# Pushdown Automata

Dr. Mousumi Dutt

Module 3

## Generation vs. Recognition

We saw two approaches to describe regular languages:

● Build **automata** that accept precisely the strings in the language.

● Design **regular expressions** that describe precisely the strings in the language.

● Regular expressions **generate** all of the strings in the language.

● Useful for listing off all strings in the language.

● Finite automata **recognize** all of the strings in the language.

● Useful for detecting whether a specific string is in the language.

## Context Free Language

- We saw the **context-free languages**, which are those that can be generated by **context-free grammars**.
- Is there some way to build an automaton that can **recognize** the context-free languages?

## The Problem

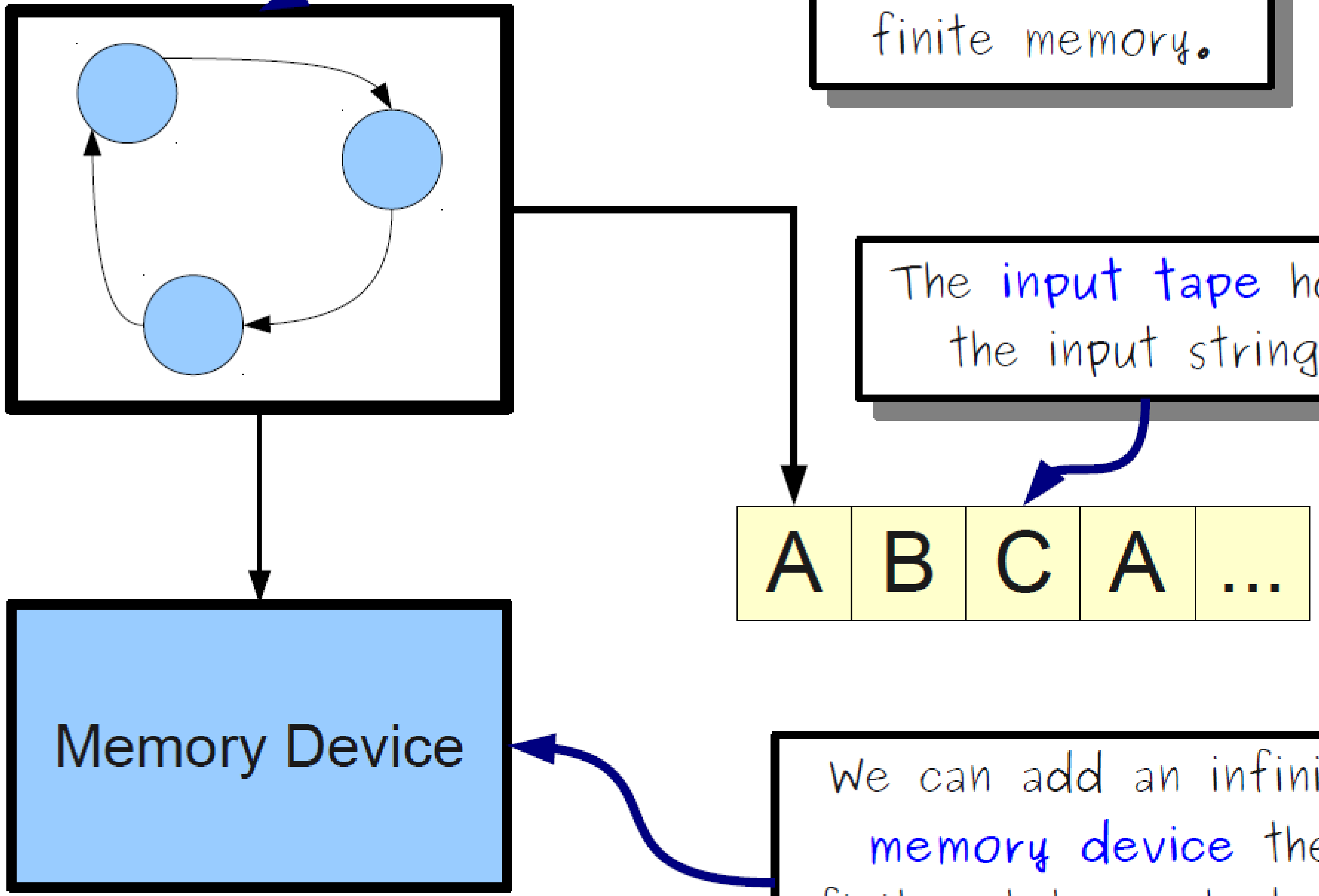
Finite automata accept precisely the regular languages.

We may need unbounded memory to recognize context-free languages.

- e.g. $\{\ \mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\ \}$ requires unbounded counting.

How do we build an automaton with finitely many states but unbounded memory?

The finite-state control acts as a finite memory.



The input tape holds the input string.

We can add an infinite memory device the finite-state control can use to store information.

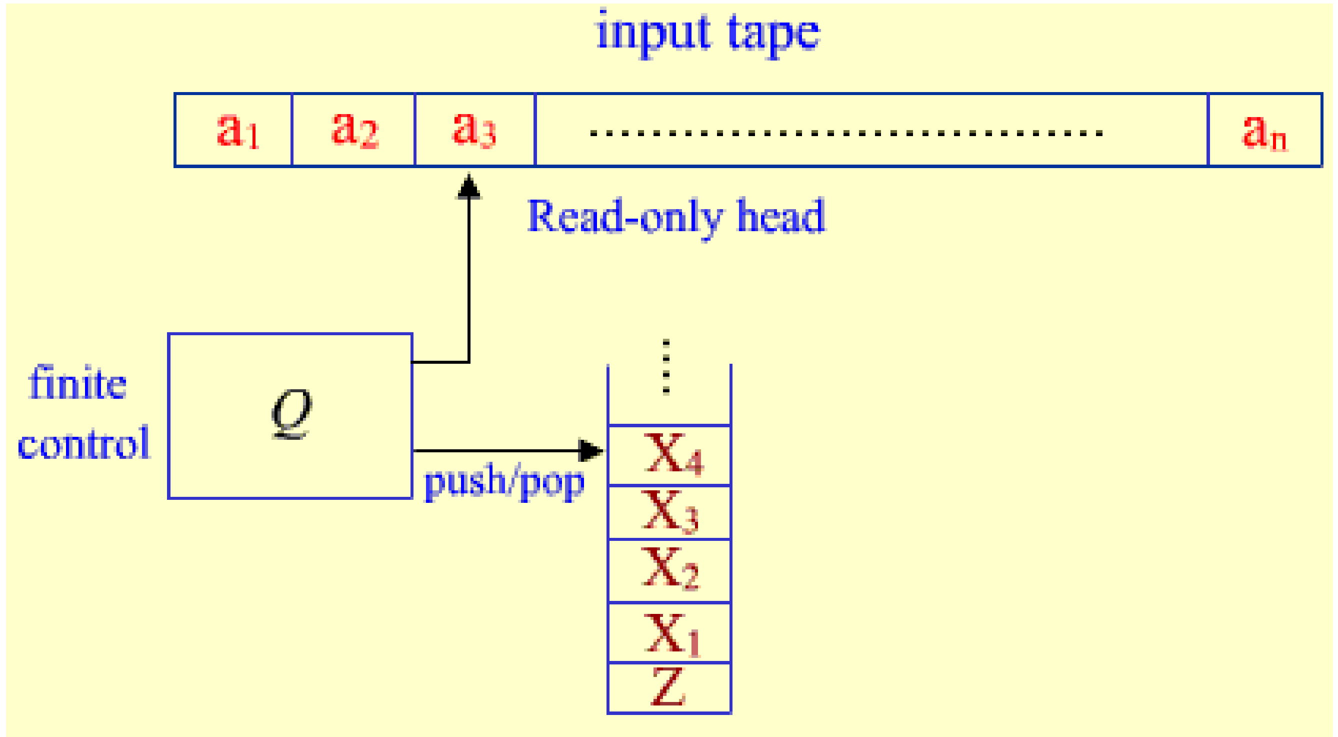
## Adding Memory to Automata

We can augment a finite automaton by adding in a **memory device** for the automaton to store extra information.

- The finite automaton now can base its transition on both the current symbol being read and values stored in memory.
- The finite automaton can issue commands to the memory device whenever it makes a transition.
- e.g. add new data, change existing data, etc.

## Stack Based Memory

- Only the top of the stack is visible at any point in time.
- New symbols may be pushed onto the stack, which cover up the old stack top.
- The top symbol of the stack may be popped, exposing the symbol below it.



# Pushdown Automata

Regular language can be charaterized as the language accepted by finite automata. Similarly, we can characterize the context-free language as the langauge accepted by a class of machines called "Pushdown Automata" (PDA). A pushdown automation is an extension of the NFA.

It is observed that FA have limited capability. (in the sense that the class of languages accepted or characterized by them is small). This is due to the "finite memory" (number of states) and "no external memory" involved with them. A PDA is simply an NFA augmented with an "external stack memory". The addition of a stack provides the PDA with a last-in, first-out memory management cpapability. This "Stack" or "pushdown store" can be used to record a potentially unbounded information. It is due to this memory management capability with the help of the stack that a PDA can overcome the memory limitations that prevents a FA to accept many interesting languages like $\{a^n b^n \mid n \geq 0\}$. Although, a PDA can store an unbounded amount of information on the stack, its access to the information on the stack is limited. It can push an element onto the top of the stack and pop off an element from the top of the stack. To read down into the stack the top elements must be popped off and are lost. Due to this limited access to the information on the stack, a PDA still has some limitations and cannot accept some other interesting languages.

As shown in figure, a PDA has three components: an input tape with read only head, a finite control and a pushdown store.

The input head is read-only and may only move from left to right, one symbol (or cell) at a time. In each step, the PDA pops the top symbol off the stack; based on this symbol, the input symbol it is currently reading, and its present state, it can push a sequence of symbols onto the stack, move its read-only head one cell (or symbol) to the right, and enter a new state, as defined by the transition rules of the PDA.

PDA are nondeterministic, by default. That is, $\epsilon$- transitions are also allowed in which the PDA can pop and push, and change state without reading the next input symbol or moving its read-only head. Besides this, there may be multiple options for possible next moves.

# Pushdown Automata

**Formal Definitions :** Formally, a PDA $M$ is a 7-tuple $M$ = $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where,

- $Q$ is a finite set of states,
- $\Sigma$ is a finite set of input symbols (input alphabets),
- $\Gamma$ is a finite set of stack symbols (stack alphabets),
- $\delta$ is a transition function from $Q \times (\Sigma \cup \{\in\}) \times \Gamma$ to subset of $Q \times \Gamma^*$
- $q_0 \in Q$ is the start state
- $z_0 \in \Gamma$, is the initial stack symbol, and
- $F^* \subseteq Q$, is the final or accept states.

## Pushdown Automata

**Formal Definitions :** Formally, a PDA $M$ is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where,

- $Q$ is a finite set of states,
- $\Sigma$ is a finite set of input symbols (input alphabets),
- $\Gamma$ is a finite set of stack symbols (stack alphabets),
- $\delta$ is a transition function from $Q \times (\Sigma \cup \{\in\}) \times \Gamma$ to subset of $Q \times \Gamma^*$
- $q_0 \in Q$ is the start state
- $z_0 \in \Gamma$, is the initial stack symbol, and
- $F^* \subseteq Q$, is the final or accept states.

**Explanation of the transition function, $\delta$:**

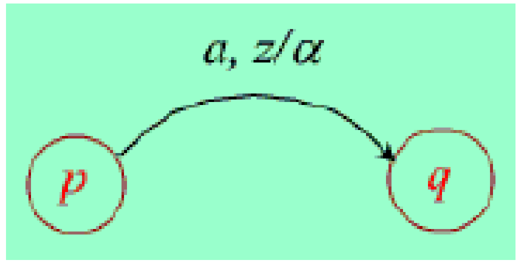
If, for any $a \in \Sigma$, $\delta(q,a,z) = \{(p_1, \beta_1), (p_2, \beta_2), \cdots, (p_k, \beta_k)\}$. This means intitutively that whenever the PDA is in state $q$ reading input symbol $a$ and $z$ on top of the stack, it can nondeterministically for any $i$, $1 \leq i \leq k$

- go to state $p_i$
- pop $z$ off the stack
- push $\beta_i$ onto the stack (where $\beta_1 \in \Gamma^*$) (The usual convention is that if $\beta_i = X_1 X_2 \cdots X_n$, then $X_1$ will be at the top and $X_n$ at the bottom.)
- move read head right one cell past the current symbol $a$.

If $a$ = $\in$, then $\delta(q, \in, z) = \{(p_1, \beta_1), (p_2, \beta_2), \cdots (p_k, \beta_k)\}$ means intitutively that whenver the PDA is in state $q$ with $z$ on the top of the stack regardless of the current input symbol, it can nondeterministically for any $i$, $1 \leq i \leq k$,

- go to state $p_i$
- pop $z$ off the stack
- push $\beta_1$ onto the stack, and
- leave its read-only head where it is.

**State transition diagram :** A PDA can also be depicted by a state transition diagram. The labels on the arcs indicate both the input and the stack operation. The transition $\delta(p,a,z)=\{(q,\alpha)\}$ for $a \in \Sigma \cup \{\epsilon\}$, $p,q \in Q, z \in \Gamma$ and $\alpha \in \Gamma^*$ is depicted by



Final states are indicated by double circles and the start state is indicated by an arrow to it from nowhere.

## Configuration or Instantaneous Description (ID) :

A configuration or an instantaneous description (ID) of PDA at any moment during its computation is an element of $Q \times \Sigma^* \times \Gamma^*$ describing the current state, the portion of the input remaining to be read (i.e. under and to the right of the read head), and the current stack contents. Only these three elements can affect the computation from that point on and, hence, are parts of the ID.

The start or inital configuartion (or ID) on input $\omega$ is $(q_0, \omega, z_0)$. That is, the PDA always starts in its start state, $q_0$ with its read head pointing to the leftmost input symbol and the stack containing only the start/initial stack symbol, $z_0$.

The "next move relation" one figure describes how the PDA can move from one configuration to another in one step.

Formally,

$$(q, a\omega, z\alpha) \vdash_{M} (p, \omega, \beta\alpha)$$

iff $(p, \beta) \in \delta(q, a, z)$

'$a$' may be $\in$ or an input symbol.

Let $I$, $J$, $K$ be IDs of a PDA. We define we write $I \vdash^{i}_{M} K$, if ID $I$ can become $K$ after exactly $i$ moves. The relations $\vdash^{N}_{M}$ and $\vdash^{*}_{M}$ define as follows

$I K \vdash^{0}_{N}$

$I \vdash^{n+1}_{N} J$ if $\exists K$ such that $I \vdash^{n}_{N} K$ and $K \vdash^{1}_{N} J$

$I \vdash^{*}_{N} J$ if $\exists\ n \geq 0$ such that $I \vdash^{n}_{N} J$.

That is, $\vdash^{*}_{M}$ is the reflexive, transitive closure of $\vdash_{M}$. We say that $I \vdash^{*}_{N} J$ if the ID $J$ follows from the ID $I$ in zero or more moves.

( Note : subscript $M$ can be dropped when the particular PDA $M$ is understood. )

## Pushdown Automata

Consider the language

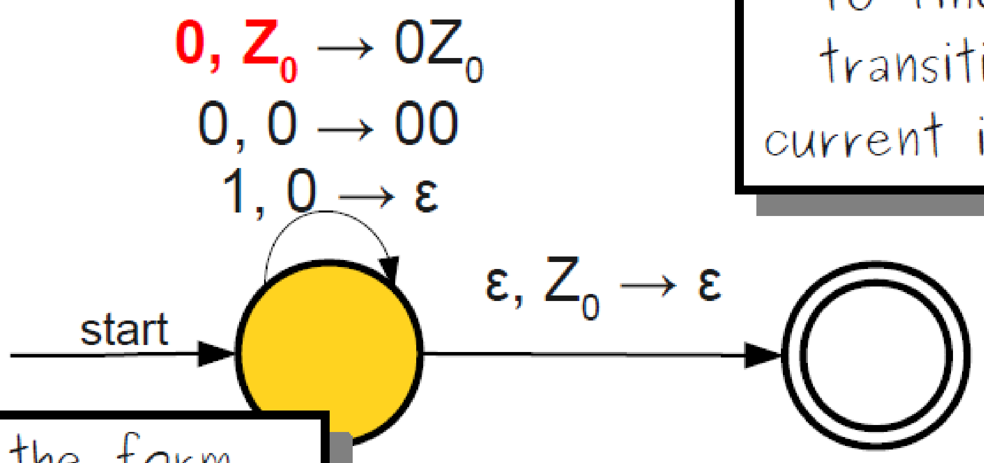
$$L = \{\ w \in \Sigma^* \mid w \text{ is a string of balanced digits}\ \}$$

over $\Sigma = \{\ 0, 1\ \}$

We can exploit the stack to our advantage:

- Whenever we see a 0, push it onto the stack.
- Whenever we see a 1, pop the corresponding 0 from the stack (or fail if not matched)
- When input is consumed, if the stack is empty, accept.

# Pushdown Automata

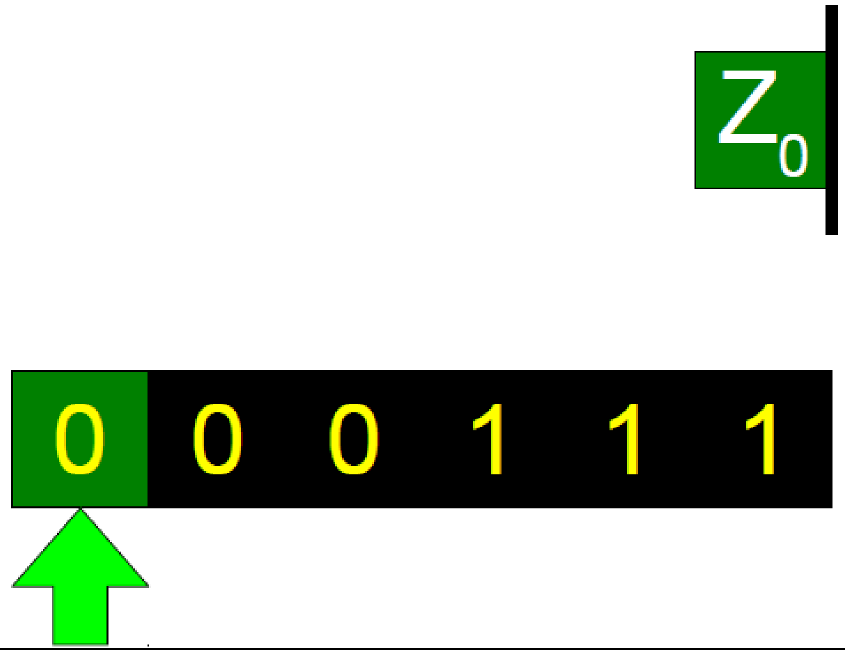


To find an applicable transition, match the current input/stack pair.

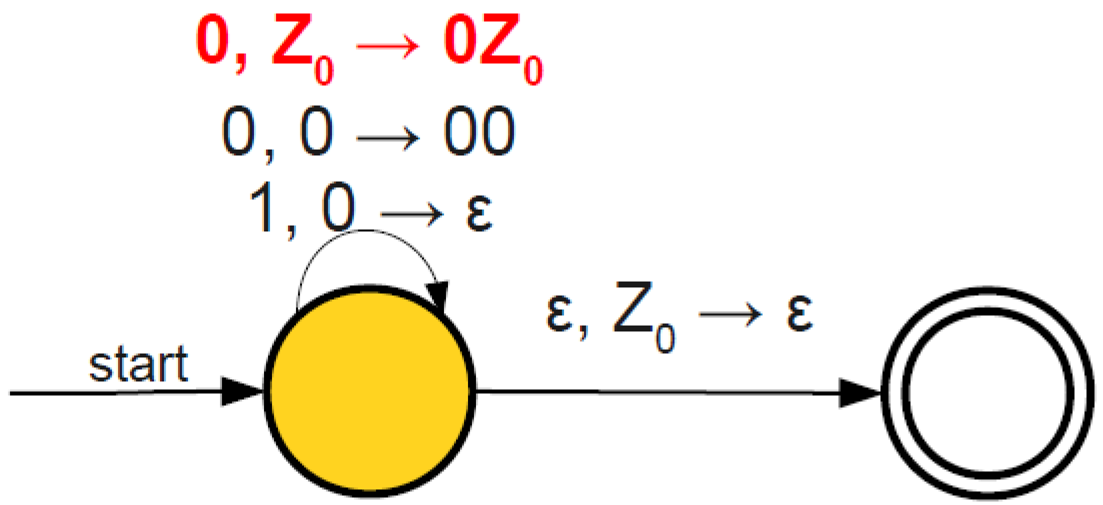
A transition of the form

$$a, b \to z$$

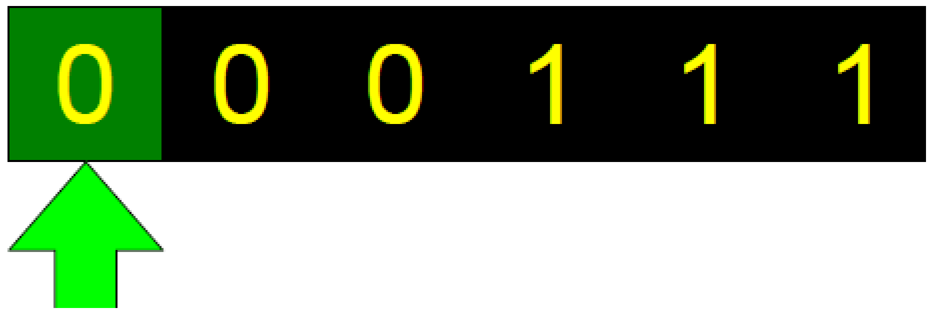
Means "If the current input symbol is a and the current stack symbol is b, then follow this transition, pop b, and push the string z.



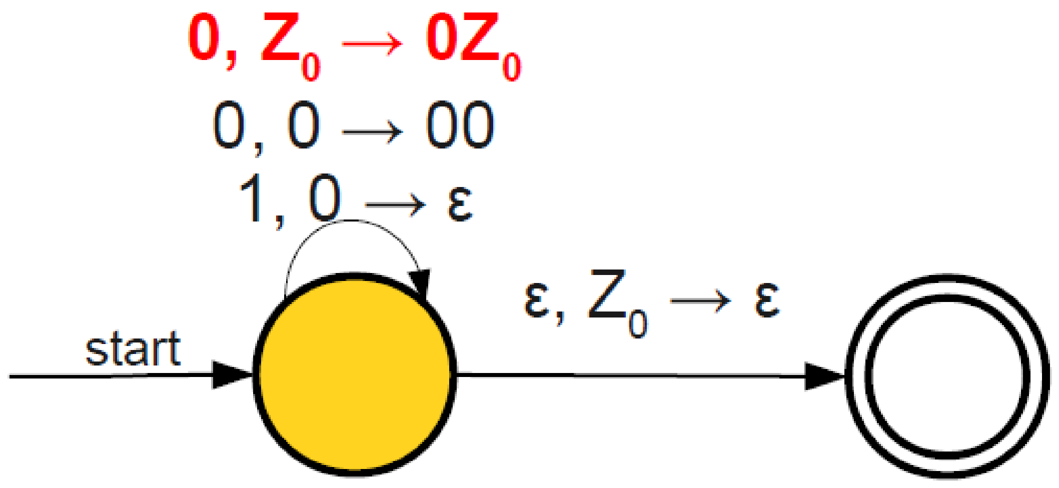
# Pushdown Automata



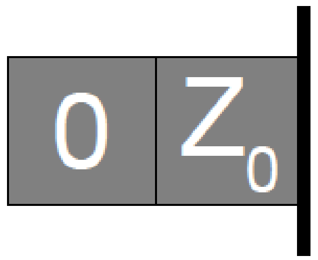
If a transition reads the top symbol of the stack, it always pops that symbol (though it might replace it)

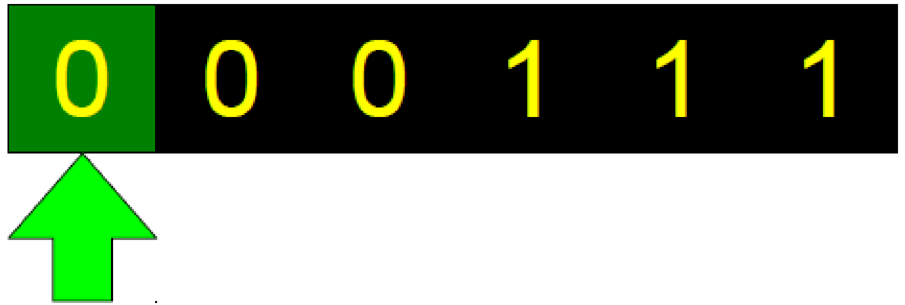


# Pushdown Automata

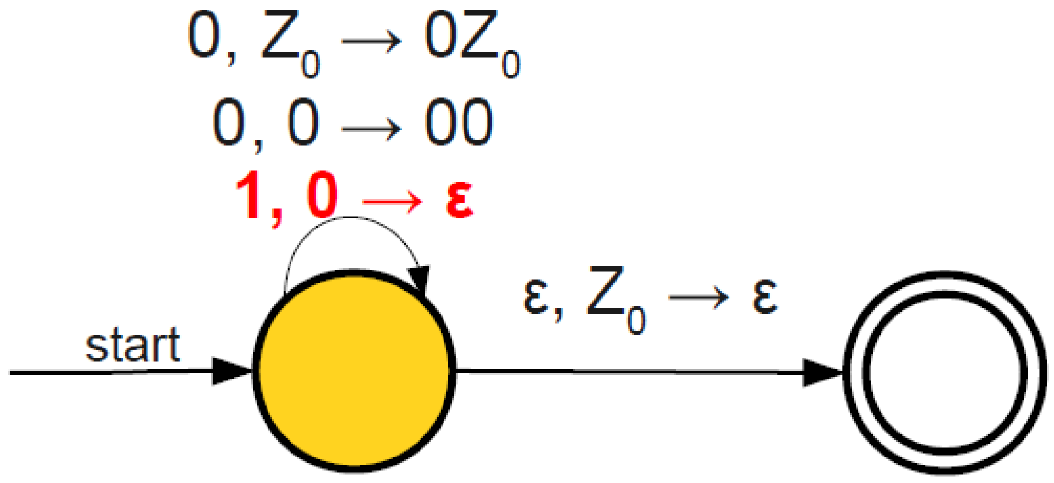


Each transition then pushes some (possibly empty) string back onto the stack. Notice that the leftmost symbol is pushed onto the top.



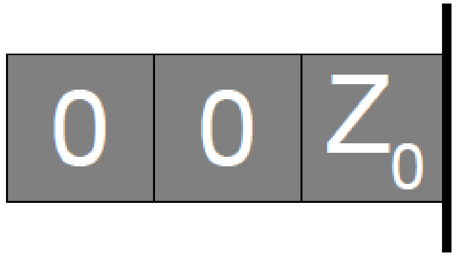


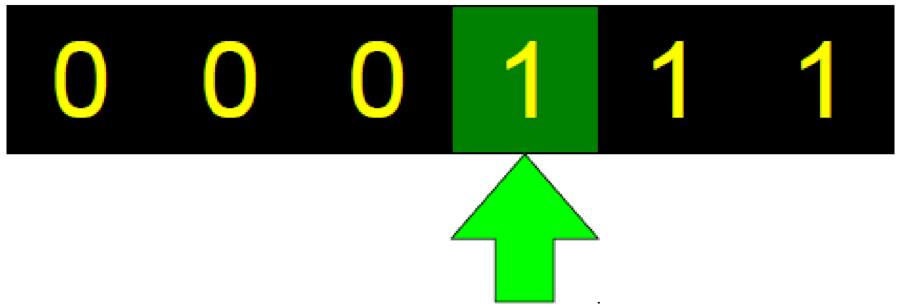
## Pushdown Automata



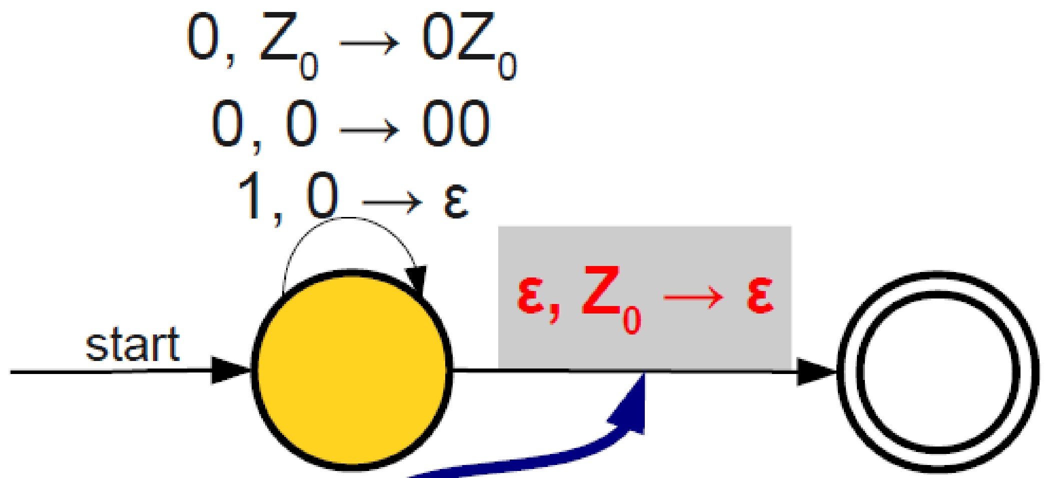
$0, Z_0 \rightarrow 0Z_0$
$0, 0 \rightarrow 00$
$1, 0 \rightarrow \varepsilon$
start
$\varepsilon, Z_0 \rightarrow \varepsilon$

We now push the string ε onto the stack, which adds no new characters. This essentially means "pop the stack."

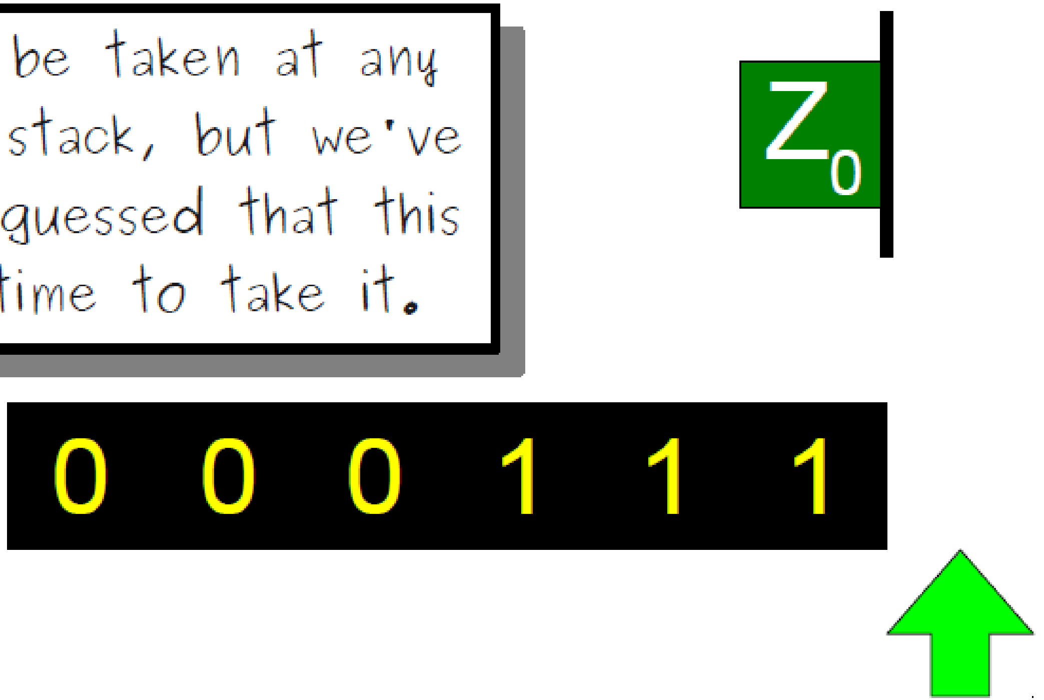




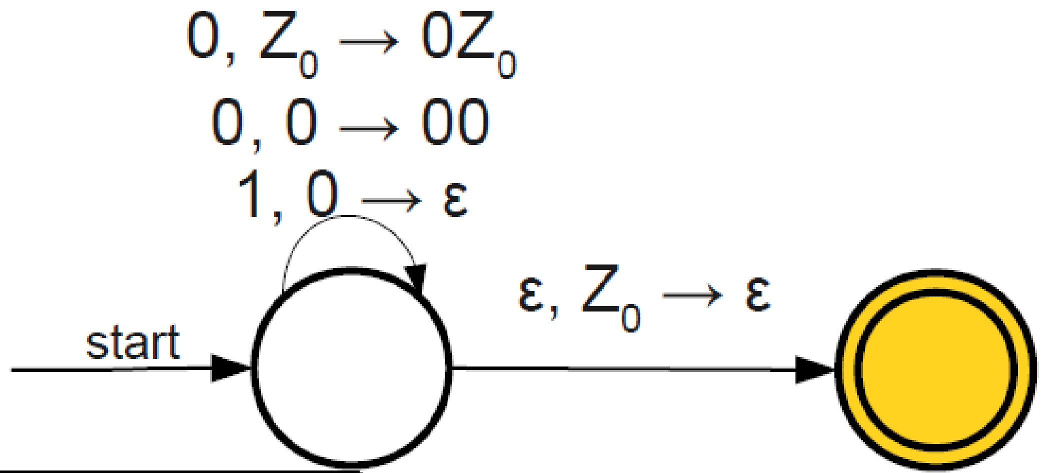
# Pushdown Automata

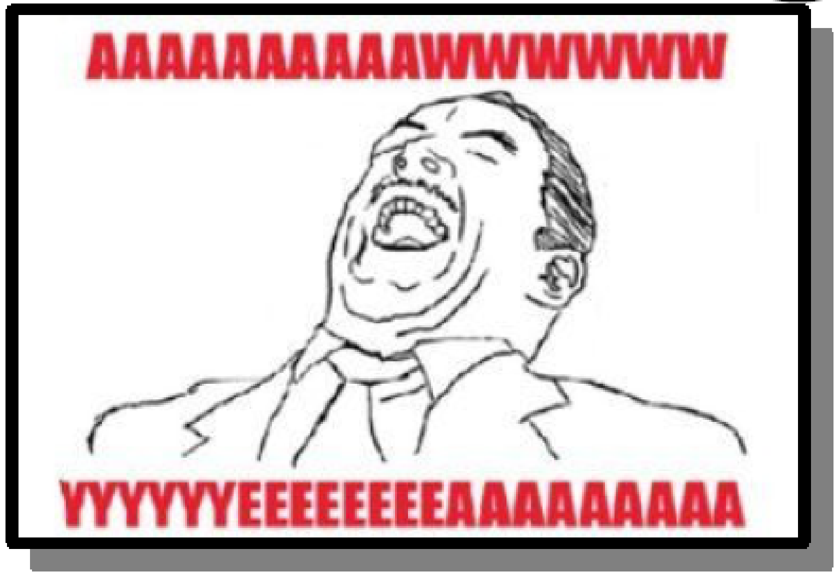


This transition can be taken at any time $Z_0$ is atop the stack, but we've nondeterministically guessed that this would be a good time to take it.



## Pushdown Automata





0 0 0 1 1 1

## Language of a Pushdown Automata

The **language of a PDA** is the set of strings that the PDA accepts:

$$\mathscr{L}(P) = \{ w \in \Sigma^* \mid P \text{ accepts } w \}$$

If $P$ is a PDA where $\mathscr{L}(P) = L$, we say that $P$ **recognizes** $L$.

- Another language defined by the same PDA is by *empty stack*.
- If P is a PDA, then N(P) is the set of strings w such that $(q_0, w, Z_0) \vdash^* \quad (q, \epsilon, \epsilon)$ for any state q.

## Instantaneous Descriptions

- We can formalize the pictures just seen with an *instantaneous description* (ID).
- A ID is a triple ($q$, $w$, $\alpha$), where:

1. $q$ is the current state.
2. $w$ is the remaining input.
3. $\alpha$ is the stack contents, top at the left.

## The "Goes-To" Relation

- To say that ID I can become ID J in one move of the PDA, we write $I \vdash J$.
- Formally, $(q, aw, X\alpha) \vdash (p, w, \beta\alpha)$ for any w and $\alpha$, if $\delta(q, a, X)$ contains $(p, \beta)$.
- Extend $\vdash$ to $\vdash^*$, meaning "zero or more moves," by:
  - Basis: $I \vdash^* I$.
  - Induction: If $I \vdash^* J$ and $J \vdash K$, then $I \vdash^* K$.

## Example: Goes-To

- Using the previous example PDA, we can describe the sequence of moves by: $(q, 000111, Z_0) \vdash (q, 00111, XZ_0) \vdash (q, 0111, XXZ_0) \vdash (q, 111, XXXZ_0) \vdash (p, 11, XXZ_0) \vdash (p, 1, XZ_0) \vdash (p, \epsilon, Z_0) \vdash (f, \epsilon, Z_0)$
- Thus, $(q, 000111, Z_0) \vdash^* (f, \epsilon, Z_0)$.
- What would happen on input 0001111?

Legal because a PDA can use ϵ input even if input remains.

## Answer

- $(q, 0001111, Z_0) \vdash (q, 001111, XZ_0) \vdash (q, 01111, XXZ_0) \vdash (q, 1111, XXXZ_0) \vdash (p, 111, XXZ_0) \vdash (p, 11, XZ_0) \vdash (p, 1, Z_0) \vdash (f, 1, Z_0)$

- Note the last ID has no move.
- 0001111 is not accepted, because the input is not completely consumed.

## Aside: FA and PDA Notations

- We represented moves of a FA by an extended δ, which did not mention the input yet to be read.
- We could have chosen a similar notation for PDA's, where the FA state is replaced by a state-stack combination, like the pictures just shown.

## FA and PDA Notations – (2)

- Similarly, we could have chosen a FA notation with ID's.
  - Just drop the stack component.
- Why the difference? My theory:
- FA tend to model things like protocols, with indefinitely long inputs.
- PDA model parsers, which are given a fixed program to process.

## PDA Example: $L_{wwr} = \{ww^R \mid w \in (0+1)^*\}$

The language, $L_{wwr}$, is the even-length palindromes over alphabet {0,1}.

$L_{wwr}$ is a Context-Free Language (CFL) generated by the grammar:

$$S \rightarrow 0S0 \mid 1S1 \mid \varepsilon$$

One PDA for $L_{wwr}$ is given on the following slide...

$$P = (Q, \Sigma, \Gamma, \delta, q_0, \tau_0, F)$$

# PDA for $L_{wwr}$

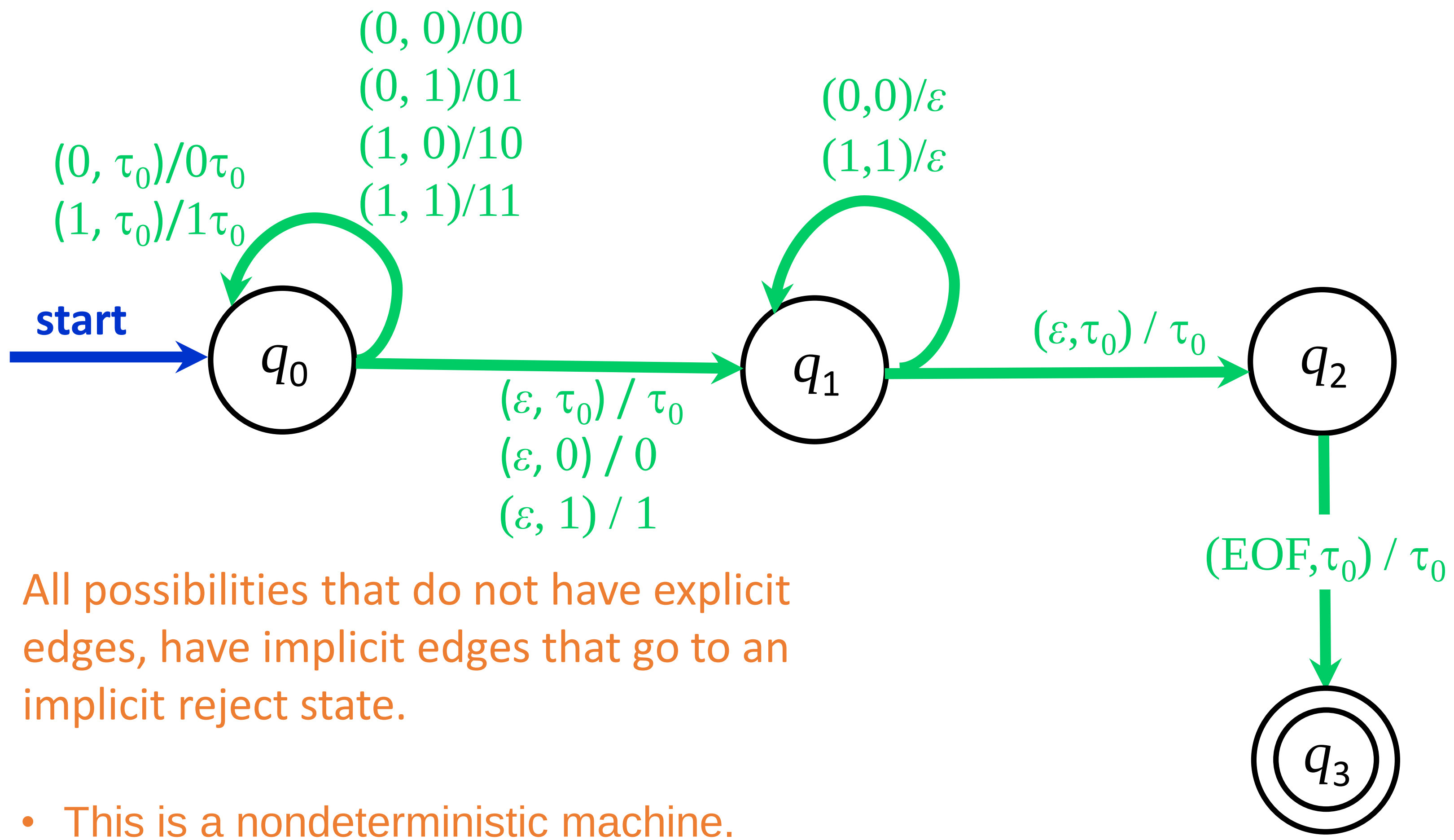
$$Q = \{q_0, q_1, q_2, q_3\} \quad \Sigma = \{0,1\} \quad \Gamma = \{0,1,\tau_0\} \quad F = \{q_3\}$$

$$\delta = \begin{cases}
\boxed{1) \quad \delta(q_0,0,\tau_0) \to (q_0,0\tau_0) \quad \delta(q_0,1,\tau_0) \to (q_0,1\tau_0)} \\
\boxed{\begin{array}{ll} 2) \quad \delta(q_0,0,0) \to (q_0,00) & \delta(q_0,0,1) \to (q_0,01) \\ \quad\;\; \delta(q_0,1,0) \to (q_0,10) & \delta(q_0,1,1) \to (q_0,11) \end{array}} \\
\boxed{3) \quad \delta(q_0,\varepsilon,\tau_0) \to (q_1,\tau_0) \quad \delta(q_0,\varepsilon,0) \to (q_1,0) \quad \delta(q_0,\varepsilon,1) \to (q_1,1)} \\
\boxed{4) \quad \delta(q_1,0,0) \to (q_1,\varepsilon) \quad \delta(q_1,1,1) \to (q_1,\varepsilon)} \\
\boxed{5) \quad \delta(q_1,\varepsilon,\tau_0) \to (q_2,\tau_0)} \\
\boxed{6) \quad \delta(q_2,\mathrm{Read\_Past\_End\_Of\_Input},\tau_0) \to (q_3,\tau_0)}
\end{cases}$$

# A Graphical Notation for PDA’s

1. The nodes correspond to the states of the PDA.
2. An arrow labeled *Start* indicates the unique start state.
3. Doubly circled states are accepting states.
4. Edges correspond to transitions in the PDA as follows:
5. An edge labeled $(a_i, \tau_m)/\tau_n$ from state $q$ to state $p$ means that $\delta(q, a_i, \tau_m)$ contains the pair $(p, \tau_n)$, perhaps among other pairs.

## Graphical Notation for PDA of $L_{wwr}$



All possibilities that do not have explicit edges, have implicit edges that go to an implicit reject state.

- This is a nondeterministic machine.
- Think of the machine as following all possible paths.
- Kill a path if it leads to a reject state.
- If any path leads to an accept state, then the machine accepts.

- **Example:** balanced parentheses,
- e.g. in-language: ((())()),  or  (())(), but not-in-language: ((())

  M = ({$q_1$}, {“(“, “)”}, {L, #}, δ, $q_1$, #, Ø)

  δ:

  (1) δ($q_1$, (, #) = {($q_1$, L#)}   // stack order: L-on top-then- # lower

  (2) δ($q_1$, ), #) = Ø   // illegal, string rejected

  (3) δ($q_1$, (, L) = {($q_1$, LL)}

  (4) δ($q_1$, ), L) = {($q_1$, ε)}

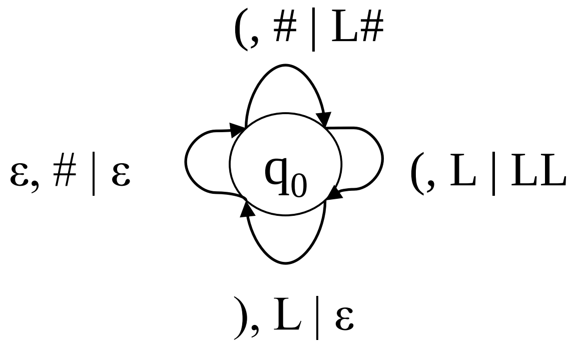
  (5) δ($q_1$, ε, #) = {($q_1$, ε)} //if ε  read & stack hits bottom, accept

  (6) δ($q_1$, ε, L) = Ø   // illegal, string rejected

  // *What does it mean? When will it happen?*

- **Goal:** (acceptance)
  - Read the entire input string
  - Terminate with an empty stack
- Informally, a string is accepted if there exists a computation that uses up all the input and leaves the stack empty.
- *How many rules should be in delta?*

- Transition Diagram:



- Example Computation:

| Current Input | Stack | Transition | |
|---|---|---|---|
| (()) | # | -- initial status | |
| ()) | L# | (1) | - Could have applied rule (5), but |
| )) | LL# | (3) | it would have done no good |
| ) | L# | (4) | |
| ε | # | (4) | |
| ε | - | (5 | |

**Example PDA #1:** For the language {x | x = wcw$^r$ and *w* in {0,1}*, but sigma={0,1,c}}

- *Is this a regular language?*
- *Note: length |x| is odd*

M = $(\{q_1, q_2\}, \{0, 1, c\}, \{\#, B, G\}, \delta, q_1, \#, \emptyset)$

$\delta$:

| | | | |
|---|---|---|---|
| (1) | $\delta(q_1, 0, \#) = \{(q_1, B\#)\}$ $\leftarrow$ | (9) | $\delta(q_1, 1, \#) = \{(q_1, G\#)\}$ |
| (2) | $\delta(q_1, 0, B) = \{(q_1, BB)\}$ | (10) | $\delta(q_1, 1, B) = \{(q_1, GB)\}$ |
| (3) | $\delta(q_1, 0, G) = \{(q_1, BG)\}$ | (11) | $\delta(q_1, 1, G) = \{(q_1, GG)\}$ |
| (4) | $\delta(q_1, c, \#) = \{(q_2, \#)\}$ | | |
| (5) | $\delta(q_1, c, B) = \{(q_2, B)\}$ | | |
| (6) | $\delta(q_1, c, G) = \{(q_2, G)\}$ | | |
| (7) | $\delta(q_2, 0, B) = \{(q_2, \varepsilon)\}$ | (12) | $\delta(q_2, 1, G) = \{(q_2, \varepsilon)\}$ |
| (8) | $\delta(q_2, \varepsilon, \#) = \{(q_2, \varepsilon)\}$ | | |

- **Notes:**
  - Stack grows leftwards
  - Only rule #8 is non-deterministic.
  - Rule #8 is used to pop the final stack symbol off at the end of a computation.

- **Example Computation:**

(1) $\delta(q_1, 0, \#) = \{(q_1, B\#)\}$ (9) $\delta(q_1, 1, \#) = \{(q_1, G\#)\}$

(2) $\delta(q_1, 0, B) = \{(q_1, BB)\}$ (10) $\delta(q_1, 1, B) = \{(q_1, GB)\}$

(3) $\delta(q_1, 0, G) = \{(q_1, BG)\}$ (11) $\delta(q_1, 1, G) = \{(q_1, GG)\}$

(4) $\delta(q_1, c, \#) = \{(q_2, \#)\}$

(5) $\delta(q_1, c, B) = \{(q_2, B)\}$

(6) $\delta(q_1, c, G) = \{(q_2, G)\}$

(7) $\delta(q_2, 0, B) = \{(q_2, \varepsilon)\}$ (12) $\delta(q_2, 1, G) = \{(q_2, \varepsilon)\}$

(8) $\delta(q_2, \varepsilon, \#) = \{(q_2, \varepsilon)\}$

| State | Input | Stack | Rule Applied | Rules Applicable |
|---|---|---|---|---|
| $q_1$ | **0**1c10 | # | | (1) |
| $q_1$ | **1**c10 | B# | (1) | (10) |
| $q_1$ | **c**10 | GB# | (10) | (6) |
| $q_2$ | **1**0 | GB# | (6) | (12) |
| $q_2$ | **0** | B# | (12) | (7) |
| $q_2$ | **ε** | # | (7) | (8) |
| $q_2$ | ε | ε | (8) | - |

- **Example Computation:**

|  |  |  |  |
|---|---|---|---|
| (1) | $\delta(q_1, 0, \#) = \{(q_1, B\#)\}$ | (9) | $\delta(q_1, 1, \#) = \{(q_1, G\#)\}$ |
| (2) | $\delta(q_1, 0, B) = \{(q_1, BB)\}$ | (10) | $\delta(q_1, 1, B) = \{(q_1, GB)\}$ |
| (3) | $\delta(q_1, 0, G) = \{(q_1, BG)\}$ | (11) | $\delta(q_1, 1, G) = \{(q_1, GG)\}$ |
| (4) | $\delta(q_1, c, \#) = \{(q_2, \#)\}$ |  |  |
| (5) | $\delta(q_1, c, B) = \{(q_2, B)\}$ |  |  |
| (6) | $\delta(q_1, c, G) = \{(q_2, G)\}$ |  |  |
| (7) | $\delta(q_2, 0, B) = \{(q_2, \varepsilon)\}$ | (12) | $\delta(q_2, 1, G) = \{(q_2, \varepsilon)\}$ |
| (8) | $\delta(q_2, \varepsilon, \#) = \{(q_2, \varepsilon)\}$ |  |  |

| State | Input | Stack | Rule Applied |
|---|---|---|---|
| $q_1$ | **1**c1 | # |  |
| $q_1$ | **c**1 | G# | (9) |
| $q_2$ | **1** | G# | (6) |
| $q_2$ | **ε** | # | (12) |
| $q_2$ | ε | ε | (8) |

- **Definition:** |—* is the reflexive and transitive closure of |—.
  - I |—* I for each instantaneous description I
  - If I |— J and J |—* K then I |—* K
- Intuitively, if I and J are instantaneous descriptions, then I |—* J means that J follows from I by zero or more transitions.

- **Definition:** Let M = ($Q$, $\Sigma$, $\Gamma$, $\delta$, $q_0$, $z_0$, $F$) be a PDA. The *language accepted by empty stack*, denoted $L_E(M)$, is the set

  $$\{w \mid (q_0, w, z_0) \mid\!\!-\!\!-^* (p, \varepsilon, \varepsilon) \text{ for some } p \text{ in } Q\}$$

- **Definition:** Let M = ($Q$, $\Sigma$, $\Gamma$, $\delta$, $q_0$, $z_0$, $F$) be a PDA. The *language accepted by final state*, denoted $L_F(M)$, is the set

  $$\{w \mid (q_0, w, z_0) \mid\!\!-\!\!-^* (p, \varepsilon, \gamma) \text{ for some } p \text{ in } F \text{ and } \gamma \text{ in } \Gamma^*\}$$

- **Definition:** Let M = ($Q$, $\Sigma$, $\Gamma$, $\delta$, $q_0$, $z_0$, $F$) be a PDA. The *language accepted by empty stack and final state*, denoted $L(M)$, is the set

  $$\{w \mid (q_0, w, z_0) \mid\!\!-\!\!-^* (p, \varepsilon, \varepsilon) \text{ for some } p \text{ in } F\}$$

- **Lemma 1:** Let $L = L_E(M_1)$ for some PDA $M_1$. Then there exits a PDA $M_2$ such that $L = L_F(M_2)$.

- **Lemma 2:** Let $L = L_F(M_1)$ for some PDA $M_1$. Then there exits a PDA $M_2$ such that $L = L_E(M_2)$.

- **Theorem:** Let L be a language. Then there exits a PDA $M_1$ such that $L = L_F(M_1)$ if and only if there exists a PDA $M_2$ such that $L = L_E(M_2)$.

- **Corollary:** The PDAs that accept by empty stack and the PDAs that accept by final state define the same class of languages.

- **Note:** Similar lemmas and theorems could be stated for PDAs that accept by both final state and empty stack.

- **Definition:** Let G = (V, T, P, S) be a CFL. If every production in P is of the form

$$A \rightarrow a\alpha$$

  Where A is in V, *a* is in T, and α is in V*, then G is said to be in Greibach Normal Form (GNF).

  Only one non-terminal in front.

- **Example:**

  $S \rightarrow aAB \mid bB$

  $A \rightarrow aA \mid a$

  $B \rightarrow bB \mid c$ *Language:* $(aa^{+}+b)b^{+}c$

- **Theorem:** Let L be a CFL. Then L – {ε} is a CFL.
- **Theorem:** Let L be a CFL not containing {ε}. Then there exists a GNF grammar G such that L = L(G).

- **Lemma 1:** Let *L* be a CFL. Then there exists a PDA *M* such that L = $L_E$(M).
- **Proof:** Assume without loss of generality that ε is not in *L*. The construction can be modified to include ε later.

  Let G = (V, T, P, S) be a CFG, and assume without loss of generality that *G* is in GNF. Construct M = (Q, Σ, Γ, δ, q, z, Ø) where:

  Q = {q}

  Σ = T

  Γ = V

  z = S

  δ: for all *a* in Σ and *A* in Γ, δ(q, a, A) contains (q, γ)

  if A –> aγ is in *P* or rather:

  δ(q, a, A) = {(q, γ) | A –> aγ  is in *P* and γ is in Γ*},

  for all *a* in Σ and *A* in Γ

- For a given string *x* in Σ* , *M* will attempt to simulate a leftmost derivation of *x* with *G*.

- **Example #1:** Consider the following CFG in GNF.

  $S \rightarrow aS$ G is in GNF

  $S \rightarrow a$ $L(G) = a^+$

  Construct M as:

  $Q = \{q\}$

  $\Sigma = T = \{a\}$

  $\Gamma = V = \{S\}$

  $z = S$

  $\delta(q, a, S) = \{(q, S), (q, \varepsilon)\}$

  $\delta(q, \varepsilon, S) = \emptyset$

- *Is $\delta$ complete?*

- **Example #2:** Consider the following CFG in GNF.

(1) S –> aA

(2) S –> aB

(3) A –> aA G is in GNF

(4) A –> aB L(G) = $a^+ b^+$ // This looks ok to me, one, two or more *a*'s in the start

(5) B –> bB

(6) B –> b *[Can you write a simpler equivalent CFG? Will it be GNF?*]

Construct M as:

Q = {q}

Σ = T = {a, b}

Γ = V = {S, A, B}

z = S

(1) δ(q, a, S) = {(q, A), (q, B)} From productions #1 and 2, S->aA, S->aB

(2) δ(q, a, A) = {(q, A), (q, B)} From productions #3 and 4, A->aA, A->aB

(3) δ(q, a, B) = Ø

(4) δ(q, b, S) = Ø

(5) δ(q, b, A) = Ø

(6) δ(q, b, B) = {(q, B), (q, ε)} From productions #5 and 6, B->bB, B->b

(7) δ(q, ε, S) = Ø

(8) δ(q, ε, A) = Ø

(9) δ(q, ε, B) = Ø *Is δ complete?*

- For a string w in L(G) the PDA M will simulate a leftmost derivation of w.
  - If w is in L(G) then $(q, w, z_0) \mid\!\!—^* (q, \varepsilon, \varepsilon)$
  - If $(q, w, z_0) \mid\!\!—^* (q, \varepsilon, \varepsilon)$ then w is in L(G)
- Consider generating a string using G. Since G is in GNF, each sentential form in a *leftmost* derivation has form:

$$=> \underbrace{t_1t_2\ldots t_i}_{\text{terminals}}\ \underbrace{A_1A_2\ldots A_m}_{\text{non-terminals}}$$

- And each step in the derivation (i.e., each application of a production) adds a terminal and some non-terminals.

$$A_1 \to t_{i+1}\alpha$$

$$=> t_1t_2\ldots t_i\ t_{i+1}\ \alpha A_1A_2\ldots A_m$$

- Each transition of the PDA simulates one derivation step. Thus, the $i^{th}$ step of the PDAs' computation corresponds to the $i^{th}$ step in a corresponding leftmost derivation with the grammar.
- After the $i^{th}$ step of the computation of the PDA, $t_1t_2\ldots t_{i+1}$ are the symbols that have already been read by the PDA and $\alpha A_1A_2\ldots A_m$are the stack contents.

- For each leftmost derivation of a string generated by the grammar, there is an equivalent accepting computation of that string by the PDA.

- Each sentential form in the leftmost derivation corresponds to an instantaneous description in the PDA's corresponding computation.

- For example, the PDA instantaneous description corresponding to the sentential form:

$$\Rightarrow t_1 t_2 \ldots t_i \, A_1 A_2 \ldots A_m$$

would be:

$$(q,\ t_{i+1} t_{i+2} \ldots t_n\ ,\ A_1 A_2 \ldots A_m)$$

- **Example:** Using the grammar from example #2:

|  |  |  |
|---|---|---|
| S | => aA | (1) |
|  | => aaA | (3) |
|  | => aaaA | (3) |
|  | => aaaaB | (4) |
|  | => aaaabB | (5) |
|  | => aaaabb | (6) |

*Grammar:*

| | |
|---|---|
| (1) | S –> aA |
| (2) | S –> aB |
| (3) | A –> aA |
| (4) | A –> aB |
| (5) | B –> bB |
| (6) | B –> b |

G is in GNF

L(G) = $a^+b^+$

- The corresponding computation of the PDA:

|  |  | (rule#)/right-side# |
|---|---|---|
| (q, aaaabb, S) | \|— (q, aaabb, A) | (1)/1 |
|  | \|— (q, aabb, A) | (2)/1 |
|  | \|— (q, abb, A) | (2)/1 |
|  | \|— (q, bb, B) | (2)/2 |
|  | \|— (q, b, B) | (6)/1 |
|  | \|— (q, ε, ε) | (6)/2 |

(1)δ(q, a, S) = {(q, A), (q, B)}
(2)δ(q, a, A) = {(q, A), (q, B)}
(3) δ(q, a, B) = Ø
(4) δ(q, b, S) = Ø
(5) δ(q, b, A) = Ø
(6) δ(q, b, B) = {(q, B), (q, ε)}
(7) δ(q, ε, S) = Ø
(8) δ(q, ε, A) = Ø
(9) δ(q, ε, B) = Ø

  - String is read
  - Stack is emptied
  - Therefore the string is accepted by the PDA

- **Another Example:** Using the PDA from example #2:

  (q, aabb, S) |— (q, abb, A) (1)/1

  |— (q, bb, B) (2)/2

  |— (q, b, B) (6)/1

  |— (q, ε, ε) (6)/2

- The corresponding derivation using the grammar:

  S => aA (1)

  => aaB (4)

  => aabB (5)

  => aabb (6)

- **Example #3:** Consider the following CFG in GNF.

|  |  |  |
|---|---|---|
| (1) | S –> aABC | |
| (2) | A –> a | G is in GNF |
| (3) | B –> b | |
| (4) | C –> cAB | |
| (5) | C –> cC | *Language?* |

Construct M as:

$aab\ cc^*\ ab$

Q = {q}

Σ = T = {a, b, c}

Γ = V = {S, A, B, C}

z = S

|  |  |  |  |  |
|---|---|---|---|---|
| (1) | δ(q, a, S) = {(q, ABC)} | S->aABC | (9) | δ(q, c, S) = Ø |
| (2) | δ(q, a, A) = {(q, ε)} | A->a | (10) | δ(q, c, A) = Ø |
| (3) | δ(q, a, B) = Ø | | (11) | δ(q, c, B) = Ø |
| (4) | δ(q, a, C) = Ø | | (12) | δ(q, c, C) = {(q, AB), (q, C))} // C->cAB\|cC |
| (5) | δ(q, b, S) = Ø | | (13) | δ(q, ε, S) = Ø |
| (6) | δ(q, b, A) = Ø | | (14) | δ(q, ε, A) = Ø |
| (7) | δ(q, b, B) = {(q, ε)} | B->b | (15) | δ(q, ε, B) = Ø |
| (8) | δ(q, b, C) = Ø | | (16) | δ(q, ε, C) = Ø |

- **Notes:**
  - Recall that the grammar *G* was required to be in GNF before the construction could be applied.
  - As a result, it was assumed at the start that ε was not in the context-free language *L*.
  - What if ε is in *L*? You  need to add ε back.

- **Suppose ε is in L:**

1) First, let L' = L – {ε}

   Fact: If *L* is a CFL, then L' = L – {ε} is a CFL.

   By an earlier theorem, there is GNF grammar *G* such that L' = L(G).

2) Construct a PDA *M* such that L' = $L_E$(M)

   How do we modify *M* to accept ε?

   Add δ(q, ε, S) = {(q, ε)}?      *NO!!*

- **Counter Example:**

  Consider L = {ε, b, ab, aab, aaab, …}= ε + a*b

  Then L’ = {b, ab, aab, aaab, …} = a*b
- **The GNF CFG for L’:**

  P:

  (1) S –> aS

  (2) S –> b
- **The PDA M Accepting L’:**

  Q = {q}

  Σ = T = {a, b}

  Γ = V = {S}

  z = S

  δ(q, a, S) = {(q, S)}

  δ(q, b, S) = {(q, ε)}

  δ(q, ε, S) = Ø

*How to add ε to L’ now?*

$\delta(q, a, S) = \{(q, S)\}$

$\delta(q, b, S) = \{(q, \varepsilon)\}$

$\delta(q, \varepsilon, S) = \emptyset$

- If $\delta(q, \varepsilon, S) = \{(q, \varepsilon)\}$ is added then:

  $L(M) = \{\varepsilon, a, aa, aaa, \ldots, b, ab, aab, aaab, \ldots\}$, wrong!

  It is like, $S \to aS \mid b \mid \varepsilon$

  which is wrong!

  Correct grammar should be:

  (0) $S_1 \to \varepsilon \mid S$, with new starting non-terminal $S_1$

  (1) $S \to aS$

  (2) $S \to b$

For PDA, add a new *Stack-bottom symbol* $S_1$, with new transitions:

$\delta(q, \varepsilon, S_1) = \{(q, \varepsilon), (q, S)\}$, where *S* was the previous stack-bottom of *M*

Alternatively, add a new *start* state q’ with transitions:

$\delta(q', \varepsilon, S) = \{(q', \varepsilon), (q, S)\}$

- **Lemma 1:** Let *L* be a CFL. Then there exists a PDA *M* such that L = $L_E$(M).

- **Lemma 2:** Let *M* be a PDA. Then there exists a CFG grammar *G* such that $L_E$(M) = L(G).
  - *Can you prove it?*
  - *First step would be to transform an arbitrary PDA to a single state PDA!*

- **Theorem:** Let *L* be a language. Then there exists a CFG *G* such that L = L(G) iff there exists a PDA *M* such that L = $L_E$(M).

- **Corollary:** The PDAs define the CFLs.

## From CFG’s to PDA

***Theorem:*** If $G$ is a CFG for a language $L$, then there exists a PDA for $L$ as well.

**Idea:** Build a PDA that simulates expanding out the CFG from the start symbol to some particular string.

Stack holds the part of the string we haven't matched yet.

## From CFG's to PDA

Example: Let $\Sigma = \{ 1, \geq \}$ and let

$GE = \{ 1^m \geq 1^n \mid m, n \in \mathbb{N} \wedge m \geq n \}$

- $111{\geq}11 \in GE$
- $11{\geq}11 \in GE$
- $1111{\geq}11 \in GE$
- $\geq \in GE$

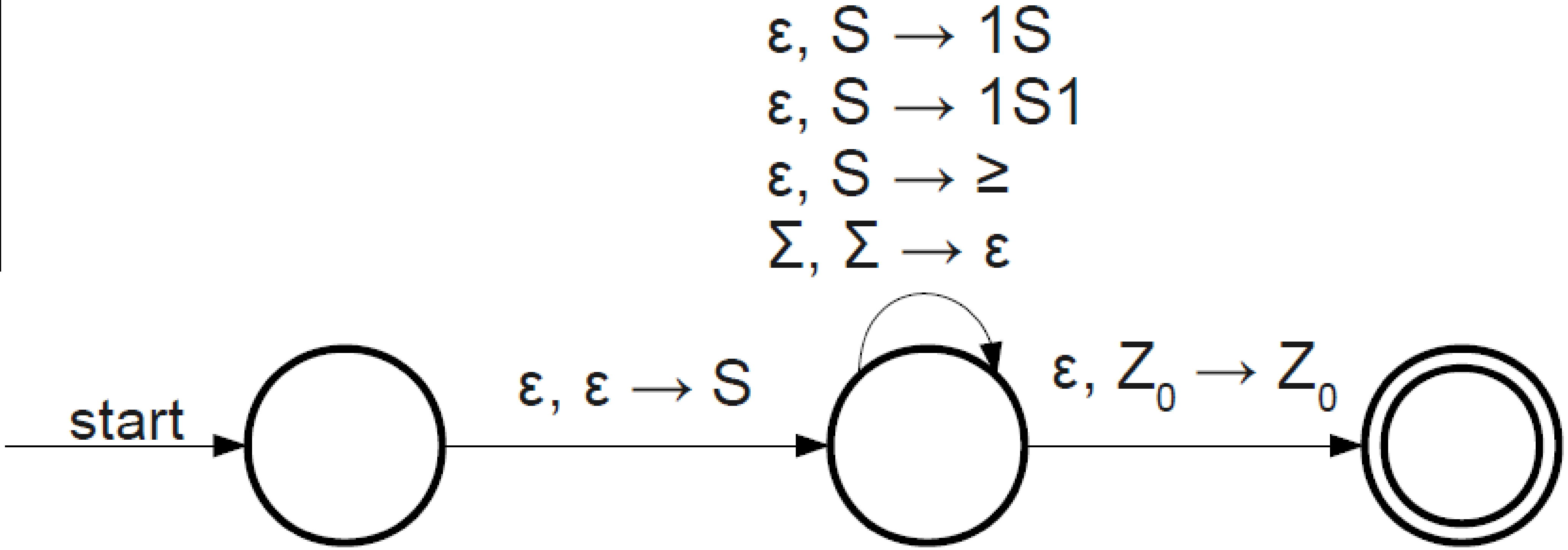
One CFG for $GE$ is the following:

$$\mathbf{S} \rightarrow 1\mathbf{S}1 \mid 1\mathbf{S} \mid \geq$$

How would we build a PDA for $GE$?

# From CFG's to PDA

$S \rightarrow 1S1$
$S \rightarrow 1S$
$S \rightarrow \geq$



# From CFG's to PDA

Make three states: **start**, **parsing**, and **accepting**.

There is a transition $\varepsilon, \varepsilon \rightarrow$ **S** from **start** to **parsing**.

- Corresponds to starting off with the start symbol S.

There is a transition $\varepsilon$, **A** $\rightarrow \boldsymbol{\omega}$ from **parsing** to itself for each production **A** $\rightarrow \boldsymbol{\omega}$.

- Corresponds to predicting which production to use.

There is a transition $\Sigma, \Sigma \rightarrow \varepsilon$ from **parsing** to itself.

- Corresponds to matching a character of the input.

There is a transition $\varepsilon, Z_0 \rightarrow Z_0$ from **parsing** to **accepting**.

- Corresponds to completely matching the input.

## From CFG’s to PDA

The PDA constructed this way is called a **predict/match parser**.

Each step either **predicts** which production to use or **matches** some symbol of the input.

## From PDA’s to CFG

The other direction of the proof (converting a PDA to a CFG) is much harder.

Intuitively, create a CFG representing paths between states in the PDA.

## DPDA

A **deterministic pushdown automaton** is a PDA with the extra property that

For each state in the PDA, and for any combination of a current input symbol and a current stack symbol, there is **at most** one transition defined.

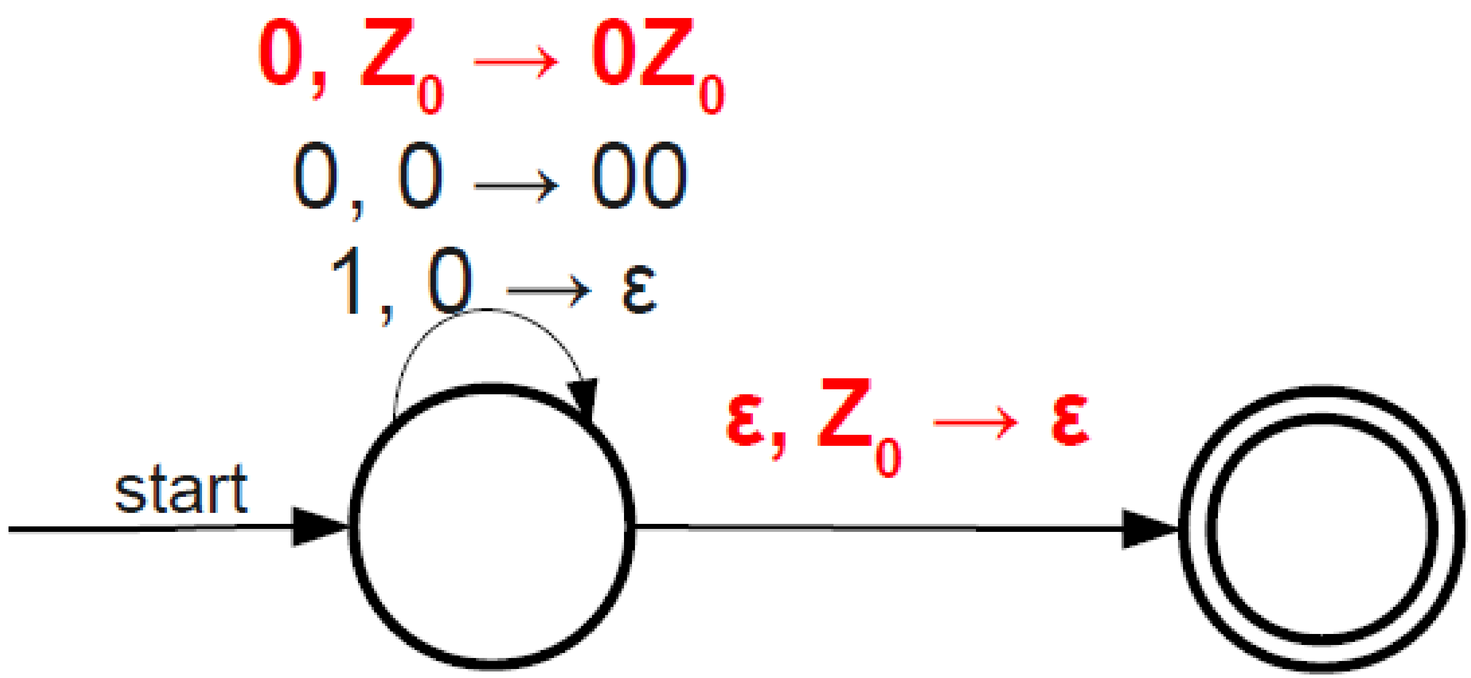
In other words, there is ***at most*** one legal sequence of transitions that can be followed for any input.

This does ***not*** preclude ε-transitions, as long as there is never a conflict between following the ε-transition or some other transition.

However, there can be ***at most*** one ε-transition that could be followed at any one time.

This does ***not*** preclude the automaton "dying" from having no transitions defined; DPDAs can have undefined transitions.

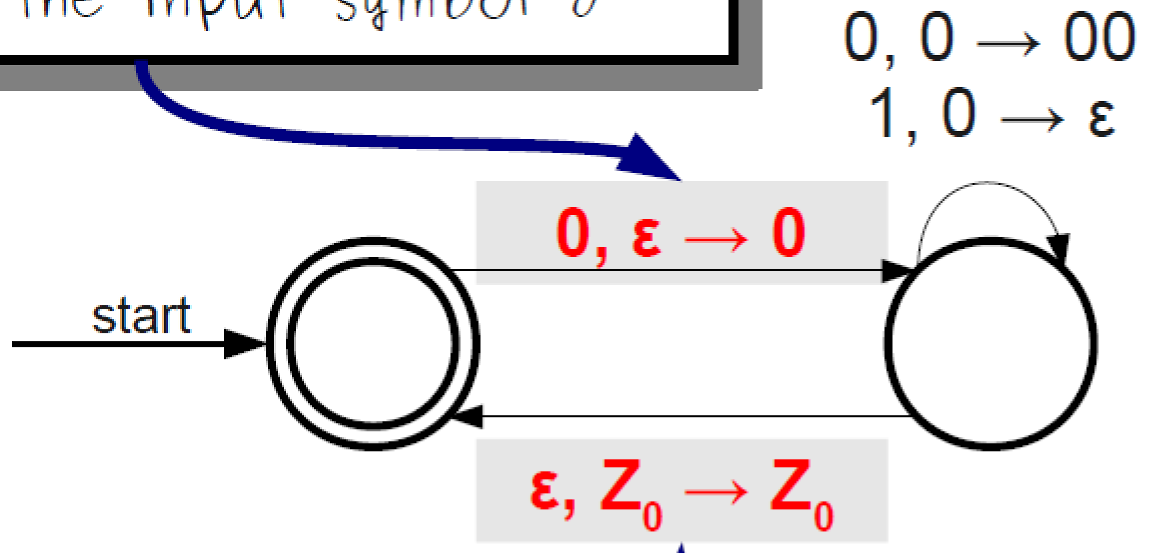
## Is this DPDA



$0, Z_0 \rightarrow 0Z_0$
$0, 0 \rightarrow 00$
$1, 0 \rightarrow \varepsilon$
start
$\varepsilon, Z_0 \rightarrow \varepsilon$

## Is this DPDA

This ε–transition is allowable because no other transitions in this state use the input symbol 0



This ε–transition is allowable because no other transitions in this state use the stack symbol Z₀.

## Why DPDA matter?

Because DPDAs are deterministic, they can be simulated efficiently:

- Keep track of the top of the stack.
- Store an **action/goto table** that says what operations to perform on the stack and what state to enter on each input/stack pair.
- Loop over the input, processing input/stack pairs until the automaton rejects or ends in an accepting state with all input consumed.

If we can find a DPDA for a CFL, then we can recognize strings in that language efficiently.

## Why DPDA matter?

*If we can find a DPDA for a CFL, then we can recognize strings in that language efficiently.*

Can we guarantee that we can always find a DPDA for a CFL?

# Deterministic PDA

Pushdown automata that we have already defined and discussed are nondeterministic by default, that is , there may be two or more moves involving the same combinations of state, input symbol, and top of the stock, and again, for some state and top of the stock the machine may either read and input symbol or make an $\in$- transition (without consuming any input).

In deterministic *PDA* , there is never a choice of move in any situation. This is handled by preventing the above mentioned two cases as described in the definition below.

Defnition : Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a *PDA* . Then *M* is deterministic if and only if both the following conditions are satisfied.

1. $\delta(q, a, X)$ has at most one element for any $q \in Q, a \in \Sigma \cup \{\in\}$, and $X \in \Gamma$ (this condition prevents multiple choice f any combination of $q, a \text{ and } X$ )
2. If $\delta(q, \in, X) \neq \phi$ and $\delta(q, a, X) = \phi$ for every $a \in \Sigma$

(This condition prevents the possibility of a choice between a move with or without an input symbol).

## The power of nondeterminism

When dealing with finite automata, there is no difference in the power of NFAs and DFAs.

However, when dealing with PDAs, there are CFLs that can be recognized by NPDAs that **cannot** be recognized by DPDAs.

Simple example: The language of palindromes.

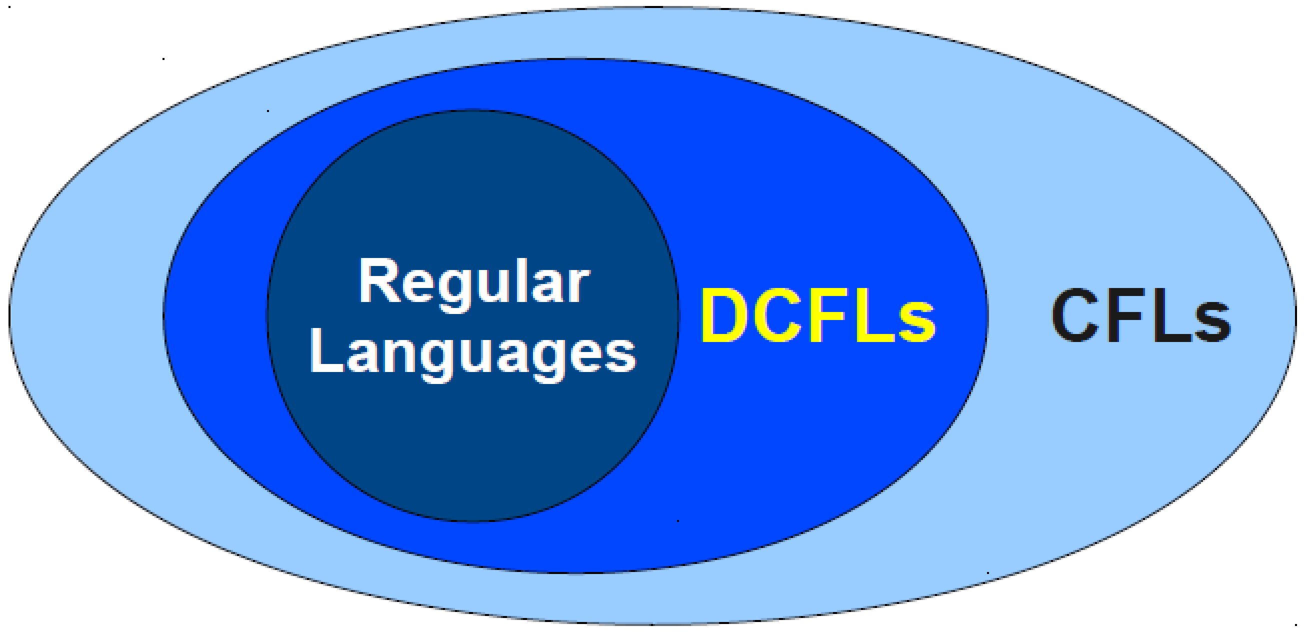
- How do you know when you've read half the string?

NPDAs are **more powerful** than DPDAs.

## Deterministic CFLs

- A context-free language L is called a **deterministic context-free language** (DCFL) if there is some DPDA that recognizes L.
- Not all CFLs are DCFLs, though many important ones are.
  - Balanced parentheses, most programming languages, etc.

Why are all regular languages DCFLs?



THANK YOU

*More on next class…*