S \in V$ is the start variable

Example: $G = \{\{S\}, \{0,1\}, R, S\}$ $R = \{ S \rightarrow 0S1, S \rightarrow \varepsilon \}$

$$L(G) = \{ 0^n 1^n \mid n \geq 0 \}$$

CFG terminology



uVw yields uvw if $(V \rightarrow v) \in R$.

A derives 00#11 in 4 steps.

Example

GIVE A CFG FOR EVEN-LENGTH PALINDROMES

$$S \rightarrow \sigma S \sigma \text{ for all } \sigma \in \Sigma$$

$$S \rightarrow \varepsilon$$

CS 464

Example

GIVE A CFG FOR THE EMPTY SET

$$G = \{ \{S\}, \Sigma, \varnothing, S \}$$

GIVE A CFG FOR…

L_3 = { strings of balanced parens }

$L_4 = \{ a^i b^j c^k \mid i, j, k \geq 0 \text{ and } (i = j \text{ or } j = k) \}$

CS 464

CFGs in the real world

The syntactic grammar for the Java programming language

BasicForStatement:

for *(; ;) Statement*
for *(; ; ForUpdate) Statement*
for *(; Expression ;) Statement*
for *(; Expression ; ForUpdate) Statement*
for *(ForInit ; ;) Statement*
for *(ForInit ; ; ForUpdate) Statement*
for *(ForInit ; Expression ;) Statement*
for *(ForInit ; Expression ; ForUpdate) Statement*

COMPILER MODULES

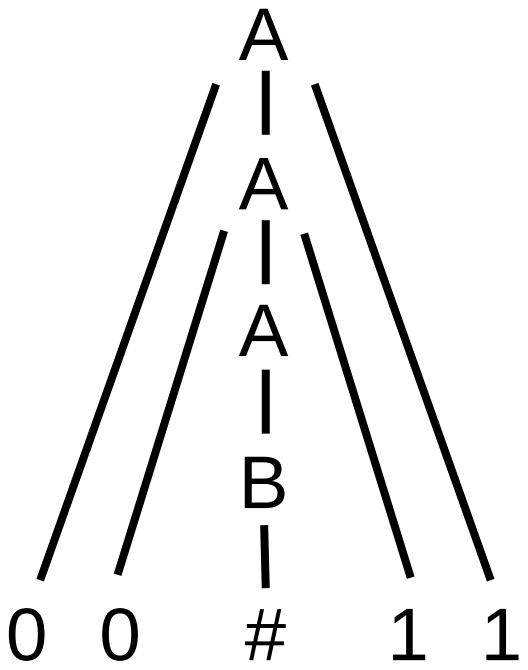
LEXER

PARSER

SEMANTIC ANALYZER

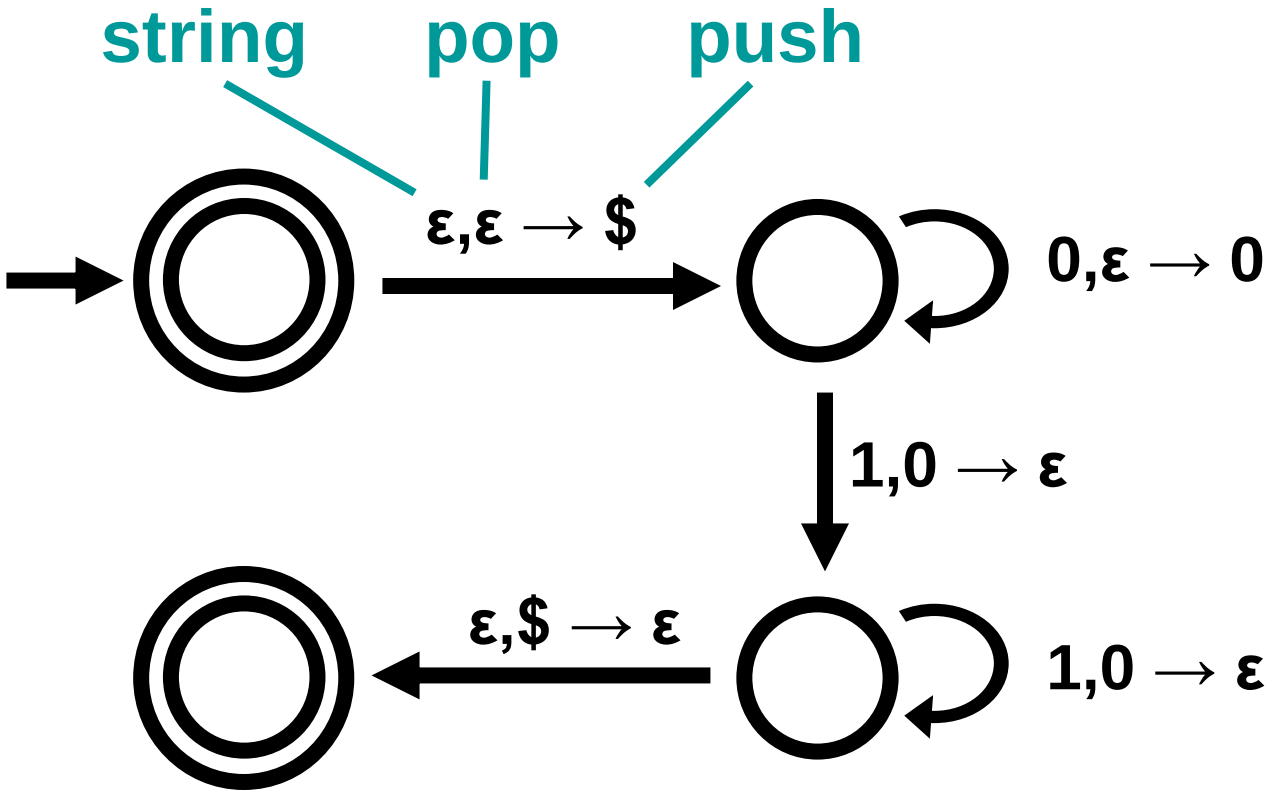
TRANSLATOR/INTERPRETER

Parse trees



$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 00B11 \Rightarrow 00\#11$$

PDAs: reminder

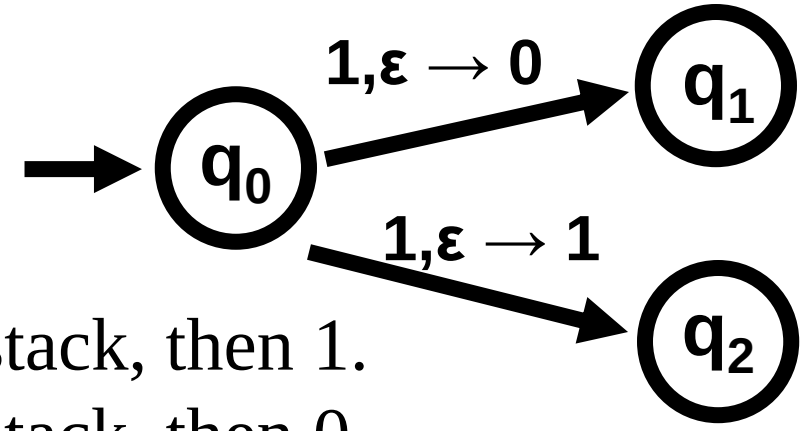


The language of P is the set of strings it accepts.
PDAs are nondeterministic.

CS 464

I-clicker problem (frequency: AC)

When PDA takes a nondeterministic step, what happens to the stack?



A. First 0 is pushed onto the stack, then 1.
B. First 1 is pushed onto the stack, then 0.
C. Now PDA has access to two stack and it pushes 0 onto one and 1 onto the other.
D. Two different computational branches are available to the PDA: it pushes exactly one symbol (0 or 1) onto the stack on each branch.
E. None of the above

CS 464

Equivalence of CFGs & PDAs

A language is generated by a CFG

$\Leftrightarrow$

It is recognized by a PDA

CS 464

Converting a CFG to a PDA

Suppose L is generated by a CFG G = (V, Σ, R, S).

Construct a PDA P = (Q, Σ, Γ, δ, q, F)
that recognizes L.

Idea: P will guess steps of a derivation of its input w and use its stack to derive it.

CS 464

Algorithmic description of PDA

(1) Place the marker symbol $ and the start variable on the stack.

(2) Repeat forever:

(a) If a variable V is on top of the stack, and (V → s) ∈ R, pop V and push string s on the stack in reverse order.

Choose the rule from R nondeterministically.

(b) If a terminal is on top of the stack, pop it and match it with input.

(3) On (ε,$), accept.

CS 464

Designing states of PDA

(q_{start}) Push S$ and go to q_{loop}

(q_{loop}) Repeat the following steps forever:

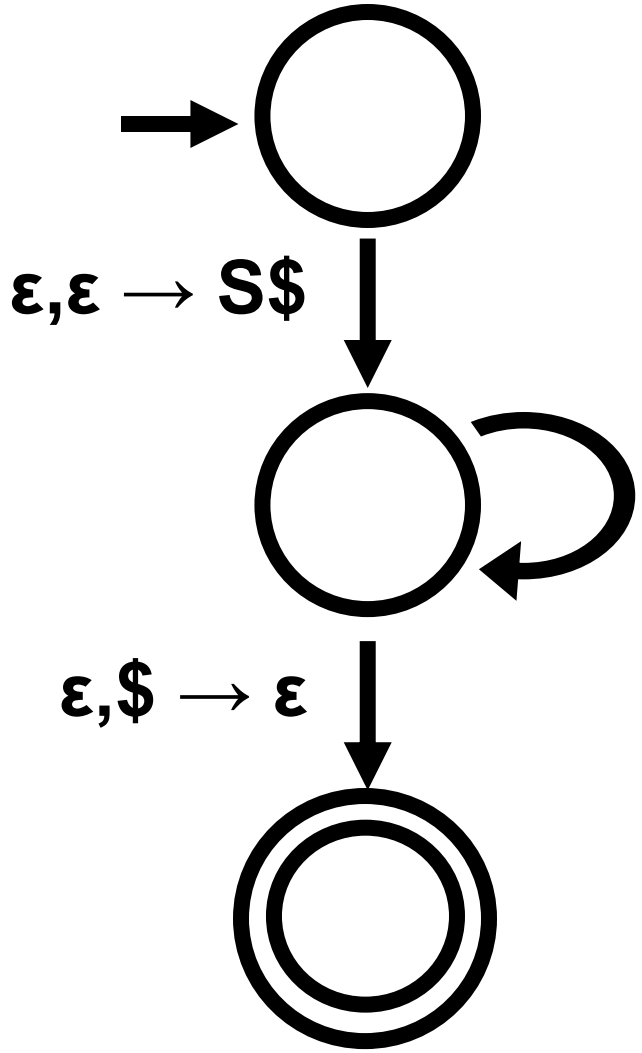
(a) On (ε,V) where (V → s) ∈ R, push s and go to q_{loop}

(b) On (σ,σ), pop σ and go to q_{loop}

(c) On (ε,$) go to q_{accept}

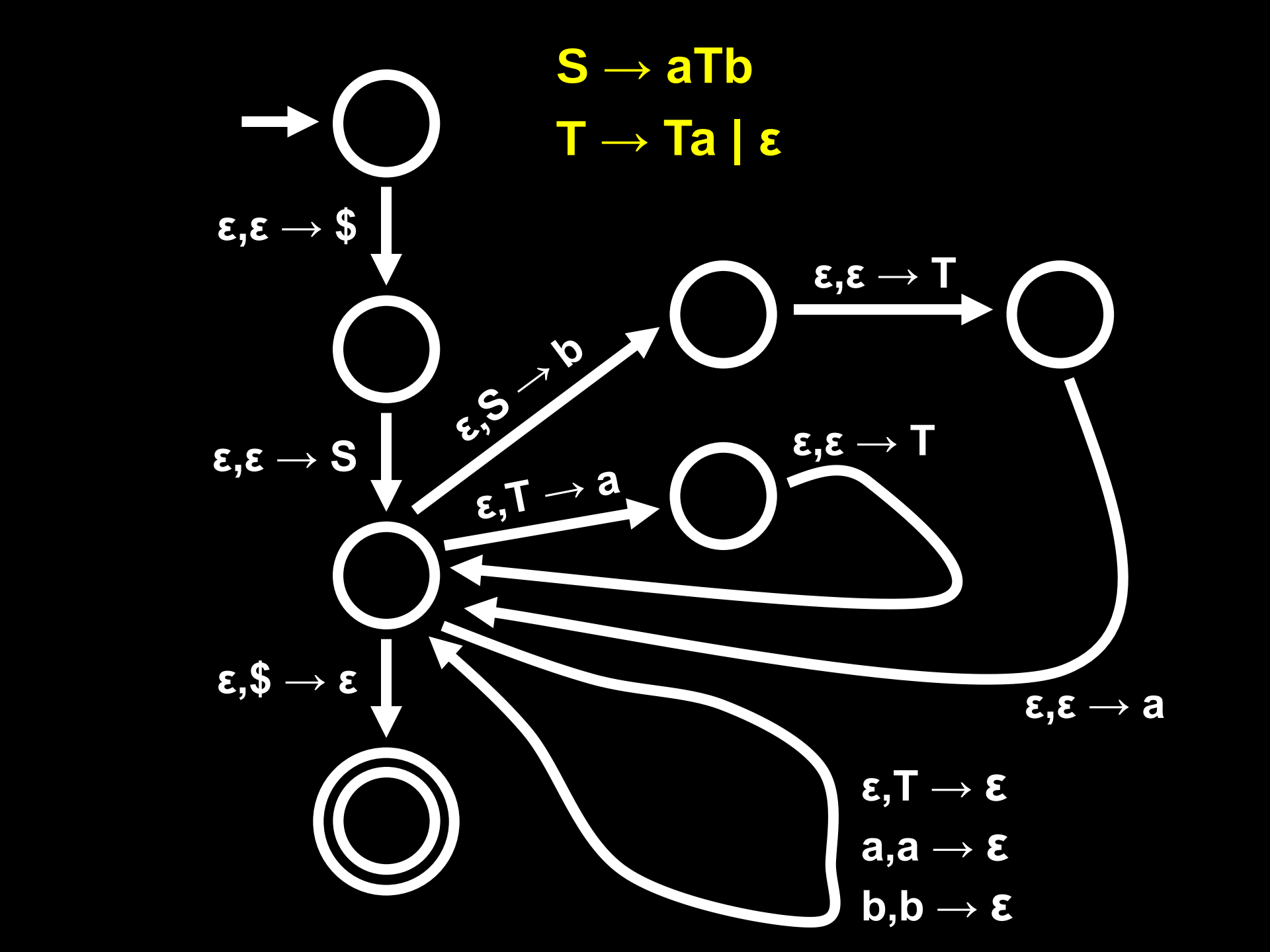
Otherwise, the PDA will get stuck!

Designing PDA



ε,A → w for each rule A → w

a,a → ε for each terminal a


S → aTb
T → Ta | ε
ε,ε → $
ε,ε → S
ε,S → b
ε,ε → T
ε,T → a
ε,ε → T
ε,$ → ε
ε,ε → a
ε,T → ε
a,a → ε
b,b → ε