Pushdown Automata

Tanjila Alam Sathi

Lecturer, CSE Department

Introduction

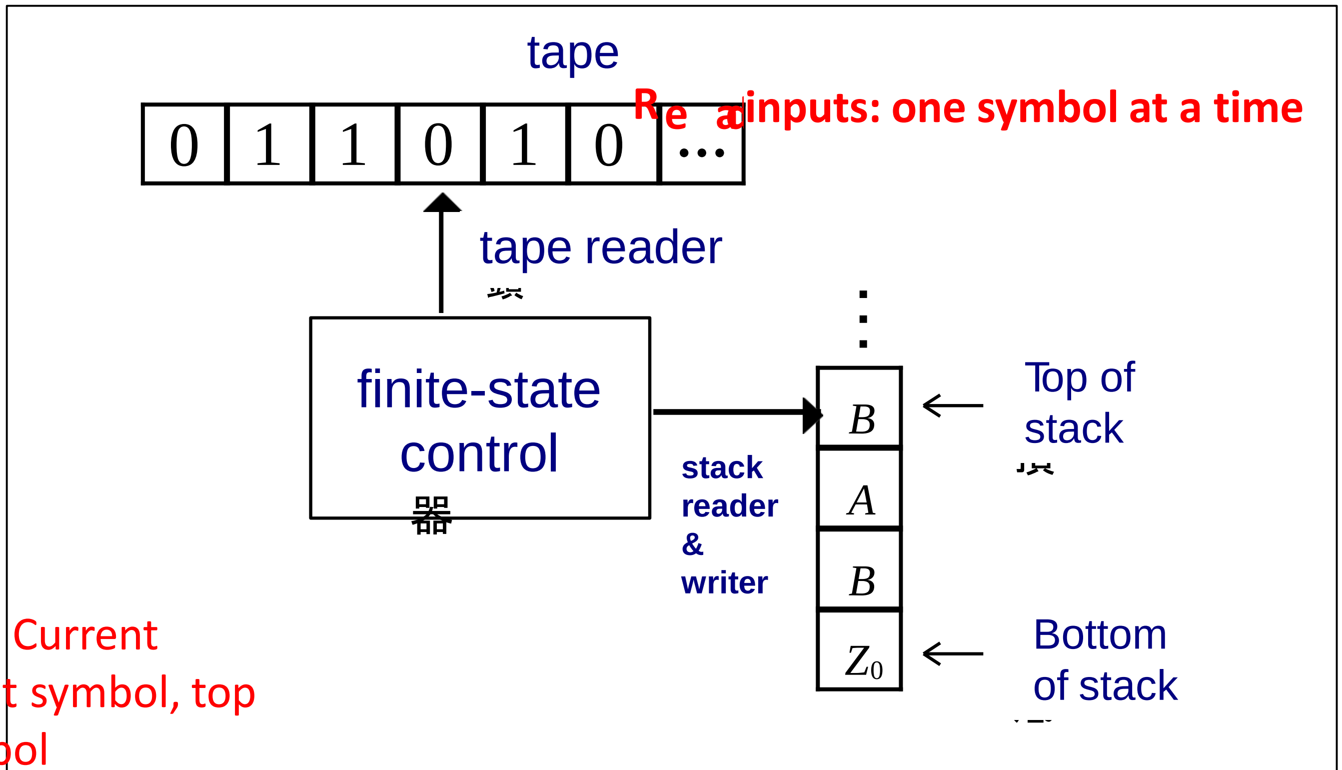
- CFL's may be accepted by pushdown automata (PDA's)
- A PDA is an ε-NFA with a stack.
- The stack can be read, pushed, and popped only at the top.
- Two different versions of PDA's
  - Accepting strings by "entering an accepting state"
  - Accepting strings by "emptying the stack"

Informal Introduction

- Advantage of the stack --- the stack can "remember" an *infinite* amount of information.
- Weakness of the stack --- the stack can only be read in a *first-in-last-out* manner.
- Therefore, it can accept languages (CFL) like $L_{wwr} = \{ww^R \mid w \text{ is in } (\mathbf{0} + \mathbf{1})^*\}$, but not languages (NCFL) like $\{a^n b^n c^n \mid n \geq 1\}$ [the set of strings consisting of equal groups of 0's, 1's, 2's]

Informal Introduction

- Graphical model



A graph model of a PDA

Transition: Current state, input symbol, top stack symbol

Informal Introduction

- The input string on the "tape" can only be read.
- But operations applied to the stack is complicated; we may replace the top symbol by any *string* ---
  - By a single symbol
  - By a string of symbols
  - Replace the top symbol on the stack by a sequence of zero or more symbols.
    - Zero symbols = "pop."
    - Many symbols = sequence of "pushes."

Example 6.1 Design a PDA to accept the language $L_{wwr} = \{ww^R \mid w \text{ is in } (\mathbf{0} + \mathbf{1})^*\}$.

- In start state q_0, copy input symbols onto the stack.
- At any time, *nondeterministically* guess **whether** the middle of ww^R is reached and enter q_1, *or* continue copying input symbols.
- In q_1, compare remaining input symbols with those on the stack one by one.

-if match, we consume input symbol, pop the stack, & proceed

-if do not match, guess wrong; branches dies

- If the stack can be so emptied, then the matching of w with w^R succeeds.

Formal Definition

- A PDA is a 7-tuple $\boldsymbol{P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)}$
  1. A finite set of *states* (Q, typically).
  2. An *input alphabet* (Σ, typically).
  3. A *stack alphabet* (Γ, typically).
  4. A *transition function* (δ, typically).
  5. A *start state* (q_0, in Q, typically).
  6. A *start symbol* (Z_0, in Γ, typically).
  7. A set of *final states* ($F \subseteq Q$, typically).

Conventions

- a, b, … are input symbols.
  - But sometimes we allow ϵ as a possible value.
- …, X, Y, Z are stack symbols.
- …, w, x, y, z are strings of input symbols.
- α, β,… are strings of stack symbols.

The Transition Function

- Takes three arguments:
  1. A state, in Q.
  2. An input, which is either a symbol in Σ or ϵ.
  3. A stack symbol in Γ.
- $\delta(q, a, Z)$ is a set of zero or more actions of the form (p, α).
  - p is a state; α is a string of stack symbols.

Actions of the PDA

- If $\delta(q, a, Z)$ contains (p, Y) among its actions, then one thing the PDA can do in state q, with *a* at the front of the input, and Z on top of the stack is:
  1. Change the state to p.
  2. Remove *a* from the front of the input (but *a* may be ϵ).
  3. Replace Z on the top of the stack by Y.

Example 6.2 Designing a PDA to accept the language $L_{wwr} = \{ww^R \mid w \text{ is in } (\mathbf{0} + \mathbf{1})^*\}$.

- Need a start symbol Z_0 of the stack and a 3rd state q_2 as the accepting state.
- **$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$** such that
  - $\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$, $\delta(q_0, 1, Z_0) = \{(q_0, 1Z_0)\}$

    (initial pushing steps with Z_0 to mark stack bottom)

    -in state q_0, the start symbol Z_0 at the top of the stack. Read first input & push it onto the stack, leaving Z_0 below to mark the bottom
  - $\delta(q_0, 0, 0) = \{(q_0, 00)\}$, $\delta(q_0, 0, 1) = \{(q_0, 01)\}$, $\delta(q_0, 1, 0) = \{(q_0, 10)\}$, $\delta(q_0, 1, 1) = \{(q_0, 11)\}$

    (continuing pushing)

Example 6.2 (cont'd

– $\delta(q_0, \varepsilon, Z_0) = \{(q_1, Z_0)\}$

(check if input is ε which is in L_{ww^R})

– $\delta(q_0, \varepsilon, 0) = \{(q_1, 0)\}$, $\delta(q_0, \varepsilon, 1) = \{(q_1, 1)\}$

(check the string's middle)

– $\delta(q_1, 0, 0) = \{(q_1, \varepsilon)\}$, $\delta(q_1, 1, 1) = \{(q_1, \varepsilon)\}$

(matching pairs)

– $\delta(q_1, \varepsilon, Z_0) = \{(q_2, Z_0)\}$

(entering final state)

How PDA Works?

- finite automaton with a data structure which will allow it to recognize strings of the form $a^n b^n$.
  - Match the a's with the b's.
  - We could use a counter for this, but thinking ahead a bit, there is a computer science way to do this.
- We shall allow the machine to build a pile of discs as it processes the a's in its input.
- Then it will unpile these disks as it passes over the b's.

How PDA Works?

```
place the input head on the leftmost
      input symbol

while symbol read = a
   advance head
   place disc on pile

while symbol read = b and pile contains discs
   advance head
   remove disc from pile

if input has been scanned
     and pile = empty then accept
```

Figure 1 - $a^n b^n$ Recognition Algorithm

How PDA Works?

- What happens? **aaabbb**.
- The machine reads the a's and builds a pile of three discs.
- Then it reads the b's and removes the discs from the pile one by one as each b is read.
- At this point it has finished the input and its pile is empty so it accepts.

How PDA Works?

- **aabbb,** it would place two discs on the pile and then remove them as it read the first two b's.
- Then it would leave the second while loop with one b left to read (since the pile was empty) and thus not accept.
- For aaabb it would end with one disk on the pile and not accept that input either.
- When given the input string aabbab, the machine would finish the second loop with ab yet to be read. **What happens?**

How PDA Works?

- We now have a new data structure (a pile) attached to our old friend, the finite automaton.
- Conventions:
  - The tape head advanced on each step,
  - Discs were placed on *top of the pile, and*
  - An empty pile means acceptance.

How PDA Works?

- Attempt something a bit more difficult.
- Why not try to recognize strings of the form **$w\#w^R$**
  - where w is a string over the alphabet {a, b} and w^R is the *reversal of* the string w? (Reversal is just turning the string around end for end.
  - **$abaa^R$ = aaba.**
  - Now we need to do some comparing, not just counting.

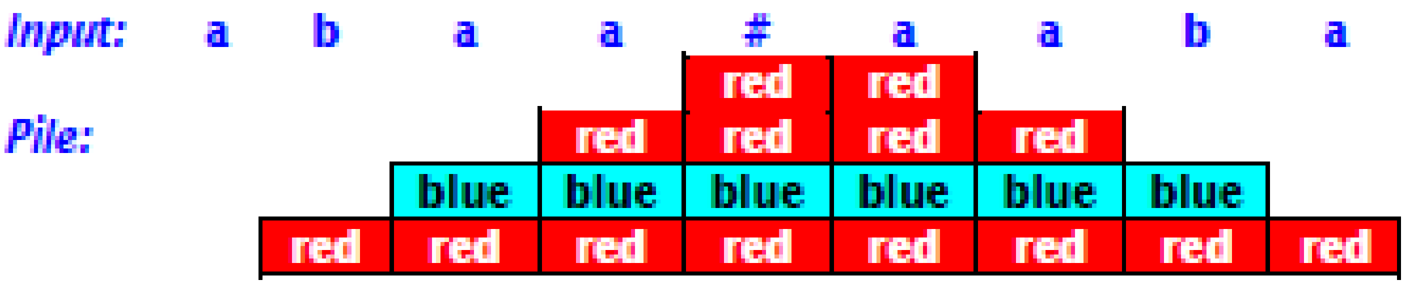
How PDA Works?

```
place input head upon leftmost input symbol

while symbol being scanned ≠ #
   if symbol scanned = a, put red disk on pile
   if symbol scanned = b, put blue disk on pile
   advance input head to next symbol

advance input head past #

repeat
  if (symbol scanned = a and red disk on pile)
     or (symbol scanned = b and blue disk on pile)
        then remove top disk; advance input head
until (pile is empty) or (no input remains)
      or (no disk removed)

if input has been read and pile is empty then accept
```

Figure 2 - Accepting Strings of the Form $w\#w^R$

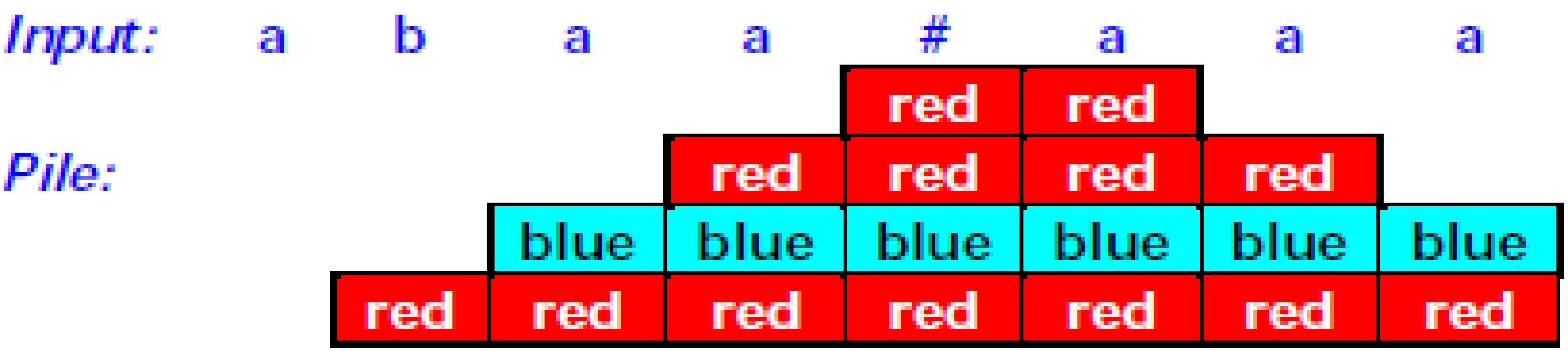
How PDA Works?

- **abaa#aaba**



- Right end: machine reads the a & removes the red disk from the stack.
- Since the stack is empty, it accepts
- The input was completely read and the pile was empty, the machine accepted

abaa#aaab What happens?



- machine stopped with ab yet to read & discs on the pile since it could not match an a with the blue disc.
- So, it rejected the input string.

Example: PDA

- Design a PDA to accept $\{0^n1^n \mid n \geq 1\}$.
- The states:
  - q = start state. We are in state q if we have seen only 0's so far.
  - p = we've seen at least one 1 and may now proceed only if the inputs are 1's.
  - f = final state; accept.

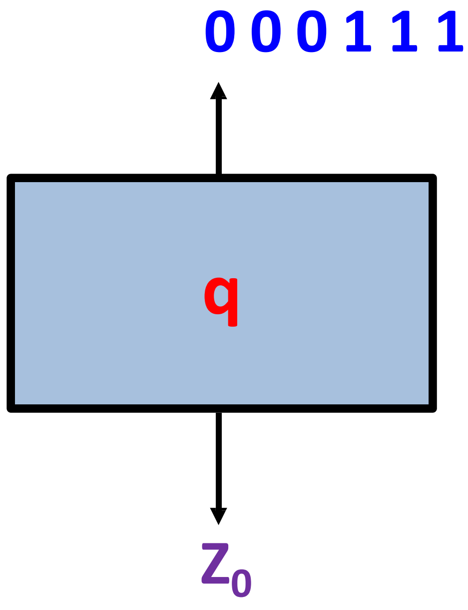
Example: PDA

- The stack symbols:
  - ❑Z_0 = start symbol. Also marks the bottom of the stack, so we know when we have counted the same number of 1's as 0's.
  - ❑X = marker, used to count the number of 0's seen on the input.

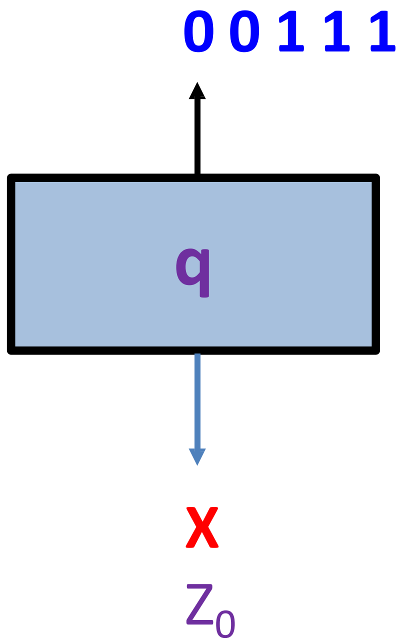
Example: PDA

- The transitions:
  - $\delta(q, 0, Z_0) = \{(q, XZ_0)\}$.
  - $\delta(q, 0, X) = \{(q, XX)\}$. These two rules cause one X to be pushed onto the stack for each 0 read from the input.
  - $\delta(q, 1, X) = \{(p, \epsilon)\}$. When we see a 1, go to state p and pop one X.
  - $\delta(p, 1, X) = \{(p, \epsilon)\}$. Pop one X per 1.
  - $\delta(p, \epsilon, Z_0) = \{(f, Z_0)\}$. Accept at bottom.

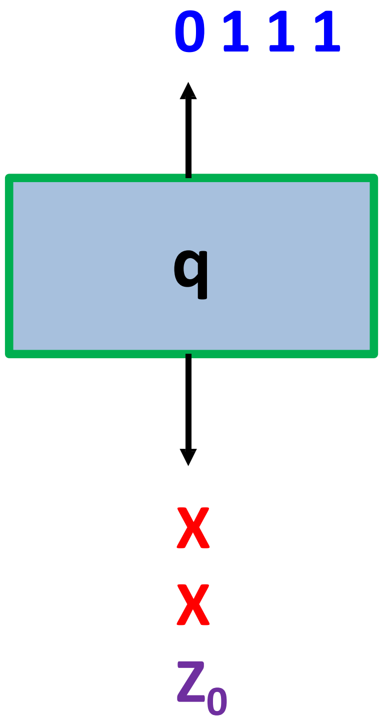
Actions of the Example PDA



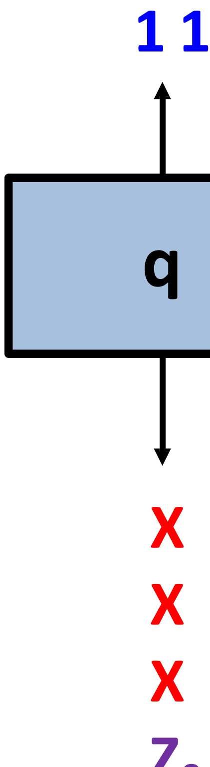
Actions of the Example PDA



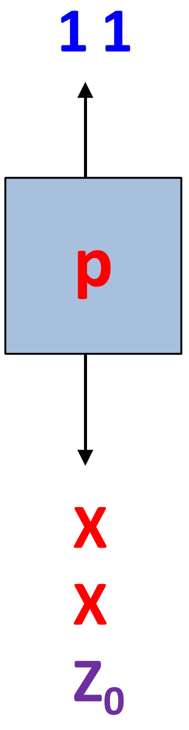
Actions of the Example PDA



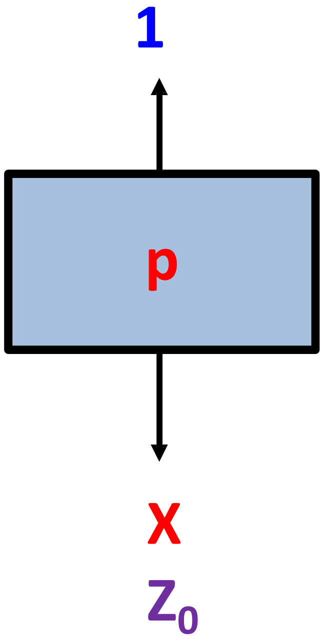
Actions of the Example PDA



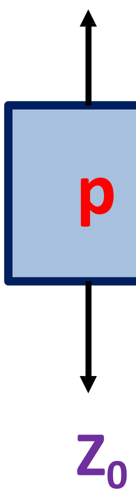
Actions of the Example PDA



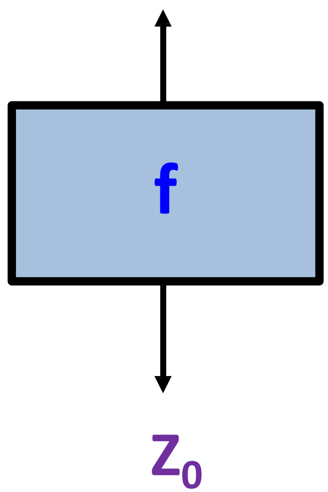
Actions of the Example PDA



Actions of the Example PDA



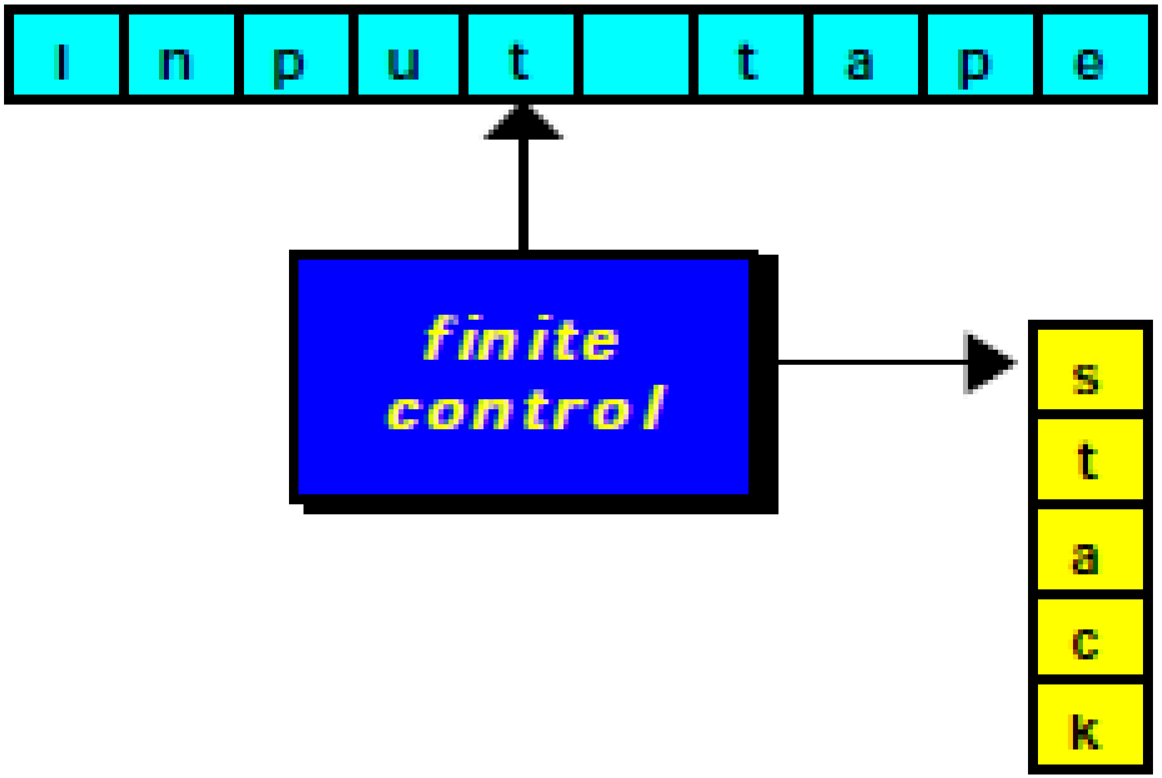
Actions of the Example PDA



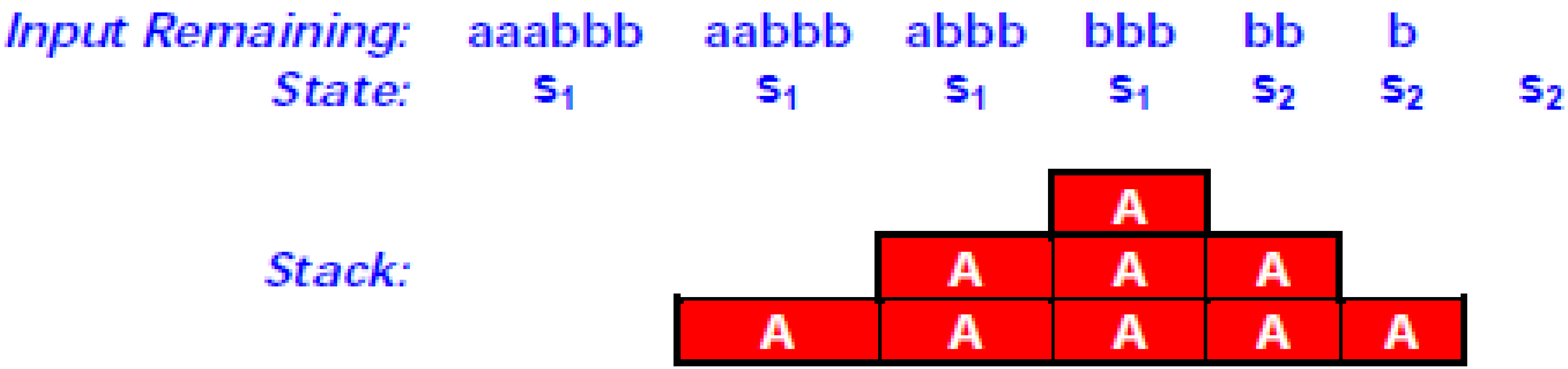
Pushdown automata-Why?

- All they are is finite automata with auxiliary storage devices called *stacks*.
- *A stack is merely* ***a pile***.
- ***symbols*** *are* normally placed on **stacks** rather than various colored discs.
- **Rules**:
- a) Symbols must always be placed upon the top of the stack.
- b) Only the top symbol of a stack can be read.
- c) No symbol other than the top one can be removed.
- **Push:** placing a symbol upon the stack
- **Pop:** *removing one* from the top of the stack

Pushdown Automata



- aaabbb


Input Remaining:
aaabbb
aabbb
abbb
bbb
bb
b
State:
s_1
s_1
s_1
s_1
s_2
s_2
s_2
Stack:
A

aaabb What happened?

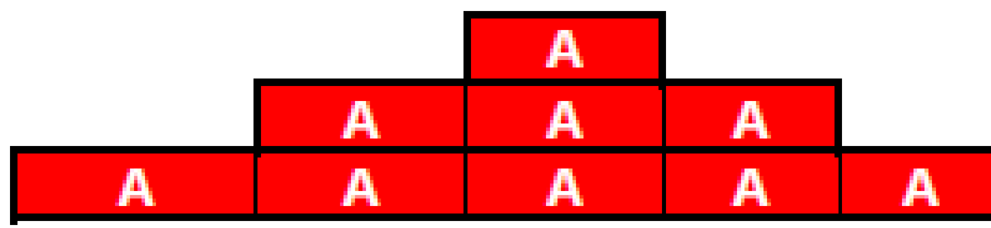
Input Remaining:	aaabb	aabb	abb	bb	b	
State:	s_1	s_1	s_1	s_1	s_2	s_2

Stack:

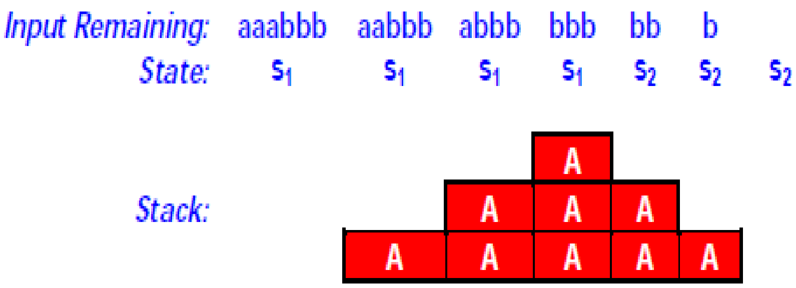
aaabba

Input Remaining:	aaabba	aabba	abba	bba	ba	a
State:	s_1	s_1	s_1	s_1	s_2	s_2

Stack:

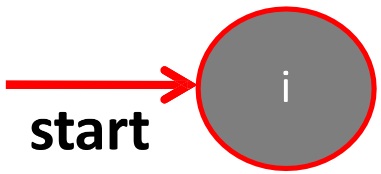


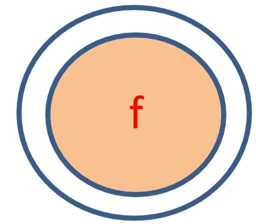
- A *configuration is a triple* *<s, x, α>*
  - *S: state*
  - *x: a* string over the input alphabet
  - α: a string over the stack alphabet.



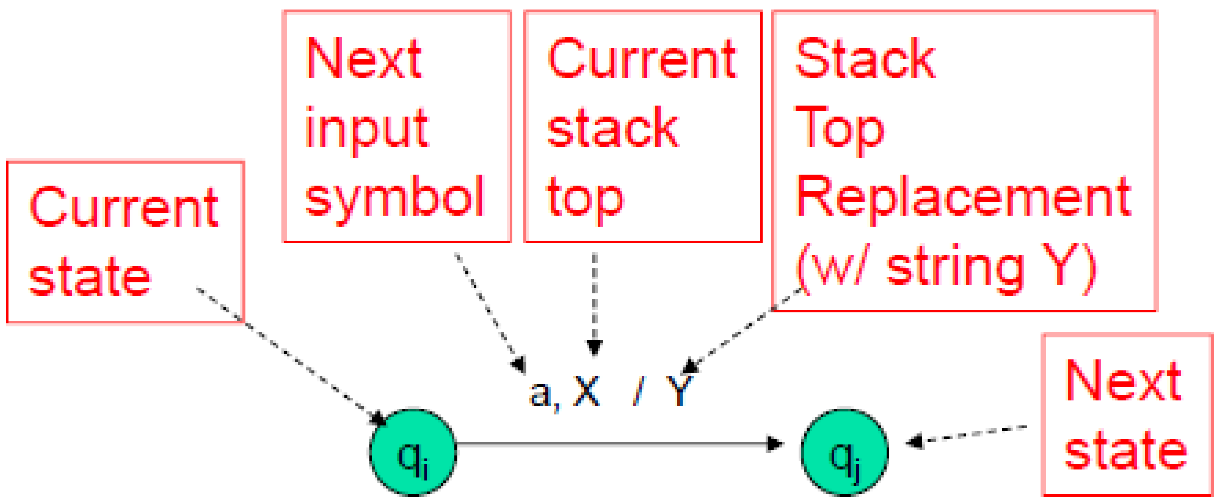
$$\langle s_1, aaabbb, \varepsilon \rangle \rightarrow \langle s_1, aabbb, A \rangle$$
$$\langle s_1, aabbb, A \rangle \rightarrow \langle s_1, abbb, AA \rangle$$
$$\langle s_1, abbb, AA \rangle \rightarrow \langle s_1, bbb, AAA \rangle$$
$$\langle s_1, bbb, AAA \rangle \rightarrow \langle s_2, bb, AA \rangle$$
$$\langle s_2, bb, AA \rangle \rightarrow \langle s_2, b, A \rangle$$
$$\langle s_2, b, A \rangle \rightarrow \langle s_2, \varepsilon, \varepsilon \rangle$$

PDA as a state diagram



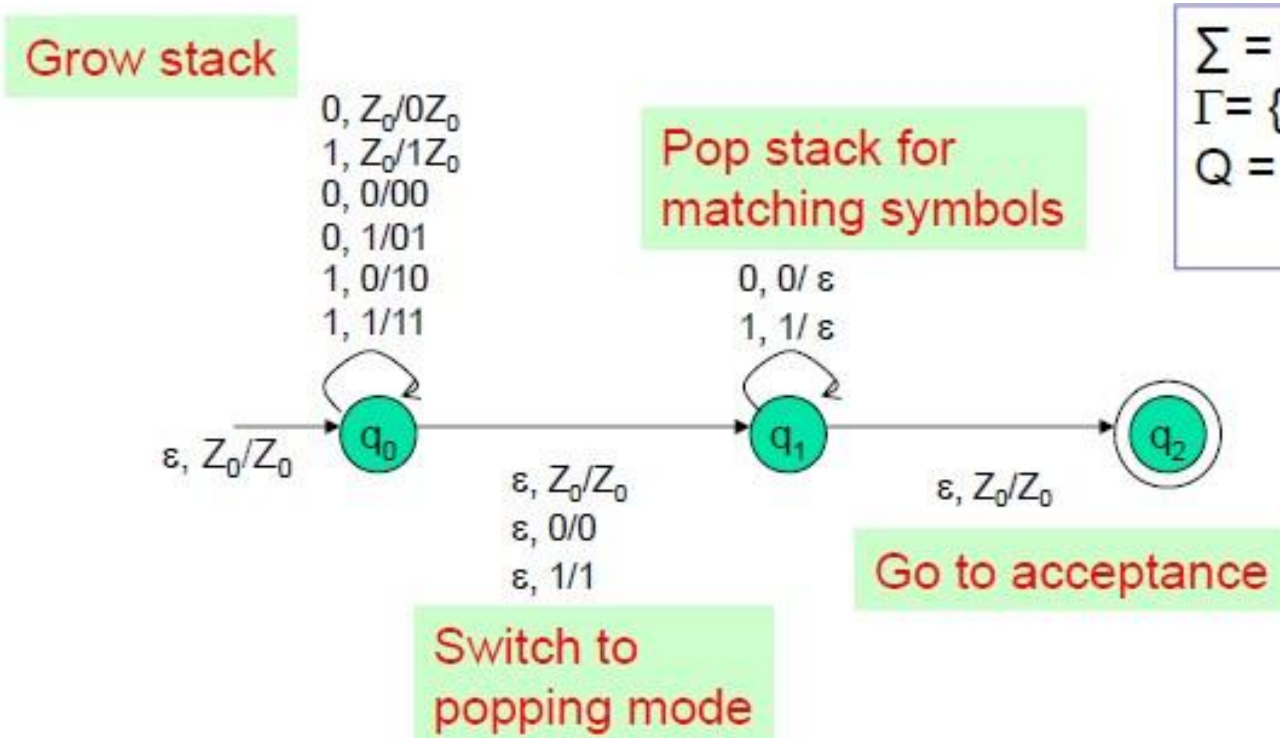


$$\delta(q_i, a, X) = \{(q_j, Y)\}$$



PDA for L_{wwr}: Transition Diagram

$\Sigma = \{0, 1\}$
$\Gamma = \{Z_0, 0, 1\}$
$Q = \{q_0, q_1, q_2\}$



This would be a non-deterministic PDA

PDA's Instantaneous Description (ID)

A PDA has a configuration at any given instance:

(q,w,y)

- q - current state
- w - remainder of the input (i.e., unconsumed part)
- y - current stack contents as a string from top to bottom of stack

---

If δ(q,a, X)={(p, A)} is a transition, then the following are also true:

- (q, a, X) |--- (p,ε,A)
- (q, aw, XB) |--- (p,w,AB)

|--- sign is called a "turnstile notation" and represents one move

|---* sign represents a sequence of moves

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 α, if $\delta(q, a, X)$ contains (p, β).
- Extend $\vdash$ to $\vdash^*$, meaning “zero or more moves,” by:
  - Basis: $\mathbf{I \vdash^* I}$
  - Induction: **If $\mathbf{I \vdash^* J}$ and $\mathbf{J \vdash K}$, then $\mathbf{I \vdash^* K}$**

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, $(q, 000111, Z_0) \vdash^* (f, \epsilon, Z_0)$

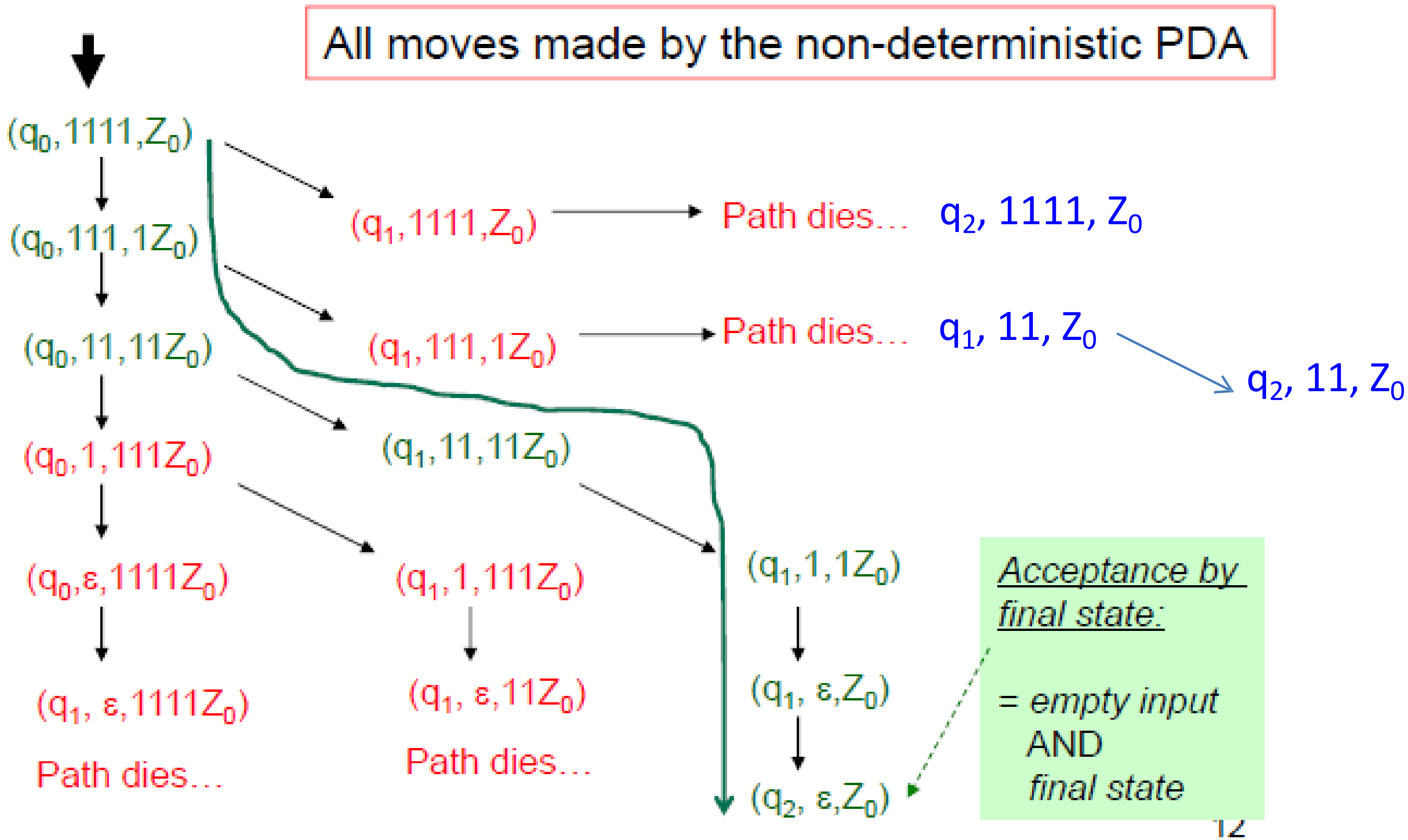
❑ What would happen on input 0001111?

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.

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

How does the PDA for L_{wwr} work on input “1111”?



z_0 is not at the top of stack. q_2 cannot be reached

Languages of a PDA

- **Acceptance by final state**: accepts input by consuming it & entering an accepting state.

- **Acceptance by empty stack**: set of strings that cause PDA to empty its stack, starting from initial ID

Acceptance by Final State

- ***PDAs that accept by final state:***
  - For a PDA P, the language accepted by P, denoted by L(P) by *final state*, is:
    - $\{w \mid (q_0,w,Z_0) \mid\text{---}^* (q,\varepsilon, A)\}$, s.t., $q \in F$

Checklist:
- input exhausted?
- in a final state?

- For some state q in F & any stack string A
- Starting in the initial ID with w waiting on the input, P consumes w from the input & enters an accepting state.

Acceptance by Empty Stack

- ***PDAs that accept by* empty stack*:***
  - For a PDA P, the language accepted by P, denoted by N(P) by *empty stack*, is:
    - $\{w \mid (q_0,w,Z_0) \vdash^* (q, \varepsilon, \varepsilon)\}$, for any $q \in Q$.

- For any state q.
- N (P) is the set of inputs w that P can consume & at the same time empty its stack.

From Empty Stack to Final State: $P_N \rightarrow P_F$

- **Theorem 6.9**

  If $L = N(P_N)$ for some PDA $P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0)$, then there is a PDA P_F **such that** $L = L(P_F)$.

  Proof. The idea is to use Fig. 6.4 below.

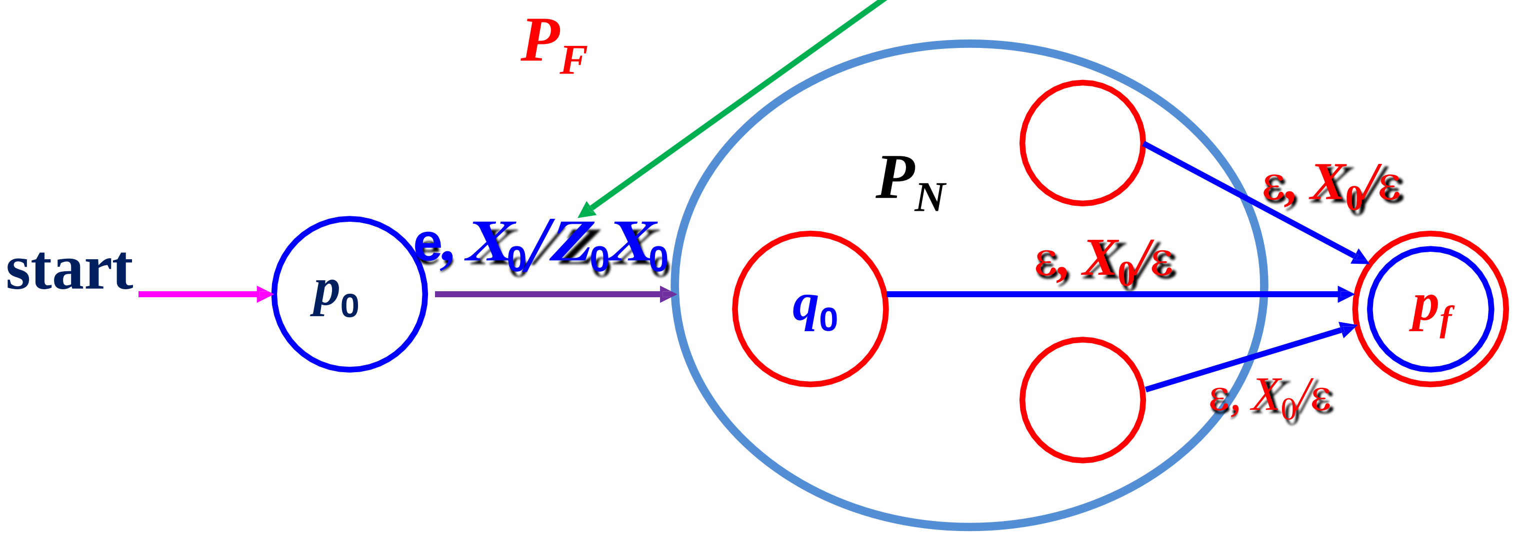


Fig. 6.4 P_F simulating P_N and accepts if P_N empties its stack

$P_N \rightarrow P_F$

- Use a new symbol X_0, which must not be a symbol of Γ;
  - X_0 is both the start symbol of $\mathbf{P_F}$ and
  - a marker on the bottom of the stack that lets us know when $\mathbf{P_N}$ has reached an empty stack.
- That is, if $\mathbf{P_F}$ sees X_0 on the top of its stack, then it knows that $\mathbf{P_N}$ would empty its stack on the same input.

$P_N \rightarrow P_F$

- New start state, p_0, whose sole function is to push $\mathbf{Z_0}$, the start symbol of $\mathbf{P_N}$, onto the top of the stack and enter state q_0, the start state of $\mathbf{P_N}$.
- Then P_F simulates P_N, until the stack of P_N is empty, which P_F detects because it sees X_0 on the top of the stack.
- Finally, we need another new state, p_f, which is the accepting state of $\mathbf{P_F}$; this PDA transfers to state p_f whenever it discovers that $\mathbf{P_N}$ would have emptied its stack.

$P_N \rightarrow P_F$

Define $\boldsymbol{P_F = (Q \cup \{p_0, p_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta_F, p_0, X_0, \{p_f\})}$
where δ_F is such that

- 1. $\delta_F(p_0, \varepsilon, X_0) = \{(q_0, Z_0X_0)\}$
  - In its start state, P_F makes a spontaneous transition to the start state of P_N, pushing its start symbol Z_0 onto the stack.
- 2. For all $q \in Q$, $a \in \Sigma$ or $a = \varepsilon$, and $Y \in \Gamma$, $\delta_F(q, a, Y)$ contains all the pairs in $\delta_N(q, a, Y)$.
- 3. $\delta_F(q, \varepsilon, X_0)$ contains (p_f, ε) for every state q in Q.

$P_N \rightarrow P_F$

- It can be proved that w is in $L(P_F)$ if and only if w is in $N(P_N)$
- (If) we are given that $(q_0, w, Z_0) \vdash_{P_N} (q, \varepsilon, \varepsilon)$ for some state q.
- Insert X_0 at the bottom of the stack and conclude $(q_0, w, Z_0X_0) \vdash_{P_N} (q, \varepsilon, X_0)$.
- Since by rule (2), P_F has all the moves of P_N, we may also conclude that $(q_0, w, Z_0X_0) \vdash_{P_F} (q, \varepsilon, X_0)$
- If we put this sequence of moves together with the initial & final moves from rules (1) & (3),
- $(p_0, w, X_0) \vdash_{P_F} (q_0, w, Z_0X_0) \vdash_{P_F} (q, \varepsilon, X_0) \vdash_{P_F} (p_f, \varepsilon, \varepsilon)$

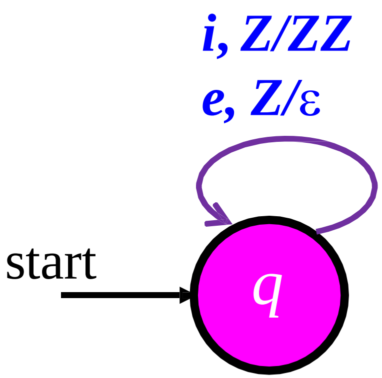
Example: Design a PDA which accepts the if/else errors by empty stack.

- Let *i* represents if; *e* represents else.
- The PDA is designed in such a way that

  if the number of *else* (#*else*) > the number of *if* (#*if*), then the stack will be emptied.

Example

- There is a problem whenever the number of else’s in any prefix exceeds the numbers of if’s, because then we cannot match each else against its previous if.
- Thus, we shall use a stack symbol Z to count the difference between the # of I’s seen so far and the # of e’s.



- when an "*if*" is seen, push a "*Z*";
- when an "*else*" is seen, pop a "*Z*";
- Since, we start with one Z on the stack, we actually follow the rule that if the stack is Z^n, then there have been n-1 more i's than e's.

- when (#*else*) > (#*if* + 1), the stack is emptied and the input sting is accepted.

❖ $$P_N = (\{q\}, \{i, e\}, \{Z\}, \delta_N, q, Z)$$

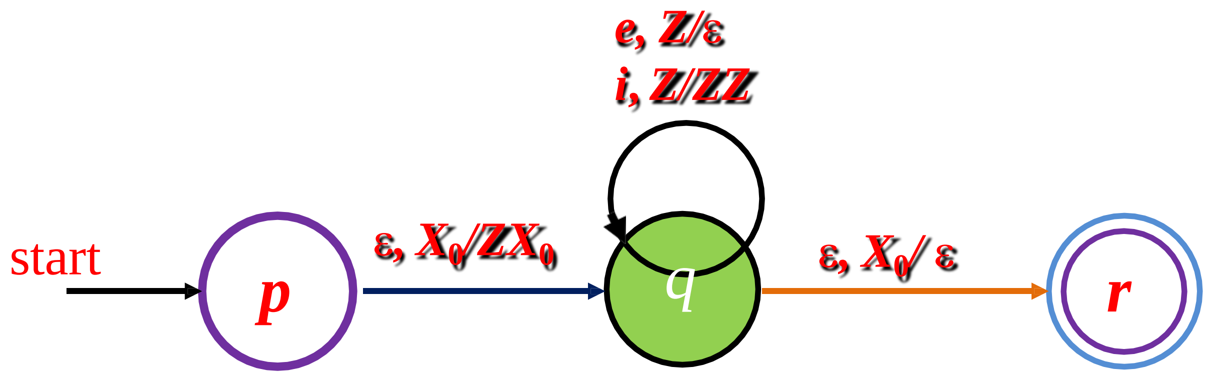
Example

- δ_N

1. $\delta_N(q, i, Z) = \{(q, ZZ)\}$. *Pushes a Z when see an i*
2. $\delta_N(q, e, Z) = \{(q, \varepsilon)\}$. *Pops a Z when see an e.*

- For example, for input string $w = iee$, the moves are:

  $(q, iee, Z) \vdash (q, ee, ZZ) \vdash (q, e, Z) \vdash (q, \varepsilon, \varepsilon)$ accept !



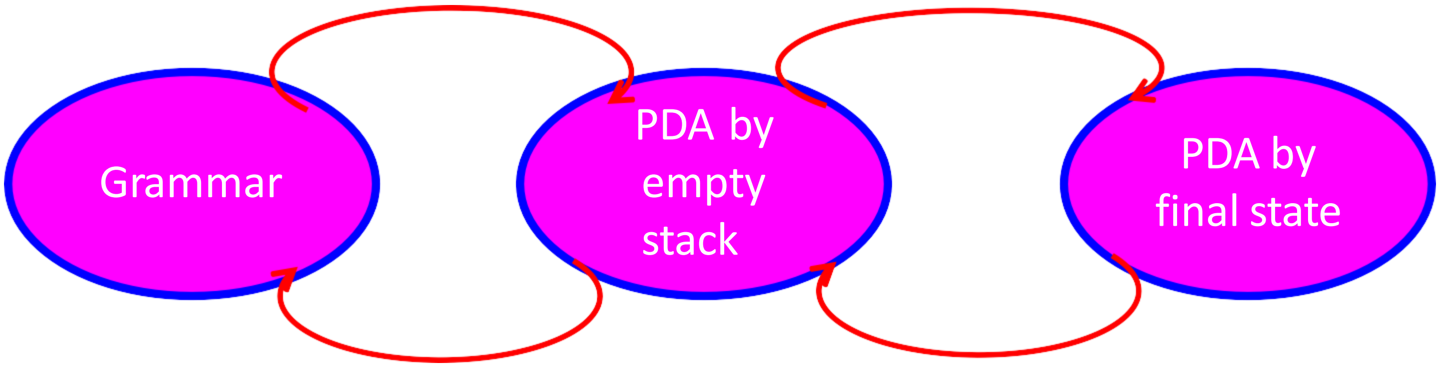
- A PDA *by final state* as follows:

$$P_F = (\{p, q, r\}, \{i, e\}, \{Z, X_0\}, \delta_F, p, X_0, \{r\})$$

- δ_N
  - ***1. (p, ε, X_0) = {q, ZX_0)}.*** **Starts P_F simulating P_N, with X_0 as a bottom-of-stack-marker**
  - ***2. (w, i, Z) = {(q, ZZ)}.*** **Pushes a Z when seen an i, it simulates P_N**
  - ***3. (q, e, Z) = {(q, e)}.*** **Pops a Z when seen an e; it simulates P_N**
  - ***4. (q, ε, X_0) = {r, ε, ε)} .*** **P_F accepts when the simulated P_N would have emptied its stack**

Equivalence of PDA's and CFG's

03 ways of defining CFL's



From Grammars to Pushdown Automata

- Given a CFG G, we construct a PDA that simulates the leftmost derivations of G.
- Any left-sentential form that is not a terminal string can be written as $x A \alpha$,
  - ✓ where A is the leftmost variable,
  - ✓ x is whatever terminals appear to its left,
  - ✓ α is the string of terminals & variables that appears to the right of A.
- We call $A\alpha$ the *tail* of this left-sentential form.
- If a left-sentential form consists of terminals only, then its tail is ε

From Grammars to Pushdown Automata

- Given a **CFG $G = (V, T, Q, S)$**, construct a PDA P that accepts $L(G)$ by empty stack in the following way:
  - $P = (\{q\}, T, V \cup T, \delta, q, S)$ where the transition function δ is defined by:
    - for each nonterminal/variable A,

$$\delta(q, \varepsilon, A) = \{(q, \beta) \mid A \rightarrow \beta \text{ is a production of } P\};$$

    - for each terminal a,

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

Example: Construct a PDA from the expression grammar :

$$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$
$$E \rightarrow I \mid E*E \mid E+E \mid (E)$$

- **Terminals**: **{a, b, 0, 1, (,), +, *} == 08 symbols**
- **Stack symbols: 08 symbols + I, E**
  - **Transition function, δ**
    - $\delta(q, \varepsilon, I) = \{(q, a), (q, b), (q, Ia), (q, Ib), (q, I0), (q, I1)\}$
    - $\delta(q, \varepsilon, E) = \{(q, \mathrm{I}), (q, E+E), (q, E*E), (q, (E))\}$
    - $\delta(q, \mathrm{a}, \mathrm{a}) = \{(q, \varepsilon)\}$; $\delta(q, \mathrm{b}, \mathrm{b}) = \{(q, \varepsilon)\}$; $\delta(q, 0, 0) = \{(q, \varepsilon)\}$;

      $\delta(q, 1, 1) = \{(q, \varepsilon)\}$; $\delta(q, (, () = \{(q, \varepsilon)\}$; $\delta(q,),)) = \{(q, \varepsilon)\}$;

      $\delta(q, +, +) = \{(q, \varepsilon)\}$; $\delta(q, *, *) = \{(q, \varepsilon)\}$

From PDA to Grammars

- **Theorem 6.14**

  Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ be a PDA. Then there is a context-free grammar G such that $L(G) = N(P)$.

Proof. Construct $G = (V, T, P, S)$ where the set of nonterminals consists of:

- ✓the special **symbol *S*** as the start symbol;
- ✓all symbols of the form **[*pXq*]** where ***p*** and ***q*** are states in Q and ***X*** is a stack symbol in Γ.

From PDA to Grammars

The productions of G are as follows.

(a) For all states p, G has the production $S \to [q_0 Z_0 p]$.

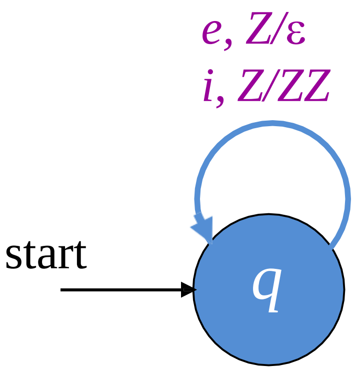
(b) Let $\delta(q, a, X)$ contain the pair $(r, Y_1 Y_2 \ldots Y_k)$, where

- a is either a symbol in Σ or $a = \varepsilon$;
- k can be any number, including 0, in which case the pair is (r, ε).

Then for all lists of states $r_1, r_2, \ldots, r_k$, G has the production

$$[qXr_k] \to a[rY_1r_1][r_1Y_2r_2]\ldots[r_{k-1}Y_kr_k].$$

Example 6.15 --- Convert the PDA of to a grammar.



Nonterminals include only two symbols, S and $[qZq]$.

Productions:

1. $S \rightarrow [qZq]$ (for the start symbol S);

2. $[qZq] \rightarrow i[qZq][qZq]$ (from $(q, ZZ) \in \delta_N(q, i, Z)$)

3. $[qZq] \rightarrow e$ (from $(q, \varepsilon) \in \delta_N(q, e, Z)$)

From PDA to Grammars

If we replace $[qZq]$ by a simple symbol A, then the productions become

1. $S \rightarrow A$
2. $A \rightarrow iAA$
3. $A \rightarrow e$

Obviously, these productions can be simplified to be

1. $S \rightarrow iSS$
2. $S \rightarrow e$

And the grammar may be written simply as

$$G = (\{S\}, \{i, e\}, \{S \rightarrow iSS \mid e\}, S)$$