Pushdown Automata

CSE 211 (Theory of Computation)

Atif Hasan Rahman

Assistant Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering & Technology

Adapted from slides by

Dr. Muhammad Masroor Ali

- The context free languages have a type of automaton that defines them.
- This automaton is called “pushdown automaton.”
- It is an extension of the nondeterministic finite automaton with ϵ -transitions.
- The pushdown automaton is essentially an ϵ -NFA with the addition of a stack.
- The stack can be read, pushed, and popped only at the top, just like the “stack” data structure.

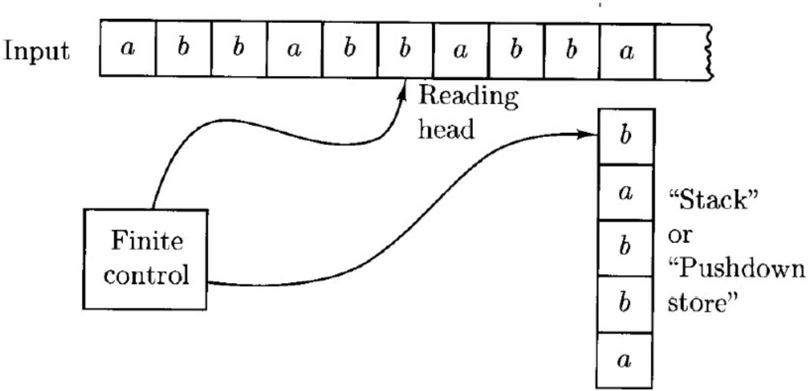
Languages which can not be Recognized by an FA

- Consider $\left\{ww^R : w \in \{a, b\}^*\right\}$.
- It is context-free, since it is generated by the grammar with rules $S \to aSa$, $S \to bSb$, and $S \to \epsilon$.
- It would seem that any device that recognizes the strings in this language
  - by reading them from left to right must "remember" the first half of the input string,
  - so that it can check it — in reverse order — against the second half of the input.

Languages which can . . . an FA

- It is not surprising that this function cannot be performed by a finite automaton.
- If, however, the machine is,
  - capable of accumulating its input string as it is read,
  - appending symbols one at a time to a stored string,
  - then it could nondeterministically guess when the center of the input has been reached,
  - and thereafter check the symbols off from its memory one at a time.
- The storage device need not be a general-purpose one.
- A sort of “stack” or “pushdown store,” allowing read and write access only to the top symbol, would do nicely.

Languages which can . . . an FA



Languages which can . . . an FA

(()(()))(()(()))

- To take another example, the set of strings of balanced parentheses is also nonregular.
- However, computer programmers are familiar with a simple algorithm for recognizing this language:
    - Start counting at zero,
    - add one for every left parenthesis,
    - and subtract one for every right parenthesis.
    - If the count either goes negative at any time, or ends up different from zero, then the string should be rejected as unbalanced;
    - otherwise it should be accepted.
- Now, a counter can be considered as a special case of a stack, on which only one kind of symbol can be written.

Informal Introduction

- The pushdown automaton is in essence a ***nondeterministic*** finite automaton with ϵ -transitions permitted.
- It has one additional capability: a stack on which it can store a string of "stack symbols."
- The presence of a stack means that, unlike the finite automaton, the pushdown automaton can "remember" an infinite amount of information.
- However, unlike a general purpose computer, which also has the ability to remember arbitrarily large amounts of information, the pushdown automaton can only access the information on its stack in a last-in-first-out way.

Informal Introduction

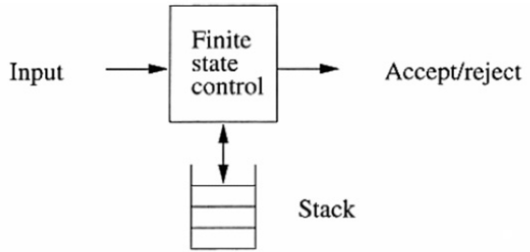
- As a result, there are languages that could be recognized by some computer program, but are not recognizable by any pushdown automaton.
- In fact, pushdown automata recognize all and only the context-free languages.
- There are many languages that are context-free, including some we have seen that are not regular languages.
- There are also some simple-to-describe languages that are not context-free.
- An example of a non-context-free language is $\{0^n1^n2^n \mid n \geq 1\}$, the set of strings consisting of equal groups of 0's, 1's, and 2's.

Informal Introduction

- As a result, there are languages that could be recognized by some computer program, but are not recognizable by any pushdown automaton.
- In fact, pushdown automata recognize all and only the context-free languages.
- There are many languages that are context-free, including some we have seen that are not regular languages.
- There are also some simple-to-describe languages that are not context-free.
- An example of a non-context-free language is $\{0^n1^n2^n \mid n \geq 1\}$, the set of strings consisting of equal groups of 0's, 1's, and 2's.

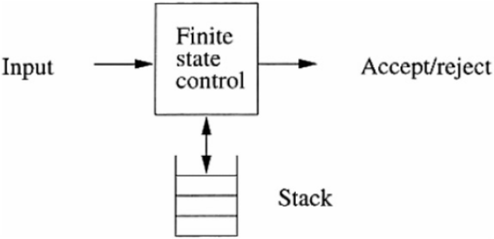
Informal Introduction

We can view the pushdown automaton informally as the device suggested in figure.



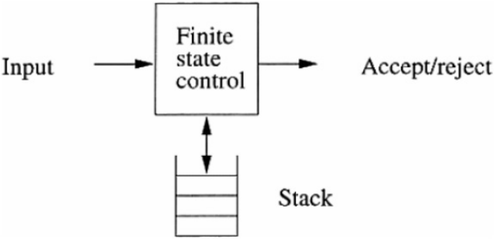
A pushdown automaton is essentially a finite automaton with a stack data structure

Informal Introduction



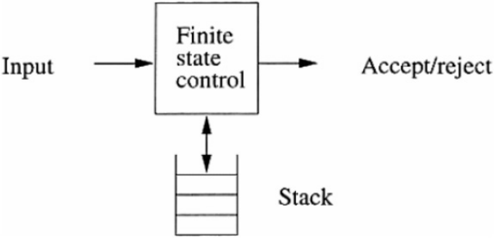
- A “finite-state control” reads inputs, one symbol at a time.

Informal Introduction



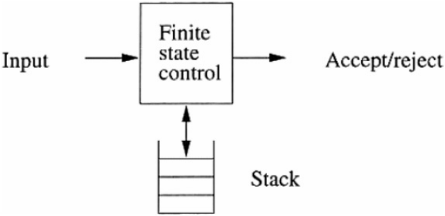
- The pushdown automaton observes the symbol at the top of the stack.
- It bases its transition on its current state, the input symbol, and the symbol at the top of stack.

Informal Introduction



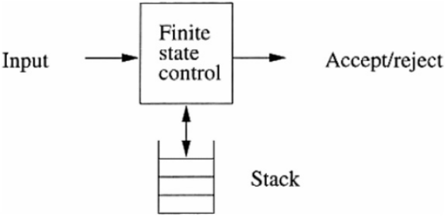
- Alternatively, it may make a spontaneous transition, using ϵ as its input instead of an input symbol.

Informal Introduction



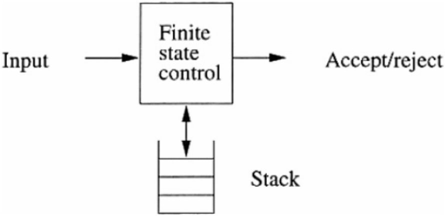
In one transition, the pushdown automaton:

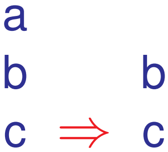
1. Consumes from the input the symbol that it uses in the transition.
   - If ϵ is used for the input, then no input symbol is consumed.



In one transition, the pushdown automaton:

2. Goes to a new state, which may or may not be the same as the previous state.

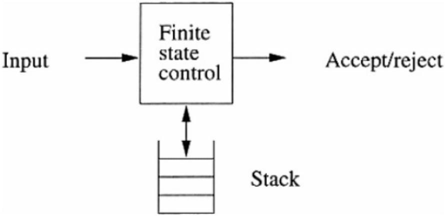




In one transition, the pushdown automaton:

3. Replaces the symbol at the top of the stack by any string.
   - The string could be ϵ , which corresponds to a pop of the stack.

Informal Introduction

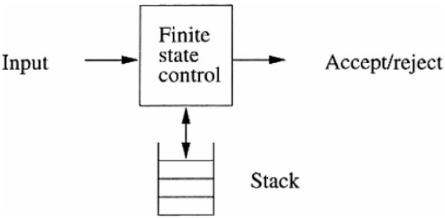


a		a
b		b
c	$\Rightarrow$	c

In one transition, the pushdown automaton:

3. Replaces the symbol at the top of the stack by any string.
   - It could be the same symbol that appeared at the top of the stack previously; i.e., no change to the stack is made.

Informal Introduction

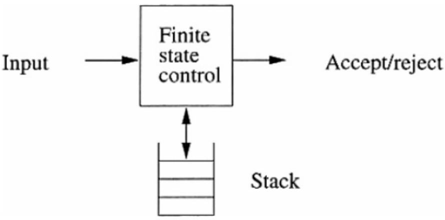


a		d
b		b
c	$\Rightarrow$	c

In one transition, the pushdown automaton:

3. Replaces the symbol at the top of the stack by any string.
   - It could also replace the top stack symbol by one other symbol.
   - This in effect changes the top of the stack but does not push or pop it.

Informal Introduction



		e
a		d
b		b
c	$\Rightarrow$	c

In one transition, the pushdown automaton:

3. Replaces the symbol at the top of the stack by any string.
   - Finally, the top stack symbol could be replaced by two or more symbols.
   - This has the effect of (possibly) changing the top stack symbol, and then pushing one or more new symbols onto the stack.

Example

- Let us consider the language

$$l_{wwr} = \left\{ ww^R \mid w \text{ is in } (\mathbf{0} + \mathbf{1})^* \right\}$$

- This language, often referred to as "*w*-*w*-reversed," is the even-length palindromes over alphabet $\{0, 1\}$.
- It is a CFL, generated by the grammar G_{pal}, with the productions $P \rightarrow 0$ and $P \rightarrow 1$ omitted.

$$
\begin{array}{llll}
1. & P & \rightarrow & \epsilon \\
\cancel{2.} & \cancel{P} & \cancel{\rightarrow} & \cancel{0} \\
\cancel{3.} & \cancel{P} & \cancel{\rightarrow} & \cancel{1} \\
4. & P & \rightarrow & 0P0 \\
5. & P & \rightarrow & 1P1
\end{array}
$$

Figure 5.1: A context-free grammar for palindromes

We can design an informal pushdown automaton accepting l_{wwr}, as follows.

1. Start in a state q_0 that represents a "guess" that we have not yet seen the middle.
   - i.e., we have not seen the end of the string w that is to be followed by its own reverse.
   - While in state q_0, we read symbols and store them on the stack, by pushing a copy of each input symbol onto the stack, in turn.

$$\underbrace{0101\overset{\downarrow}{1}011}_{w}\ \underbrace{11011010}_{w^R}$$

We are in q_0

Stack contains

1
1
0
1
0

We can design an informal pushdown automaton accepting l_{wwr}, as follows.

2. At any time, we may guess that we have seen the middle.
   - i.e., the end of w.
   - At this time, w will be on the stack, with the right end of w at the top and the left end at the bottom.
   - We signify this choice by spontaneously going to state q_1.
   - Since the automaton is nondeterministic, we actually make both guesses:
     - we guess we have seen the end of w,
     - but we also stay in state q_0 and continue to read inputs and store them on the stack.

$$\underbrace{01011011}_{w}\downarrow\underbrace{11011010}_{w^R}$$

Stack contains
$$\begin{array}{|c|}1\\1\\0\\1\\1\\0\\1\\0\\\hline\end{array}$$

We switch to q_1

We can design an informal pushdown automaton accepting l_{wwr}, as follows.

3. Once in state q_1, we compare input symbols with the symbol at the top of the stack.
    - If they match, we consume the input symbol, pop the stack, and proceed.
    - If they do not match, we have guessed wrong; our guessed w was not followed by w^R.
    - This branch dies, although other branches of the nondeterministic automaton may survive and eventually lead to acceptance.

$$\underbrace{01011011}_{w}\ \underbrace{\overset{\downarrow}{1}1011010}_{w^R}$$

Stack contains

1
0
1
1
0
1
0

We are in q_1

We can design an informal pushdown automaton accepting l_{wwr}, as follows.

4. If we empty the stack, then we have indeed seen some input w followed by w^R.
   - We accept the input that was read up to this point.

$$\underbrace{01011011}_{w}\,\underbrace{11011010}_{w^R}$$

We are in q_1

Stack contains $\boxed{0}$

The Formal Definition of Pushdown Automata

- Our formal notation for a pushdown automaton (PDA) involves seven components.
- We write the specification of a PDA P as follows:

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

The components have the following meanings:

Q: A finite set of *states*, like the states of a finite automaton.

Σ: A finite set of *input symbols*, also analogous to the corresponding component of a finite automaton.

Γ: A finite *stack alphabet*.

- This component, which has no finite-automaton analog, is the set of symbols that we are allowed to push onto the stack.

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

The components have the following meanings:

Q: A finite set of *states*, like the states of a finite automaton.

Σ: A finite set of *input symbols*, also analogous to the corresponding component of a finite automaton.

Γ: A finite *stack alphabet*.

- This component, which has no finite-automaton analog, is the set of symbols that we are allowed to push onto the stack.

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

The components have the following meanings:

Q: A finite set of *states*, like the states of a finite automaton.

Σ: A finite set of *input symbols*, also analogous to the corresponding component of a finite automaton.

Γ: A finite *stack alphabet*.

- This component, which has no finite-automaton analog, is the set of symbols that we are allowed to push onto the stack.

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

δ: The transition function.

- As for a finite automaton, δ governs the behavior of the automaton.
- Formally, δ takes as argument a triple $\delta(q, a, X)$, where:
  1. q is a state in Q.
  2. a is either an input symbol in Σ or $a = \epsilon$, the empty string, which is assumed not to be an input symbol.
  3. X is a stack symbol, that is, a member of Γ.

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

$$\delta(q, a, X) \text{ is set of } (p, \gamma)$$

- The output of δ is a finite set of pairs (p, γ), where
  - p is the new state,
  - and γ is the string of stack symbols that replaces X at the top of the stack.
- For instance,
  - if $\gamma = \epsilon$, then the stack is popped,
  - if $\gamma = X$, then the stack is unchanged,
  - and if $\gamma = YZ$, then X is replaced by Z, and Y is pushed onto the stack.

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

q_0: The *start state*.

- The PDA is in this state before making any transitions.

Z_0: The *start symbol*.

- Initially, the PDA's stack consists of one instance of this symbol, and nothing else.

F: The set of *accepting states*, or *final states*.

Example

- Let us design a PDA P to accept the language l_{ww^R}.
- There are a few details that we need to understand in order to manage the stack properly.
- We shall use a stack symbol Z_0 to mark the bottom of the stack.
- We need to have this symbol present so that, after we pop w off the stack and realize that we have seen ww^R on the input, we still have something on the stack to permit us to make a transition to the accepting state, q_2.

Example-continued

The three states:

q_0: We are yet to see the midpoint of the string.

$$\underbrace{0\overset{\downarrow}{1}011011}_{w}\underbrace{11011010}_{w^R}$$

We remain in q_0

q_1: We have just seen the midpoint of the string/crossed the midpoint of the string.

$$\underbrace{01011011}_{w}\downarrow\underbrace{11011010}_{w^R}$$

We switch from q_0 to q_1

q_2: We are finished with the string and it is acceptable.

$$\underbrace{01011011}_{w}\underbrace{11011010}_{w^R}\downarrow$$

We switch from q_1 to q_2

Thus, our PDA for l_{wwr} can be described as

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

where δ is defined by the following rules:

1. i) $\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$ and
   ii) $\delta(q_0, 1, Z_0) = \{(q_0, 1Z_0)\}$.

- This rule applies initially, when we are in state q_0 and we see the start symbol Z_0 at the top of the stack.
- We read the first input, and push it onto the stack, leaving Z_0 below to mark the bottom.

$$\underbrace{01011011}_{w}\underbrace{11011010}_{w^R}$$

We are in q_0

$$\begin{array}{|c|} Z_0 \\ \hline \end{array} \Rightarrow \begin{array}{|c|} 0 \\ Z_0 \\ \hline \end{array}$$

Thus, our PDA for l_{wwr} can be described as

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

where δ is defined by the following rules:

2.
   i) $\delta(q_0, 0, 0) = \{(q_0, 00)\}$,
   ii) $\delta(q_0, 0, 1) = \{(q_0, 01)\}$,
   iii) $\delta(q_0, 1, 0) = \{(q_0, 10)\}$, and
   iv) $\delta(q_0, 1, 1) = \{(q_0, 11)\}$.

- These four similar rules allow us to stay in state q_0 and read inputs, pushing each onto the top of the stack and leaving the previous top stack symbol alone.

$$\underbrace{01\overset{\downarrow}{0}11011}_{w}\underbrace{11011010}_{w^R}$$

We remain in q_0

$$\begin{array}{|c|}1\\0\\Z_0\\\hline\end{array} \Rightarrow \begin{array}{|c|}0\\1\\0\\Z_0\\\hline\end{array}$$

Thus, our PDA for l_{wwr} can be described as

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

where δ is defined by the following rules:

3. i) $\delta(q_0, \epsilon, Z_0) = \{(q_1, Z_0)\}$,
   ii) $\delta(q_0, \epsilon, 0) = \{(q_1, 0)\}$, and
   iii) $\delta(q_0, \epsilon, 1) = \{(q_1, 1)\}$.

- These three rules allow P to go from state q_0 to state q_1 spontaneously (on ϵ input), leaving intact whatever symbol is at the top of the stack.

$$\underbrace{01011011}_{w}\downarrow\underbrace{11011010}_{w^R}$$

We switch from q_0 to q_1

$$\begin{array}{|c|}1\\1\\0\\1\\1\\0\\1\\0\\Z_0\\ \hline\end{array} \Rightarrow \begin{array}{|c|}1\\1\\0\\1\\1\\0\\1\\0\\Z_0\\ \hline\end{array}$$

Thus, our PDA for l_{wwr} can be described as

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

where δ is defined by the following rules:

4. i) $\delta(q_1, 0, 0) = \{(q_1, \epsilon)\}$, and
   ii) $\delta(q_1, 1, 1) = \{(q_1, \epsilon)\}$.

- Now, in state q_1, we can match input symbols against the top symbols on the stack, and pop when the symbols match.

$$\underbrace{01011011}_{w}\underbrace{1101\overset{\downarrow}{1}010}_{w^R}$$

We remain in q_1

$$\begin{array}{|c|}1\\0\\1\\0\\Z_0\\\hline\end{array} \Rightarrow \begin{array}{|c|}0\\1\\0\\Z_0\\\hline\end{array}$$

Thus, our PDA for l_{wwr} can be described as

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

where δ is defined by the following rules:

5. ⓘ $\delta(q_1, \epsilon, Z_0) = \{(q_2, Z_0)\}$.
   - Finally, if we expose the bottom-of-stack marker Z_0 and we are in state q_1, then we have found an input of the form ww^R.
   - We go to state q_2 and accept.

$\underbrace{01011011}_{w}\underbrace{11011010}_{w^R}\downarrow$

We switch from q_1 to q_2

$\boxed{Z_0} \Rightarrow \boxed{Z_0}$

No “Mixing and Matching”

- There may be several pairs that are options for a PDA in some situation.
- For instance, suppose $\delta\,(q, a, X) = \{(p, YZ), (r, \epsilon\,)\}$.
- When making a move of the PDA, we have to choose one pair in its entirety.
- We cannot pick a state from one and a stack-replacement string from another.
- Thus, in state q, with X on the top of the stack, reading input a, we could go to state p and replace X by YZ.
- Or we could go to state r and pop X.
- However, we cannot go to state p and pop X.
- And we cannot go to state r and replace X by YZ.

No “Mixing and Matching”

- There may be several pairs that are options for a PDA in some situation.
- For instance, suppose $\delta(q, a, X) = \{(p, YZ), (r, \epsilon)\}$.
- When making a move of the PDA, we have to choose one pair in its entirety.
- We cannot pick a state from one and a stack-replacement string from another.
- Thus, in state q, with X on the top of the stack, reading input a, we could go to state p and replace X by YZ.
- Or we could go to state r and pop X.
- However, we cannot go to state p and pop X.
- And we cannot go to state r and replace X by YZ.

No “Mixing and Matching”

- There may be several pairs that are options for a PDA in some situation.
- For instance, suppose $\delta\,(q, a, X) = \{(p, YZ), (r, \epsilon\,)\}$.
- When making a move of the PDA, we have to choose one pair in its entirety.
- We cannot pick a state from one and a stack-replacement string from another.
- Thus, in state q, with X on the top of the stack, reading input a, we could go to state p and replace X by YZ.
- Or we could go to state r and pop X.
- However, we cannot go to state p and pop X.
- And we cannot go to state r and replace X by YZ.

No "Mixing and Matching"

- There may be several pairs that are options for a PDA in some situation.
- For instance, suppose $\delta(q, a, X) = \{(p, YZ), (r, \epsilon)\}$.
- When making a move of the PDA, we have to choose one pair in its entirety.
- We cannot pick a state from one and a stack-replacement string from another.
- Thus, in state q, with X on the top of the stack, reading input a, we could go to state p and replace X by YZ.
- Or we could go to state r and pop X.
- However, we cannot go to state p and pop X.
- And we cannot go to state r and replace X by YZ.

A Graphical Notation for PDA's

- The list of δ facts is not too easy to follow.
- A diagram, generalizing the transition diagram of a finite automaton, will make aspects of the behavior of a given PDA clearer.
- We shall therefore introduce and subsequently use a *transition diagram* for PDA's.

A Graphical Notation for PDA's

We shall introduce and subsequently use a transition diagram for PDA's in which:

a) The nodes correspond to the states of the PDA.

b) An arrow labeled *Start* indicates the start state

c) And doubly circled states are accepting, as for finite automata.

d) The arcs correspond to transitions of the PDA in the following sense.
   - An arc labeled $a, X/\alpha$ from state q to state p means that $\delta(q, a, X)$ contains the pair (p, α), perhaps among other pairs.
   - That is, the arc label tells what input is used, and also gives the old and now tops of the stack.

- The only thing that the diagram does not tell us is which stack symbol is the start symbol.
- Conventionally, it is Z_0, unless we indicate otherwise.

A Graphical Notation for PDA's

We shall introduce and subsequently use a transition diagram for PDA's in which:

a) The nodes correspond to the states of the PDA.
b) An arrow labeled *Start* indicates the start state
c) And doubly circled states are accepting, as for finite automata.
d) The arcs correspond to transitions of the PDA in the following sense.
   - An arc labeled $a, X/\alpha$ from state q to state p means that $\delta(q, a, X)$ contains the pair (p, α), perhaps among other pairs.
   - That is, the arc label tells what input is used, and also gives the old and now tops of the stack.

- The only thing that the diagram does not tell us is which stack symbol is the start symbol.
- Conventionally, it is Z_0, unless we indicate otherwise.

A Graphical Notation for PDA's

We shall introduce and subsequently use a transition diagram for PDA's in which:

a) The nodes correspond to the states of the PDA.
b) An arrow labeled *Start* indicates the start state
c) And doubly circled states are accepting, as for finite automata.
d) The arcs correspond to transitions of the PDA in the following sense.
    - An arc labeled $a, X/\alpha$ from state q to state p means that $\delta(q, a, X)$ contains the pair (p, α), perhaps among other pairs.
    - That is, the arc label tells what input is used, and also gives the old and now tops of the stack.

- The only thing that the diagram does not tell us is which stack symbol is the start symbol.
- Conventionally, it is Z_0, unless we indicate otherwise.

A Graphical Notation for PDA's

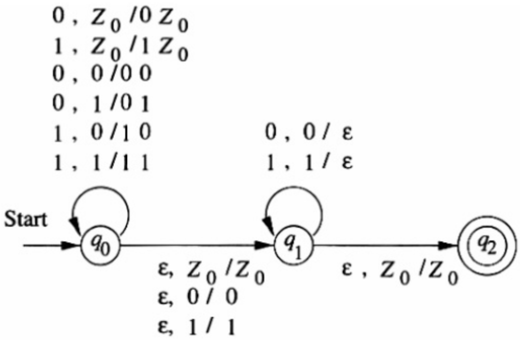
We shall introduce and subsequently use a transition diagram for PDA's in which:

a) The nodes correspond to the states of the PDA.
b) An arrow labeled *Start* indicates the start state
c) And doubly circled states are accepting, as for finite automata.
d) The arcs correspond to transitions of the PDA in the following sense.
    - An arc labeled $a, X/\alpha$ from state q to state p means that $\delta(q, a, X)$ contains the pair (p, α), perhaps among other pairs.
    - That is, the arc label tells what input is used, and also gives the old and now tops of the stack.

- The only thing that the diagram does not tell us is which stack symbol is the start symbol.
- Conventionally, it is Z_0, unless we indicate otherwise.

Example

The PDA of previous example is represented by the diagram shown.



Representing a PDA as a generalized transition diagram

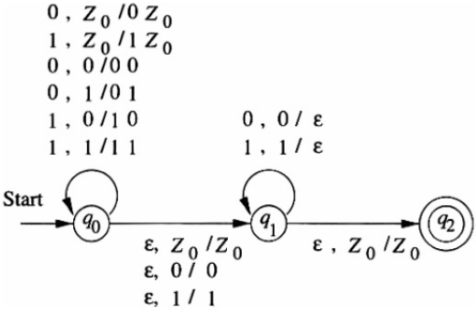
a) The nodes correspond to the states of the PDA.

b) An arrow labeled *Start* indicates the start state. and doubly circled states are accepting, as for finite automata.

c) The arcs correspond to transitions of the PDA. An arc labeled $a, X/\alpha$ from state q to state p means that $\delta(q, a, X)$ contains the pair (p, α), perhaps among other pairs.

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

δ is defined by the following rules:



a) The nodes correspond to the states of the PDA.

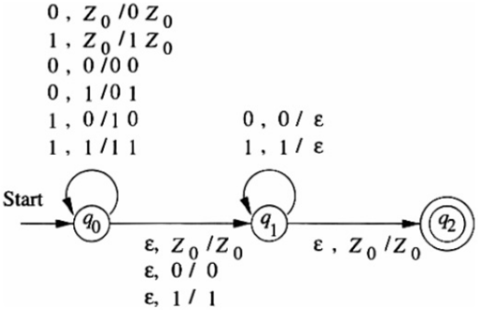
b) An arrow labeled *Start* indicates the start state. and doubly circled states are accepting, as for finite automata.

c) The arcs correspond to transitions of the PDA. An arc labeled $a, X/\alpha$ from state q to state p means that $\delta(q, a, X)$ contains the pair (p, α), perhaps among other pairs.

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

δ is defined by the following rules:

1. i) $\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$ and
   ii) $\delta(q_0, 1, Z_0) = \{(q_0, 1Z_0)\}$.

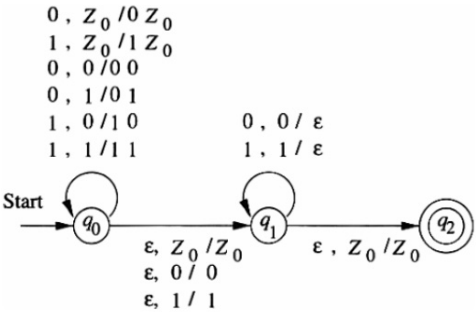


- a) The nodes correspond to the states of the PDA.
- b) An arrow labeled *Start* indicates the start state. and doubly circled states are accepting, as for finite automata.
- c) The arcs correspond to transitions of the PDA. An arc labeled $a, X/\alpha$ from state q to state p means that $\delta(q, a, X)$ contains the pair (p, α), perhaps among other pairs.

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

δ is defined by the following rules:

2. i) $\delta(q_0, 0, 0) = \{(q_0, 00)\}$,
   ii) $\delta(q_0, 0, 1) = \{(q_0, 01)\}$,
   iii) $\delta(q_0, 1, 0) = \{(q_0, 10)\}$, and
   iv) $\delta(q_0, 1, 1) = \{(q_0, 11)\}$.



a) The nodes correspond to the states of the PDA.

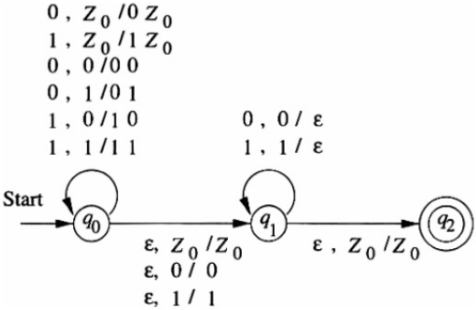
b) An arrow labeled *Start* indicates the start state. and doubly circled states are accepting, as for finite automata.

c) The arcs correspond to transitions of the PDA. An arc labeled $a, X/\alpha$ from state q to state p means that $\delta(q, a, X)$ contains the pair (p, α), perhaps among other pairs.

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

δ is defined by the following rules:

3. i) $\delta(q_0, \epsilon, Z_0) = \{(q_1, Z_0)\}$,
   ii) $\delta(q_0, \epsilon, 0) = \{(q_1, 0)\}$, and
   iii) $\delta(q_0, \epsilon, 1) = \{(q_1, 1)\}$.



a) The nodes correspond to the states of the PDA.

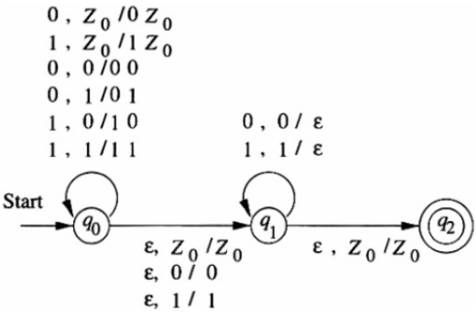
b) An arrow labeled *Start* indicates the start state. and doubly circled states are accepting, as for finite automata.

c) The arcs correspond to transitions of the PDA. An arc labeled $a, X/\alpha$ from state q to state p means that $\delta(q, a, X)$ contains the pair (p, α), perhaps among other pairs.

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

δ is defined by the following rules:

4.
   i) $\delta(q_1, 0, 0) = \{(q_1, \epsilon)\}$, and
   ii) $\delta(q_1, 1, 1) = \{(q_1, \epsilon)\}$.



a) The nodes correspond to the states of the PDA.

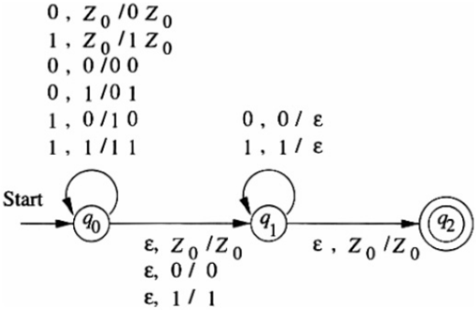
b) An arrow labeled *Start* indicates the start state. and doubly circled states are accepting, as for finite automata.

c) The arcs correspond to transitions of the PDA. An arc labeled $a, X/\alpha$ from state q to state p means that $\delta(q, a, X)$ contains the pair (p, α), perhaps among other pairs.

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

δ is defined by the following rules:

5. i) $\delta(q_1, \epsilon, Z_0) = \{(q_2, Z_0)\}$.



Instantaneous Descriptions of a PDA

- To this point, we have only an informal notion of how a PDA "computes."
- The PDA goes from configuration to configuration, in response to input symbols (or sometimes ϵ).
- In the finite automaton, the state is the only thing that we need to know about the automaton.
- The PDA's configuration involves both the state and the contents of the stack.
- Being arbitrarily large, the stack is often the more important part of the total configuration of the PDA at any time.
- It is also useful to represent as part of the configuration the portion of the input that remains.

Instantaneous Descriptions of a PDA

- To this point, we have only an informal notion of how a PDA "computes."
- The PDA goes from configuration to configuration, in response to input symbols (or sometimes ϵ).
- In the finite automaton, the state is the only thing that we need to know about the automaton.
- The PDA's configuration involves both the state and the contents of the stack.
- Being arbitrarily large, the stack is often the more important part of the total configuration of the PDA at any time.
- It is also useful to represent as part of the configuration the portion of the input that remains.

Instantaneous Descriptions of a PDA

- To this point, we have only an informal notion of how a PDA "computes."
- The PDA goes from configuration to configuration, in response to input symbols (or sometimes ϵ).
- In the finite automaton, the state is the only thing that we need to know about the automaton.
- The PDA's configuration involves both the state and the contents of the stack.
- Being arbitrarily large, the stack is often the more important part of the total configuration of the PDA at any time.
- It is also useful to represent as part of the configuration the portion of the input that remains.

Instantaneous Descriptions of a PDA

- To this point, we have only an informal notion of how a PDA “computes.”
- The PDA goes from configuration to configuration, in response to input symbols (or sometimes ϵ).
- In the finite automaton, the state is the only thing that we need to know about the automaton.
- The PDA’s configuration involves both the state and the contents of the stack.
- Being arbitrarily large, the stack is often the more important part of the total configuration of the PDA at any time.
- It is also useful to represent as part of the configuration the portion of the input that remains.

Instantaneous Descriptions of a PDA

- Thus, we shall represent the configuration of a PDA by a triple (q, w, γ), where
  1. q is the state,
  2. w is the remaining input, and
  3. γ is the stack contents.
- Conventionally, we show the top of the stack at the left end of γ and the bottom at the right end.
- Such a triple is called an *instantaneous description*, or *ID*, of the pushdown automaton.

Instantaneous Descriptions of a PDA

- Thus, we shall represent the configuration of a PDA by a triple (q, w, γ), where
  1. q is the state,
  2. w is the remaining input, and
  3. γ is the stack contents.
- Conventionally, we show the top of the stack at the left end of γ and the bottom at the right end.
- Such a triple is called an *instantaneous description*, or *ID*, of the pushdown automaton.

Instantaneous Descriptions of a PDA

- Thus, we shall represent the configuration of a PDA by a triple (q, w, γ), where
  1. q is the state,
  2. w is the remaining input, and
  3. γ is the stack contents.
- Conventionally, we show the top of the stack at the left end of γ and the bottom at the right end.
- Such a triple is called an *instantaneous description*, or *ID*, of the pushdown automaton.

Instantaneous Descriptions of a PDA

- For finite automata, the $\hat{\delta}$ notation was sufficient to represent sequences of instantaneous descriptions through which a finite automaton moved.
- The ID for a finite automaton is just its state.
- However, for PDA's we need a notation that describes changes in the state, the input, and stack.
- Thus, we adopt the "turnstile" notation for connecting pairs of ID's that represent one or many moves of a PDA.

Instantaneous Descriptions of a PDA

- Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$be a PDA.
- Define $\vdash_P$, or just $\vdash$ when P is understood, as follows.
- Suppose $\delta\ (q, a, X)$ contains $(p, \alpha\ )$.
- Then for all strings w in Σ^* and β in Γ^*:

$$(q, aw, X\beta) \vdash (p, w, \alpha\ \beta)$$

- This move reflects the idea that, by consuming a (which may be ϵ) from the input and replacing X on top of the stack by α , we can go from state q to state p.
- What remains on the input, w, and what is below the top of the stack, β, do not influence the action of the PDA.
- They are merely carried along, perhaps to influence events later.

Instantaneous Descriptions of a PDA

- Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$be a PDA.
- Define $\vdash_P$, or just $\vdash$ when P is understood, as follows.
- Suppose $\delta\ (q, a, X)$ contains $(p, \alpha\ )$.
- Then for all strings w in Σ^* and β in Γ^*:

$$(q, aw, X\beta) \vdash (p, w, \alpha\ \beta)$$

- This move reflects the idea that, by consuming a (which may be ϵ) from the input and replacing X on top of the stack by α , we can go from state q to state p.
- What remains on the input, w, and what is below the top of the stack, β, do not influence the action of the PDA.
- They are merely carried along, perhaps to influence events later.

Instantaneous Descriptions of a PDA

- Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$be a PDA.
- Define $\vdash_P$, or just $\vdash$ when P is understood, as follows.
- Suppose δ (q, a, X) contains $(p, \alpha\ )$.
- Then for all strings w in Σ^* and β in Γ^*:

$$(q, aw, X\beta) \vdash (p, w, \alpha\ \beta)$$

- This move reflects the idea that, by consuming a (which may be ϵ) from the input and replacing X on top of the stack by α , we can go from state q to state p.
- What remains on the input, w, and what is below the top of the stack, β, do not influence the action of the PDA.
- They are merely carried along, perhaps to influence events later.

Instantaneous Descriptions of a PDA

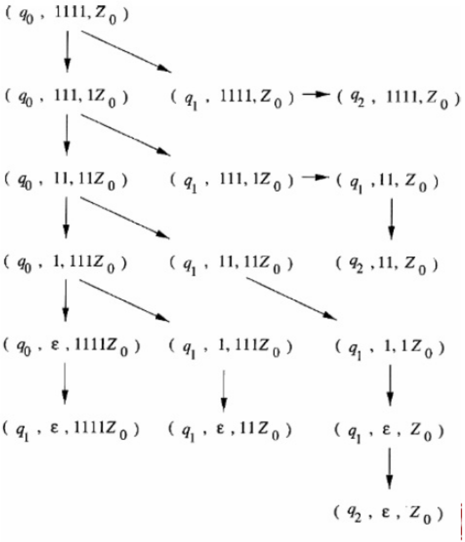
- Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$be a PDA.
- Define $\vdash_P$, or just $\vdash$ when P is understood, as follows.
- Suppose $\delta\ (q, a, X)$ contains $(p, \alpha\ )$.
- Then for all strings w in Σ^* and β in Γ^*:

$$(q, aw, X\beta) \vdash (p, w, \alpha\ \beta)$$

- This move reflects the idea that, by consuming a (which may be ϵ) from the input and replacing X on top of the stack by α , we can go from state q to state p.
- What remains on the input, w, and what is below the top of the stack, β, do not influence the action of the PDA.
- They are merely carried along, perhaps to influence events later.

Example

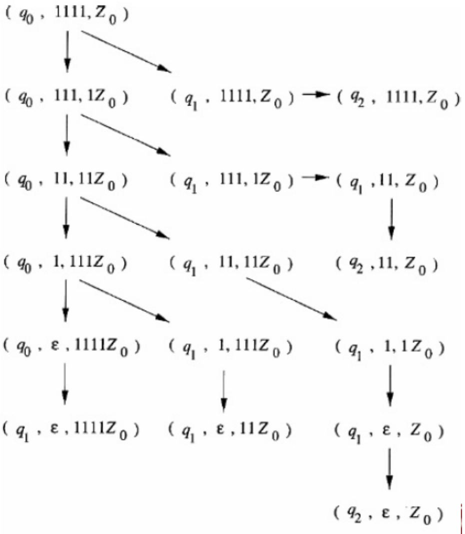
- Let us consider the action of the PDA of previous example on the input 1111.
- Since q_0 is the start state and $Z_0$is the start symbol, the initial ID is $(q_0, 1111, Z_0)$.
- On this input, the PDA has an opportunity to guess wrongly several times.
- The entire sequence of ID's that the PDA can reach from the initial ID $(q_0, 1111, Z_0)$ is shown.
- Arrows represent the $\vdash$ relation.



ID's of the PDA on input 1111

Example

- Let us consider the action of the PDA of previous example on the input 1111.
- Since q_0 is the start state and $Z_0$is the start symbol, the initial ID is $(q_0, 1111, Z_0)$.
- On this input, the PDA has an opportunity to guess wrongly several times.
- The entire sequence of ID's that the PDA can reach from the initial ID $(q_0, 1111, Z_0)$ is shown.
- Arrows represent the $\vdash$ relation.



ID's of the PDA on input 1111

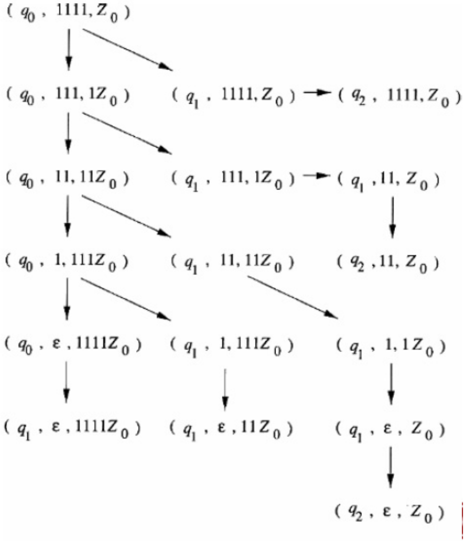
$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$

1111

δ is defined by the following rules:

1. i) $\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$ and
   ii) $\delta(q_0, 1, Z_0) = \{(q_0, 1Z_0)\}$

- From the initial ID, there are two choices of move.
- The first guesses that the middle has not been seen and leads to ID $(q_0, 111, 1Z_0)$.
- In effect, a 1 has been removed from the input and pushed onto the stack.



ID's of the PDA on input 1111

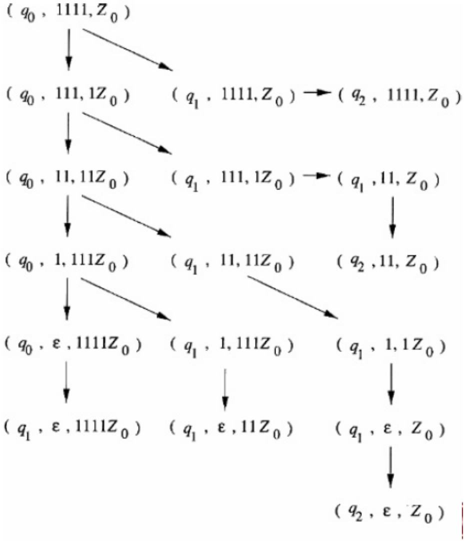
$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$

1111

δ is defined by the following rules:

3. i) $\delta(q_0, \epsilon, Z_0) = \{(q_1, Z_0)\}$,
   ii) $\delta(q_0, \epsilon, 0) = \{(q_1, 0)\}$, and
   iii) $\delta(q_0, \epsilon, 1) = \{(q_1, 1)\}$.

- The second choice from the initial ID guesses that the middle has been reached.
- Without consuming input, the PDA goes to state q_1, leading to the ID $(q_1, 1111, Z_0)$.



ID's of the PDA on input 1111

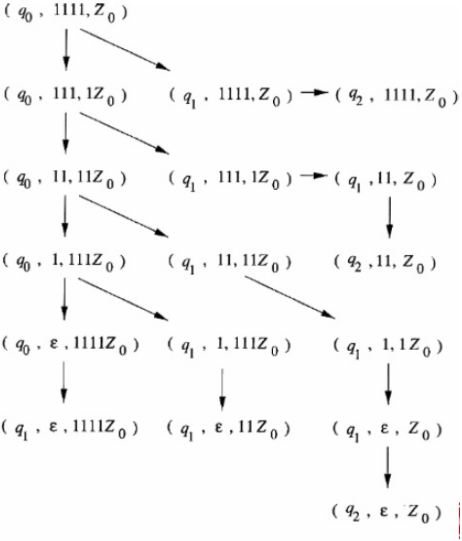
$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$

1111

δ is defined by the following rules:

5. 1) $\delta(q_1, \epsilon, Z_0) = \{(q_2, Z_0)\}$.

- Since the PDA may accept if it is in state q_1 and sees Z_0 on top of its stack, the PDA goes from there to ID $(q_2, 1111, Z_0)$.
- That ID is not exactly an accepting ID, since the input has not been completely consumed.



ID's of the PDA on input 1111

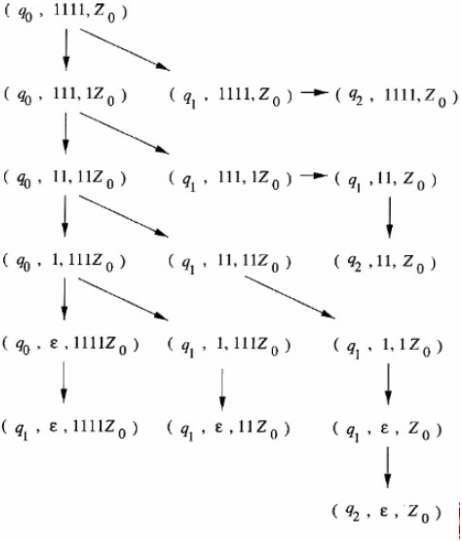
$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$

1111

δ is defined by the following rules:

5. ① $\delta(q_1, \epsilon, Z_0) = \{(q_2, Z_0)\}$.

- Had the input been ϵ rather than 1111, the same sequence of moves would have led to ID (q_2, ϵ, Z_0), which would show that ϵ is accepted.



ID's of the PDA on input 1111

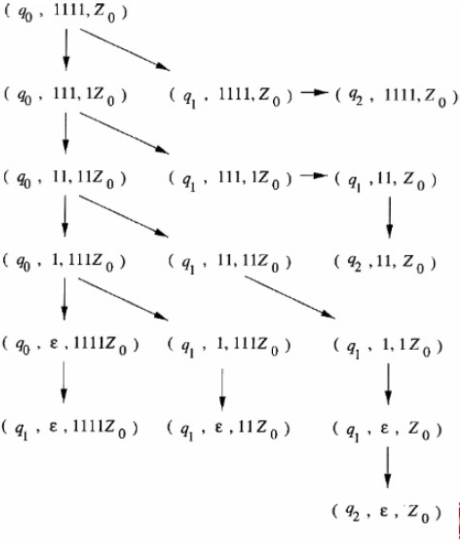
$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$

1111

δ is defined by the following rules:

3. i) $\delta(q_0, \epsilon, Z_0) = \{(q_1, Z_0)\}$,
   ii) $\delta(q_0, \epsilon, 0) = \{(q_1, 0)\}$, and
   iii) $\delta(q_0, \epsilon, 1) = \{(q_1, 1)\}$.

- The PDA may also guess that it has seen the middle after reading one 1, that is, when it is in the ID $(q_0, 111, 1Z_0)$.
- That guess also leads to failure, since the entire input cannot be consumed.



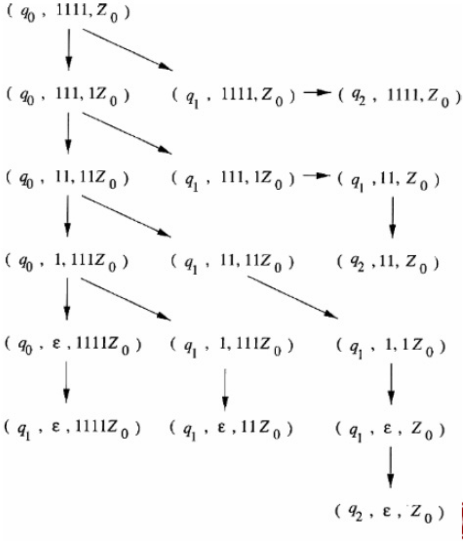
ID's of the PDA on input 1111

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

1111

δ is defined by the following rules:

- The correct guess, that the middle is reached after reading two 1's, gives us a sequence of ID's.



ID's of the PDA on input 1111

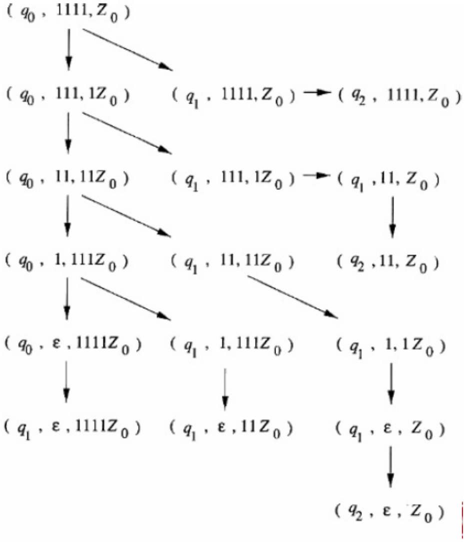
$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$

1111

δ is defined by the following rules:

1. i) $\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$ and
   ii) $\delta(q_0, 1, Z_0) = \{(q_0, 1Z_0)\}$

$$
\begin{aligned}
& (q_0, 1111, Z_0) \\
& \vdash (q_0, 111, 1Z_0) \\
& \vdash (q_0, 11, 11Z_0) \\
& \vdash (q_1, 11, 11Z_0) \\
& \vdash (q_1, 1, 1Z_0) \\
& \vdash (q_1, \epsilon\ , Z_0) \\
& \vdash (q_2, \epsilon\ , Z_0)
\end{aligned}
$$



ID's of the PDA on input 1111

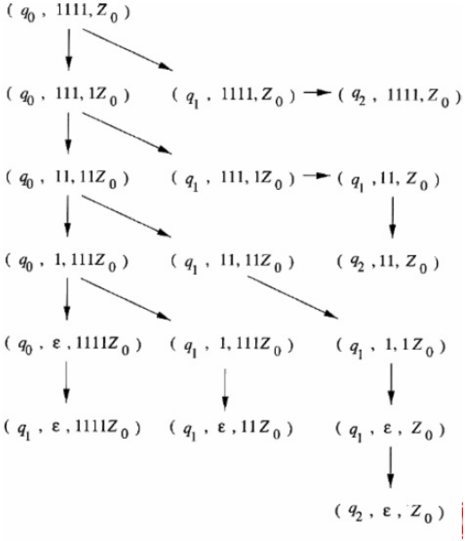
$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$

1111

δ is defined by the following rules:

2. i) $\delta(q_0, 0, 0) = \{(q_0, 00)\}$,
   ii) $\delta(q_0, 0, 1) = \{(q_0, 01)\}$,
   iii) $\delta(q_0, 1, 0) = \{(q_0, 10)\}$, and
   iv) $\delta(q_0, 1, 1) = \{(q_0, 11)\}$.

$(q_0, 1111, Z_0)$
$\vdash (q_0, 111, 1Z_0)$
$\vdash (q_0, 11, 11Z_0)$
$\vdash (q_1, 11, 11Z_0)$
$\vdash (q_1, 1, 1Z_0)$
$\vdash (q_1, \epsilon, Z_0)$
$\vdash (q_2, \epsilon, Z_0)$



ID's of the PDA on input 1111

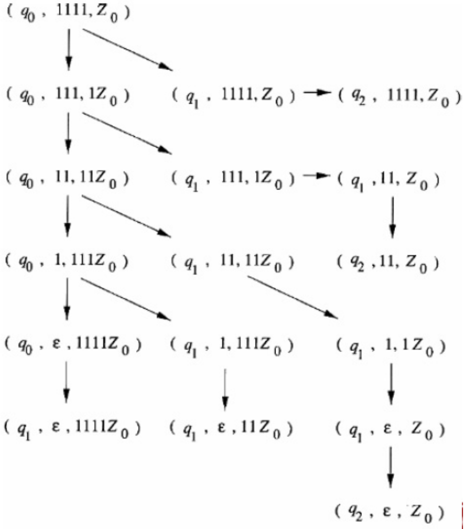
$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$

1111

δ is defined by the following rules:

3. i) $\delta(q_0, \epsilon, Z_0) = \{(q_1, Z_0)\}$,
   ii) $\delta(q_0, \epsilon, 0) = \{(q_1, 0)\}$, and
   iii) $\delta(q_0, \epsilon, 1) = \{(q_1, 1)\}$.

$$
\begin{aligned}
&(q_0, 1111, Z_0)\\
&\vdash (q_0, 111, 1Z_0)\\
&\vdash (q_0, 11, 11Z_0)\\
&\vdash (q_1, 11, 11Z_0)\\
&\vdash (q_1, 1, 1Z_0)\\
&\vdash (q_1, \epsilon, Z_0)\\
&\vdash (q_2, \epsilon, Z_0)
\end{aligned}
$$



ID's of the PDA on input 1111

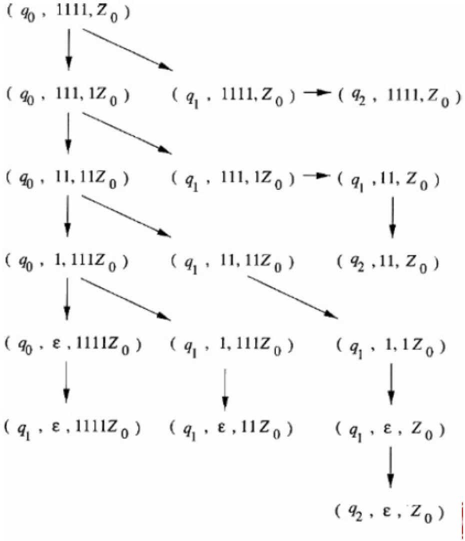
$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

1111

δ is defined by the following rules:

4.
   i) $\delta(q_1, 0, 0) = \{(q_1, \epsilon)\}$, and
   ii) $\delta(q_1, 1, 1) = \{(q_1, \epsilon)\}$.

$$
\begin{aligned}
&(q_0, 1111, Z_0)\\
&\vdash (q_0, 111, 1Z_0)\\
&\vdash (q_0, 11, 11Z_0)\\
&\vdash (q_1, 11, 11Z_0)\\
&\vdash (q_1, 1, 1Z_0)\\
&\vdash (q_1, \epsilon, Z_0)\\
&\vdash (q_2, \epsilon, Z_0)
\end{aligned}
$$



ID's of the PDA on input 1111

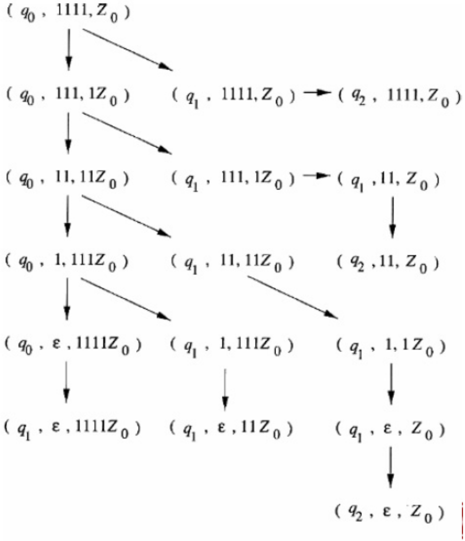
$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$

1111

δ is defined by the following rules:

5. $\delta(q_1, \epsilon, Z_0) = \{(q_2, Z_0)\}$.

$$
\begin{aligned}
&(q_0, 1111, Z_0) \\
&\vdash (q_0, 111, 1Z_0) \\
&\vdash (q_0, 11, 11Z_0) \\
&\vdash (q_1, 11, 11Z_0) \\
&\vdash (q_1, 1, 1Z_0) \\
&\vdash (q_1, \epsilon, Z_0) \\
&\vdash (q_2, \epsilon, Z_0)
\end{aligned}
$$



ID's of the PDA on input 1111

Instantaneous Descriptions of a PDA

There are three important principles about ID's and their transitions that we shall need in order to reason about PDA's:

1. If a sequence of ID's (*computation*) is legal for a PDA P, then the computation formed by adding the same additional input string to the end of the input (second component) in each ID is also legal.
2. If a computation is legal for a PDA P, then the computation formed by adding the same additional stack symbols below the stack in each ID is also legal.
3. If a computation is legal for a PDA P, and some tail of the input is not consumed, then we can remove this tail from the input in each ID, and the resulting computation will still be legal.

- Intuitively, data that P never looks at cannot affect its computation.

Instantaneous Descriptions of a PDA

There are three important principles about ID's and their transitions that we shall need in order to reason about PDA's:

1. If a sequence of ID's (*computation*) is legal for a PDA P, then the computation formed by adding the same additional input string to the end of the input (second component) in each ID is also legal.
2. If a computation is legal for a PDA P, then the computation formed by adding the same additional stack symbols below the stack in each ID is also legal.
3. If a computation is legal for a PDA P, and some tail of the input is not consumed, then we can remove this tail from the input in each ID, and the resulting computation will still be legal.

- Intuitively, data that P never looks at cannot affect its computation.

Instantaneous Descriptions of a PDA

There are three important principles about ID's and their transitions that we shall need in order to reason about PDA's:

1. If a sequence of ID's (*computation*) is legal for a PDA P, then the computation formed by adding the same additional input string to the end of the input (second component) in each ID is also legal.
2. If a computation is legal for a PDA P, then the computation formed by adding the same additional stack symbols below the stack in each ID is also legal.
3. If a computation is legal for a PDA P, and some tail of the input is not consumed, then we can remove this tail from the input in each ID, and the resulting computation will still be legal.

- Intuitively, data that P never looks at cannot affect its computation.

Instantaneous Descriptions of a PDA

There are three important principles about ID's and their transitions that we shall need in order to reason about PDA's:

1. If a sequence of ID's (*computation*) is legal for a PDA *P*, then the computation formed by adding the same additional input string to the end of the input (second component) in each ID is also legal.
2. If a computation is legal for a PDA *P*, then the computation formed by adding the same additional stack symbols below the stack in each ID is also legal.
3. If a computation is legal for a PDA *P*, and some tail of the input is not consumed, then we can remove this tail from the input in each ID, and the resulting computation will still be legal.

- Intuitively, data that *P* never looks at cannot affect its computation.

ID's for Finite Automata?

- One might wonder why we did not introduce for finite automata a notation like the ID's we use for PDA's.
- Although a FA has no stack, we could use a pair (q, w), where q is the state and w the remaining input, as the ID of a finite automaton.
- While we could have done so, we would not glean any more information from reachability among ID's than we obtain from the $\hat{\delta}$ notation.
- That is, for any finite automaton, we could show that $\hat{\delta}(q, w) = p$ if and only if $(q, wx) \vdash (p, x)$ for all strings x.

ID's for Finite Automata?

- One might wonder why we did not introduce for finite automata a notation like the ID's we use for PDA's.
- Although a FA has no stack, we could use a pair (q, w), where q is the state and w the remaining input, as the ID of a finite automaton.
- While we could have done so, we would not glean any more information from reachability among ID's than we obtain from the $\hat{\delta}$ notation.
- That is, for any finite automaton, we could show that $\hat{\delta}(q, w) = p$ if and only if $(q, wx) \vdash (p, x)$ for all strings x.

Example

- Design a PDA which accepts a string of balanced parenthesis.
- The string must always start with a left parenthesis.
- So the PDA accepts all of the following:

  $(()(())(()))$ $\quad$ $()()()$ $\quad$ ϵ

- But it does not accept any of the following:

  $())$ $\quad$ $)()($

Example-continued

- The principle involved here should be very simple.
- Whenever we see a left parenthesis, we push it onto the stack.
- Every left parenthesis can be paired with a unique subsequent right parenthesis.
- Whenever we see a right parenthesis, we try to match it with one in the stack.

Example-continued

- The states involved are,

  q_0: We are about to start scanning or we are in the middle of the string.

  q_1: We have finished scanning the whole string.

- While in q_1, if the stack is empty, the string is accepted.

Example-continued

- So, the PDA is as follows:

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

- The δ are as follows:
  - For the initial condition (or when the parentheses so far has been perfectly matched), $\delta(q_0, (, Z_0) = \{(q_0, (Z_0)\}$.
  - For storing the left parentheses, $\delta(q_0, (, () = \{(q_0, (()\}$.
  - For left-right matches, $\delta(q_0,), () = \{(q_0, \epsilon)\}$.
  - For acceptance, $\delta(q_0, \epsilon, Z_0) = \{(q_1, Z_0)\}$.

$$
\begin{aligned}
(q_0, (()(())(())), Z_0) &\vdash (q_0, ()(())(())), (Z_0)\\
&\vdash (q_0,)(())(())), ((Z_0)\\
&\vdash (q_0, (())(())), (Z_0)\\
&\vdash (q_0, ())(())), ((Z_0)\\
&\vdash (q_0,))(())), (((Z_0)\\
&\vdash (q_0,)(())), ((Z_0)\\
&\vdash (q_0, (())), (Z_0)\\
&\vdash (q_0, ())), ((Z_0)\\
&\vdash (q_0,))), (((Z_0)\\
&\vdash (q_0,)), ((Z_0)\\
&\vdash (q_0,), (Z_0)\\
&\vdash (q_0, \epsilon, Z_0)\\
&\vdash (q_1, \epsilon, Z_0)
\end{aligned}
$$

- $\delta(q_0, (, Z_0) = \{(q_0, (Z_0)\}$,
- $\delta(q_0, (, () = \{(q_0, (()\}$,
- $\delta(q_0,), () = \{(q_0, \epsilon)\}$,
- $\delta(q_0, \epsilon, Z_0) = \{(q_1, Z_0)\}$.

$$
\begin{aligned}
(q_0, ()()(), Z_0) &\vdash (q_0,)()(), (Z_0) \\
&\vdash (q_0, ()(), Z_0) \\
&\vdash (q_0,)(), (Z_0) \\
&\vdash (q_0, (), Z_0) \\
&\vdash (q_0,), (Z_0) \\
&\vdash (q_0, \epsilon, Z_0) \\
&\vdash (q_1, \epsilon, Z_0)
\end{aligned}
$$

- $\delta(q_0, (, Z_0) = \{(q_0, (Z_0)\}$,
- $\delta(q_0, (, () = \{(q_0, (()\}$,
- $\delta(q_0,), () = \{(q_0, \epsilon)\}$,
- $\delta(q_0, \epsilon, Z_0) = \{(q_1, Z_0)\}$.

$$(q_0, \epsilon, Z_0) \vdash (q_1, \epsilon, Z_0)$$

- $\delta(q_0, (, Z_0) = \{(q_0, (Z_0)\}$,
- $\delta(q_0, (, () = \{(q_0, (()\}$,
- $\delta(q_0,), () = \{(q_0, \epsilon)\}$,
- $\delta(q_0, \epsilon, Z_0) = \{(q_1, Z_0)\}$.

$$
\begin{aligned}
(q_0, ()), Z_0) &\vdash (q_0,)), (Z_0) \\
&\vdash (q_0,), Z_0) \\
&????
\end{aligned}
$$

- $\delta(q_0, (, Z_0) = \{(q_0, (Z_0)\}$,
- $\delta(q_0, (, () = \{(q_0, (()\}$,
- $\delta(q_0,), () = \{(q_0, \epsilon)\}$,
- $\delta(q_0, \epsilon, Z_0) = \{(q_1, Z_0)\}$.

$$(q_0,)()()(, Z_0) \vdash \quad ????$$

- $\delta(q_0, (, Z_0) = \{(q_0, (Z_0)\}$,
- $\delta(q_0, (, () = \{(q_0, (()\}$,
- $\delta(q_0,), () = \{(q_0, \epsilon)\}$,
- $\delta(q_0, \epsilon, Z_0) = \{(q_1, Z_0)\}$.

Example

- This pushdown automaton accepts the language $\{w \in \{a, b\}^* \ : \ w$ has the same number of a's and b's$\}$.
- Either a string of a's or a string of b's is kept by P on its stack.
- A stack of a's indicates the excess of a's over b's thus far read.
- If in fact P has read more a's than b's; a stack of b's indicates the excess of b's over a's.
- In either case, P keeps a special symbol Z_0 on the bottom of the stack as a marker.

Example-continued

- Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where
  - $Q = \{q, f\}$,
  - $\Sigma = \{a, b\}$,
  - $\Gamma = \{a, b, Z_0\}$,
  - $Z_0 = Z_0$,
  - $F = f$,
  - δ is listed below,
    1. $\delta(q, a, Z_0) = (q, aZ_0)$,
    2. $\delta(q, a, a) = (q, aa)$,
    3. $\delta(q, a, b) = (q, \epsilon)$,
    4. $\delta(q, b, Z_0) = (q, bZ_0)$,
    5. $\delta(q, b, b) = (q, bb)$,
    6. $\delta(q, b, a) = (q, \epsilon)$,
    7. $\delta(q, \epsilon, Z_0) = (f, Z_0)$.

Example-continued

1. $\delta(q, a, Z_0) = (q, aZ_0)$,
2. $\delta(q, a, a) = (q, aa)$,
3. $\delta(q, a, b) = (q, \epsilon)$,
4. $\delta(q, b, Z_0) = (q, bZ_0)$,
5. $\delta(q, b, b) = (q, bb)$,
6. $\delta(q, b, a) = (q, \epsilon)$,
7. $\delta(q, \epsilon, Z_0) = (f, Z_0)$.

- In state q, when P reads an a,
  - it either starts up a stack of a's from the bottom, while keeping the bottom marker (transition 1),
  - or pushes an a onto a stack of a's (transition 2),
  - or pops a b from a stack of b's (transition 3).

Example-continued

1. $\delta(q, a, Z_0) = (q, aZ_0)$,
2. $\delta(q, a, a) = (q, aa)$,
3. $\delta(q, a, b) = (q, \epsilon)$,
4. $\delta(q, b, Z_0) = (q, bZ_0)$,
5. $\delta(q, b, b) = (q, bb)$,
6. $\delta(q, b, a) = (q, \epsilon)$,
7. $\delta(q, \epsilon, Z_0) = (f, Z_0)$.

Example-continued

1. $\delta(q, a, Z_0) = (q, aZ_0)$,
2. $\delta(q, a, a) = (q, aa)$,
3. $\delta(q, a, b) = (q, \epsilon)$,
4. $\delta(q, b, Z_0) = (q, bZ_0)$,
5. $\delta(q, b, b) = (q, bb)$,
6. $\delta(q, b, a) = (q, \epsilon)$,
7. $\delta(q, \epsilon, Z_0) = (f, Z_0)$.

- When reading a *b* from the input, the machine acts analogously,
  - pushing a *b* onto a stack of *b*'s (transition 4),
  - or pushing a *b* onto a stack consisting of just a bottom marker (transition 5),
  - and popping an a from a stack of *a*'s (transition 6).

Example-continued

1. $\delta(q, a, Z_0) = (q, aZ_0)$,
2. $\delta(q, a, a) = (q, aa)$,
3. $\delta(q, a, b) = (q, \epsilon)$,
4. $\delta(q, b, Z_0) = (q, bZ_0)$,
5. $\delta(q, b, b) = (q, bb)$,
6. $\delta(q, b, a) = (q, \epsilon)$,
7. $\delta(q, \epsilon, Z_0) = (f, Z_0)$.

- Finally, when Z_0 is the topmost (and therefore the only) symbol on the stack, the machine may pass to a final state (transition 7).
- If at this point all the input has been read, then the configuration (f, ϵ, Z_0) has been reached, and the input string is accepted.

Language of a PDA

- The language accepted by P, denoted $L(P)$, is the set of all strings accepted by P.
- Two approaches
  1. acceptance by final state
  2. acceptance by empty stack
- Two methods are equivalent.
- A language L has a PDA that accepts it by final state if and only if L has a PDA that accepts it by empty stack.

Language of a PDA

- The language accepted by P, denoted $L(P)$, is the set of all strings accepted by P.
- Two approaches
  1. acceptance by final state
  2. acceptance by empty stack
- Two methods are equivalent.
- A language L has a PDA that accepts it by final state if and only if L has a PDA that accepts it by empty stack.

Equivalence of PDA's and CFG's

- Following three classes of languages are equivalent
  - The context free languages, i.e. the languages defined by CFG's.
  - The languages that are accepted by final state by some PDA.
  - The languages that are accepted by empty stack by some PDA.

Acceptance by Final State and Empty Stack

- Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA
- $L(P)$, the language accepted by P by final state, is

$$\{w | (q_0, w, Z_0) \vdash_P^* (q, \epsilon, \alpha)\}$$

for some state $q \in F$ and any stack string α.

- $N(P)$, the language accepted by P by empty stack, is

$$\{w | (q_0, w, Z_0) \vdash_P^* (q, \epsilon, \epsilon)\}$$

for any state q. Since the set of accepting states is irrelevant we can omit F.

From Empty Stack to Final State

- If $L = N(P_N)$ for some PDA $P_N = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, then there is a PDA P_F such that $L = L(P_F)$.
- We use a new symbol $X_0 \notin \Gamma$
- X_0 is the start symbol and a marker on the bottom of the stack.
- We also need a new start state p_0 whose sole function is to push Z_0 and enter state q_0.
- We need another state, p_f, which is accepting state of P_F
- Whenever P_F sees X_0 on the top of the stack, it transfers to p_f

From Empty Stack to Final State

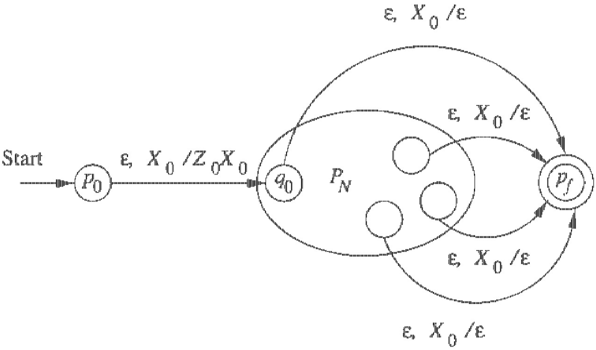


Figure 6.4: P_F simulates P_N and accepts if P_N empties its stack

From Final State to Empty Stack

- If $L = L(P_F)$ for some PDA $P_F = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, then there is a PDA P_N such that $L = N(P_N)$.
- From each accepting state of P_F, add a transition on ϵ to a new state p.
- When in state p, P_N pops its stack and does not consume any input.
- To avoid a situation where P_F accidentally empties its stack, P_N must use a marker X_0 on the bottom of the stack.
- We also need a new start state p_0 whose sole function is to push start symbol of P_F and enter start state of P_F.

From Final State to Empty Stack

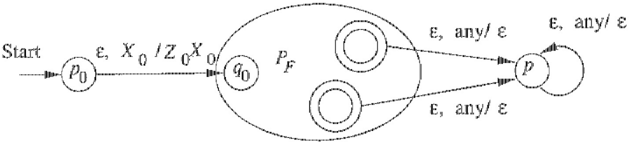


Figure 6.7: P_N simulates P_F and empties its stack when and only when P_N enters an accepting state

From Grammar to PDA (Empty Stack)

- Given a CFG G, we construct a PDA that simulates the leftmost derivations of G.
- Consider $S \stackrel{*}{\Longrightarrow} xA\alpha \stackrel{*}{\Longrightarrow} w$.
  - A is the leftmost variable
  - x is whatever terminals appear to its left
  - α is the string of terminals and variables that appear to its right
- The PDA has matched x with inputs
- $A\alpha$ is in the stack
- Suppose the PDA is an ID $(q, y, A\alpha)$, it guesses the production to use $A \to \beta$ and moves to $(q, y, \beta\alpha)$

From Grammar to PDA (Empty Stack)

- Let $G = (V, T, P, S)$ be a CFG. Construct the PDA that accepts $L(G)$ by empty stack as follows.

$$P = (\{q\}, T, V \cup T, \delta, q, S)$$

where δ is defined by

1. For each variable A,

$$\delta(q, \epsilon, A) = \{(q, \beta) | A \to \beta \text{ is a production of } G\}$$

2. For each terminal a, $\delta(q, a, a) = \{(q, \epsilon)\}$

Example

- Consider the grammar

$$\begin{array}{lcl} I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ E & \rightarrow & I \mid E * E \mid E + E \mid (E) \end{array}$$

- The transition function is

a) $\delta(q, \epsilon, I) = \{(q, a),\ (q, b),\ (q, Ia),\ (q, Ib),\ (q, I0),\ (q, I1)\}$.

b) $\delta(q, \epsilon, E) = \{(q, I),\ (q, E + E),\ (q, E * E),\ (q, (E))\}$.

c) $\delta(q, a, a) = \{(q, \epsilon)\}$; $\delta(q, b, b) = \{(q, \epsilon)\}$; $\delta(q, 0, 0) = \{(q, \epsilon)\}$; $\delta(q, 1, 1) = \{(q, \epsilon)\}$; $\delta(q, (, () = \{(q, \epsilon)\}$; $\delta(q,),)) = \{(q, \epsilon)\}$; $\delta(q, +, +) = \{(q, \epsilon)\}$; $\delta(q, *, *) = \{(q, \epsilon)\}$.

$$a + a * b$$

$$\begin{aligned}
E &\Longrightarrow E + E \\
&\Longrightarrow I + E \\
&\Longrightarrow a + E \\
&\Longrightarrow a + E * E \\
&\Longrightarrow a + I * E \\
&\Longrightarrow a + a * E \\
&\Longrightarrow a + a * I \\
&\Longrightarrow a + a * b
\end{aligned}$$

$$\begin{aligned}
(q, a + a * b, E) &\vdash (q, a + a * b, E + E) \\
&\vdash (q, a + a * b, I + E) \\
&\vdash (q, a + a * b, a + E) \\
&\vdash (q, + a * b, +E) \\
&\vdash (q, a * b, E) \\
&\vdash (q, a * b, E * E) \\
&\vdash (q, a * b, I * E) \\
&\vdash (q, a * b, a * E) \\
&\vdash (q, * b, *E) \\
&\vdash (q, b, E) \\
&\vdash (q, b, I) \\
&\vdash (q, b, b) \\
&\vdash (q, \epsilon, \epsilon)
\end{aligned}$$

From Grammar to PDA (Empty Stack)

Theorem

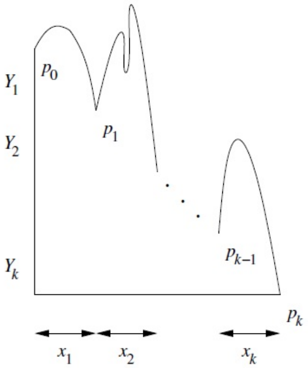
If PDA P is constructed from CFG G by the construction above, then $N(P) = L(G)$

Proof

- By induction
- Skipped

From PDA (Empty Stack) to Grammar

- We are interested in the net popping of one symbol off the stack, while consuming some input
- A PDA may change state as it pops stack symbols, so we also note the state that it enters when it finally pops a level off its stack



From PDA (Empty Stack) to Grammar

Our equivalent grammar uses variables each of which represents an event consisting of

- The net popping of some symbol X from the stack
- A change in state from some p at the beginning to q when X has finally been replaced by ϵ on the stack
- We represent such a variable by the composite symbol $[pXq]$

From PDA (Empty Stack) to Grammar

We shall construct $G = (V, \Sigma, R, S)$ where the set of variables V consists of

- The special symbol S, which is the start symbol
- All symbols of the form $[pXq]$ where p and q are states in Q, and X is a stack symbol

The productions of G are as follows

- For *all* states p, G has the production $S \rightarrow [q_0 Z_0 p]$
- Let $\delta(q, a, X)$ contain the pair $(r, Y_1 Y_2 \dots Y_k)$, where
  1. a is either a symbol in Σ or $a = \epsilon$
  2. k can be any number, including 0 in which case the pair is (r, ϵ)

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

$$[qXr_k] \rightarrow a[rY_1 r_1][r_1 Y_2 r_2] \dots [r_{k-1} Y_k r_k]$$

Example

- $\delta(q_0, (, Z_0) = \{(q_0, (Z_0)\}$,
- $\delta(q_0, (, () = \{(q_0, (()\}$,
- $\delta(q_0,), () = \{(q_0, \epsilon)\}$,
- $\delta(q_0, \epsilon, Z_0) = \{(q_1, \epsilon)\}$.

The last transition modified to empty the stack

- $S \rightarrow [q_0 Z_0 q_0]$
- $S \rightarrow [q_0 Z_0 q_1]$
- $[q_0 Z_0 q_0] \rightarrow ([q_0 (q_0][q_0 Z_0 q_0]$
- $[q_0 Z_0 q_0] \rightarrow ([q_0 (q_1][q_1 Z_0 q_0]$
- $[q_0 Z_0 q_1] \rightarrow ([q_0 (q_0][q_0 Z_0 q_1]$
- $[q_0 Z_0 q_1] \rightarrow ([q_0 (q_1][q_1 Z_0 q_1]$

...

From PDA (Empty Stack) to Grammar

Theorem

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

Proof - skipped

Pumping Lemma for CFL

- Both CFG and PDA describe context free language
- Some simple languages are not context free
  - $L = \{0^n1^n2^n | n \geq 1\}$ - cannot match three groups
  - $L = \{0^i1^j2^i3^j | i \geq 1 \text{ and } j \geq 1\}$ - cannot match two pairs if they interleave
  - $L = \{ww | w \text{ is in } \{0,1\}^*\}$ - cannot match two strings of arbitrary length if alphabet size greater than one
- Pumping lemma for context-free languages is used to show certain languages are not context free
- It says that in any sufficiently long string in a CFL, it is possible to find at most two short, nearby substrings, that we can pump in tandem

Pumping Lemma for CFL

Theorem (The pumping lemma for context-free languages)

Let L be a CFL. Then there exists a constant n such that if z is any string in L such that $|z|$ is at least n, then we can write $z = uvwxy$, subject to the following conditions

1. $|vwx| \leq n$. *That is, the middle portion is not too long*
2. $vx \neq \epsilon$. *Since v and x are the pieces to be pumped,this condition says that at least one of the strings we pump must not be empty*
3. *For all $i \geq 0$, uv^iwx^iy is in L. That is, the two strings v and x may be pumped any number of times including 0 and the resulting string will still be a member of L*

Example

Consider $L = \{0^n 1^n 2^n | n \geq 1\}$

- $|vwx| \leq n$ and v and x are not both ϵ
- vwx cannot include both 0's and 2's
- If vwx has no 2's then uwy, which should be in L, has fewer 0's or 1's than 2's
- If vwx has no 0's then uwy has n 0's but fewer 1's or 2's

We get a contradiction, so L cannot be context-free.

Size of Parse Trees

Theorem

Suppose we have a parse tree according to a Chomsky Normal Form grammar $G = (V, T, P, S)$ and suppose that the yield of the tree is a terminal string w. If the length of the longest path is n then $|w| \leq 2^{n-1}$.

Proof - skipped

Pumping Lemma for CFL Proof

- If G has m variables, choose $n = 2^m$
- Suppose that z in L is of length at least n
- Any parse tree whose longest path is of length $\leq m$ must have a yield of length $\leq 2^{m-1} = n/2$
- It cannot have yield z because z is too long. Any parse tree with yield z has a path of length at least $m + 1$
- Longest path in the tree for z is of length $k + 1$ where $k \geq m$

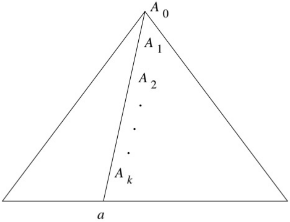
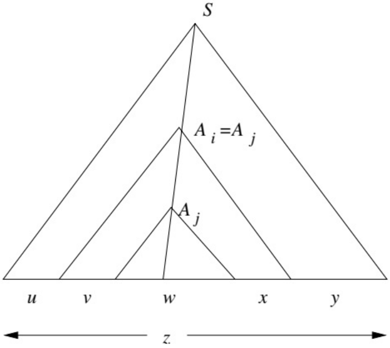


Figure 7.5: Every sufficiently long string in L must have a long path in its parse tree

Proof

- Since $k \geq m$, there are at least $m + 1$ occurrences of variables $A_0, A_1, \ldots A_k$ on the path
- As there are only m different variables in V at least two of the last $m + 1$ variables on the path must be same
- Suppose $A_i = A_j$. Then it is possible to divide the tree as shown



Proof

- $A_i = A_j = A$
- Replace the subtree rooted at A_i which has yield vwx by the subtree rooted at A_j which has yield w (Fig b)
- Corresponds to the case $i = 0$ in the pattern of strings uv^iwx^iy
- Replace the subtree rooted at A_j by the subtree rooted at A_i (Fig c)
- The yield of this tree is uv^2wx^2y

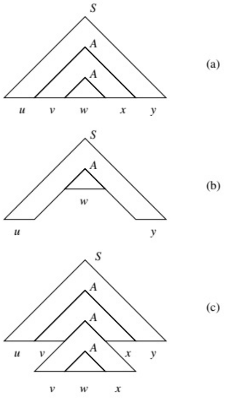


Figure 7.7: Pumping strings v and x zero times and pumping them twice

Deterministic PDA

- A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is said to be deterministic iff the following conditions are met:
  - $\delta\,(q, a, X)$ has at most one member for any $q \in Q$, $a \in \Sigma$ or $a = \epsilon$ and $X \in \Gamma$.
  - IF $\delta\,(q, a, X)$ is nonempty for some $a \in \Sigma$, the $\delta\,(q, \epsilon\,, X)$ must be empty.
- That is at most one choice in every step

Deterministic PDA

- The language accepted by DPDA's by empty stack is limited. It has the prefix property
  - no two different strings x and y in L such that one is a prefix on another
- The language accepted by DPDA's by final state properly include the regular languages, but are properly included in the CFL's. They are called ***deterministic context-free language (DCFL)***
- The language DPDA's accept all have unambiguous grammars.
  - But DPDA languages are not exactly the subset of CFL's that are not inherently ambiguous

Example

- L_{ww^R} is a CFL that has no DPDA
- By putting a center marker c in the middle, we can make the language recognizable by a DPDA. That is, we can recognize the language

$$L_{wcw^R} = \{wcw^R | w \text{ is in } (0+1)^*\}$$

Example

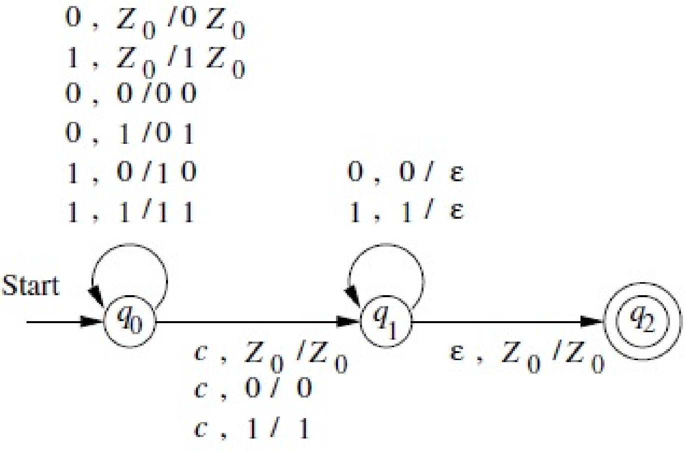


Figure 6.11: A deterministic PDA accepting L_{wcwr}

LR(k) grammar

- Widely used in parsers for programming languages/compilers
- Generates exactly the DPDA languages
- LR(k) stands for
  - (L)eft to right input processing
  - (R)ightmost derivations
  - k symbols lookahead
- Informally means we can decide which production to apply in a ***reduction*** (reversed or bottom-up derivation) of a string by looking ahead k symbols

Context-sensitive grammar

- In context-free grammars, the productions are of the form

$$A \to \gamma$$

where A is a variable and γ is a string over variables and terminals

- In ***context-sensitive grammars***, they are of the form

$$\alpha A \beta \to \alpha \gamma \beta$$

where α and β are strings over variables and terminals
  - α and β can be empty
- There are grammars called ***unrestricted grammars*** that can generate exactly all languages that can be recognized by a Turing machine
  - Recursively enumerable or Turing-recognizable languages

Chomsky hierarchy