FORMAL LANGUAGES AND AUTOMATA THEORY

Module 4

Rijin IK

Assistant Professor
Department of Computer Science and Engineering
Vimal Jyothi Engineering College
Chemperi

November 1, 2023

Outline

Course Outcomes

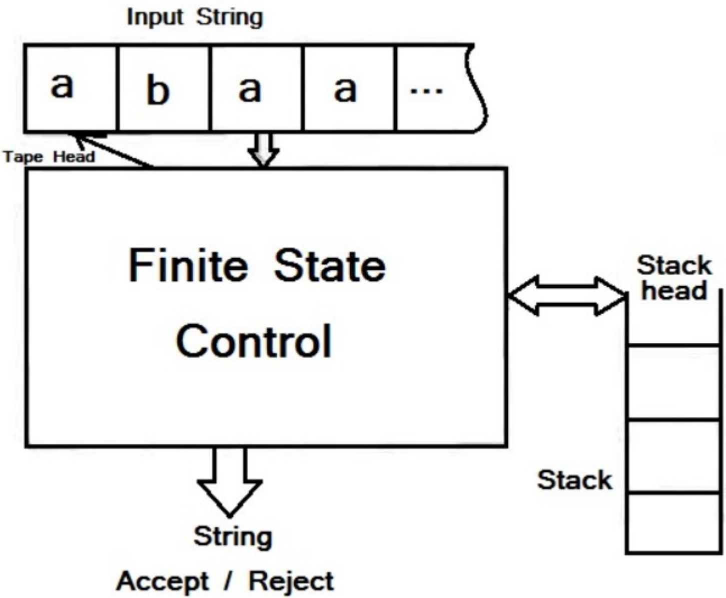
After the completion of the course the student will be able to

1. Classify a given formal language into Regular, Context-Free, Context Sensitive, Recursive or Recursively Enumerable. [Cognitive knowledge level: Understand]
2. Explain a formal representation of a given regular language as a finite state automaton, regular grammar, regular expression and Myhill-Nerode relation. [Cognitive knowledge level: Understand]
3. Design a Pushdown Automaton and a Context-Free Grammar for a given context-free language. [Cognitive knowledge level : Apply]
4. Design Turing machines as language acceptors or transducers. [Cognitive knowledge level: Apply]
5. Explain the notion of decidability. [Cognitive knowledge level: Understand]

Nondeterministic Pushdown Automata

- A non-deterministic pushdown automaton (NPDA), or just pushdown automaton (PDA) is a variation on the idea of a non-deterministic finite automaton (NDFA).
- Unlike an NDFA, a PDA is associated with a stack (hence the name pushdown).
- The transition function must also take into account the "state" of the stack.

Nondeterministic Pushdown Automata



Nondeterministic Pushdown Automata

A pushdown automaton has three components

1. An input tape,
2. A control unit, and
3. A stack with infinite size.

Transition

- Read the current input symbol and the current symbol on the top of the stack
- Based on the **current state** , **current input** symbol and **current stack symbol** , change the state and replace the symbol on the top of of the stack with a string symbols
- After reading of a symbol from the input tape the input tape head (read head) moves one position to the right in the input tape

Acceptance criteria

1. **Empty stack acceptance**
   - Input is consumed and stack is empty
2. **Final state acceptance**
   - Input is consumed and the PDA is in a final state

Formal definition of PDA

- A Pushdown Automaton (PDA) is a seven-tuple:
  $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$
  - Q A finite set of states
  - Σ A finite input alphabet
  - Γ A finite stack alphabet
  - q_0 The initial/starting state, q_0 is in Q
  - z_0 Initial stack symbol, is in Γ
  - F A set of final/accepting states, which is a subset of Q
  - δ A transition function, where

$$\delta : Q \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow \textit{ finite subsets of } Q \times \Gamma^*$$

Nondeterministic Pushdown Automata

- The transition function δ 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.
- $\delta(q, a, x) = (p, \gamma)$ where,
  - q is the state in Q
  - a is an input symbol in Σ
  - x is the stack symbol in Γ
  - p is the new state
  - γ is a string of stack symbols
- Stack operation
  - If $\gamma = \epsilon$, then the stack is popped
  - If $\gamma =$ x , then the stack is unchanged
  - If $\gamma =$ yz , then the x is replaced by z and y is pushed on to the stack

Nondeterministic Pushdown Automata

Example: Design a PDA that accepts $\{ww^R | w \ in \ (0+1)^*\}$
Sln:

- $L = \{\epsilon, 0, 1, 00, 11, 0110, 1001, ...\}$
- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be the PDA
- Consider $M = (\{q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_1, Z_0, \phi)$

$$\begin{aligned}
\delta(q_1, 0, Z_0) &= \{(q_1, 0Z_0)\} \\
\delta(q_1, 1, Z_0) &= \{(q_1, 1Z_0)\} \\
\delta(q_1, 0, 0) &= \{(q_1, 00), (q2, \epsilon)\} \\
\delta(q_1, 1, 0) &= \{(q_1, 10)\} \\
\delta(q_1, 0, 1) &= \{(q_1, 01)\} \\
\delta(q_1, 1, 1) &= \{(q_1, 11), (q2, \epsilon)\} \\
\delta(q_2, 0, 0) &= \{(q_2, \epsilon)\} \\
\delta(q_2, 1, 1) &= \{(q_2, \epsilon)\} \\
\delta(q_1, \epsilon, Z_0) &= \{(q_2, \epsilon\} \\
\delta(q_2, \epsilon, Z_0) &= \{(q_2, \epsilon)\}
\end{aligned}$$

Transition diagram

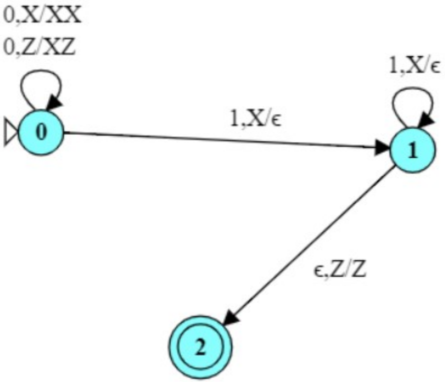


Figure: Transition diagram

Transition diagram

- We can show PDAs this in a transition diagram similar to the ones we used for FAs.
- Instead of labeling each transition with just one symbol (the input), we have to label it with three components $a, X | \alpha$ where
  - a is the next input symbol.
  - X is the symbol the symbol at the top of the stack, where X is one symbol of Γ
  - α is a string of stack symbols that are to be pushed onto the stack (after popping X)
    - Take note that this is a string: it is not limited to a single character. It can be empty.

Transition diagram

- In the rule $\epsilon, Z \Big| Z$, we match the Z on the stack, popping it, but the $\Big| Z$ means that we replace it by another Z. The next effect is that we simply left the Z where it was on the stack.
- In the rule $0, Z \Big| XZ$, if we see a 0 in the input and if see a Z on the top of the stack, we pop the matched Z from the top of the stack, and then replace it by XZ. The characters are pushed in reverse order, so first the Z is pushed back onto the bottom of the stack, and then the X is pushed on top of it.
- In the rule $1, X \Big| \epsilon$, if we see a 1 in the input and if we see an 'X' on the top of the stack, we pop the matched X and replace it by ϵ , the empty string. In other words, we don't replace the popped symbol by anything at all.

Nondeterministic Pushdown Automata

Instantaneous Descriptions of PDA

- An Instantaneous Description (also known as Configuration) of a PDA represents the state of the automaton at a particular moment during its operation.
- The Instantaneous Description of a PDA consists of the following components: (q, w, α),
  - q is the current state.
  - w is the remaining input.
  - γ is the stack contents, top of the stack is at the left end of γ.
- A move from one instantaneous description to another is denoted by the symbol $\vdash$(Turnstile notation)
- Suppose $\delta(q, a, x)$ contain (P, α) then all string w in Σ^* and β in Γ^*

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

where $a \in \Sigma \cup \{\epsilon\}$

$I_1 \vdash^* I_n$ represents a sequence of moves. $I_1 \vdash I_2 \vdash I_3 \ldots \vdash I_n$

Nondeterministic Pushdown Automata

Example: Show the IDs or moves for input string $w =$ "$aaabbb$" of PDA $L = a^n b^n$ then

$$
\begin{aligned}
(q_0, aaabbb, Z_0) &\vdash (q_0, aabbb, aZ_0) \\
&\vdash (q_0, abbb, aaZ_0) \\
&\vdash (q_0, bbb, aaaZ_0) \\
&\vdash (q_1, bb, aaZ_0) \\
&\vdash (q_1, b, aZ_0) \\
&\vdash (q_1, \epsilon, Z_0) \\
&\vdash (q_2, \epsilon, Z_0)
\end{aligned}
$$

This can be written as $(q_0, aaabbb, Z_0) \vdash (q_2, \epsilon, Z_0)$

The language of PDA

The language of a PDA is the set of all strings accepted by the PDA by performing some sequence of moves.

1. Acceptance by final state
   - PDA performs some sequence of moves on inputs and entering into the final states.

$$\{w \big| (q_0, w, Z_0) \vdash^* (p, \epsilon, \gamma) \textit{for some } p \textit{ in } F \textit{ and } \gamma \textit{ in } \Gamma\}$$

2. Acceptance by empty stack
   - PDA performs some sequence of moves on inputs and causes the PDA to empty its stack.

$$\{w \big| (q_0, w, Z_0) \vdash^* (p, \epsilon, \epsilon) \textit{for some } p \textit{ in } Q\}$$

Equivalence of Acceptance by Final State and Empty Stack

- Let's prove that the two definitions of acceptance by a PDA - acceptance by final state and empty stack - are equivalent.
- We will show that for any PDA M that accepts a language L by final state, we can construct an equivalent PDA M_1 that accepts the same language L by empty stack, and vice versa.

Equivalence of Acceptance by Final State and Empty Stack

CASE 1: PDA M accepts by empty stack. (acceptance by empty stack$M$ to acceptance by final stateM_1):

Theorem

If $L = N(M)$ for some PDA $M = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0, \phi)$ then there is a PDA M_1 such that $L = L(M_1)$

Proof:

- We will construct a new PDA M_1 from M in such a way that M_1 simulates M and enters its final state when and only when M empties its stack.
- To do this, we introduce two new states,p and p_f and a new stack symbol x_0 not in the original stack alphabet of M.

Equivalence of Acceptance by Final State and Empty Stack

Proof cont..

- The new PDA M_1 contains all the transitions of M, and in addition, it includes two more transitions:
  - $\delta_1(p, \epsilon, x_0) = (q_0, z_0x_0)$ This transition allows M_1 to enter the initial configuration of M with the bottom-of-stack marker x_0 below the symbols of M's stack.
  - $\delta_1(q, \epsilon, x_0) = (p_f, x_0)$ This transition causes M_1 to enter its final state "p_f" when M empties its stack, leaving only the bottom marker x_0.
- $M_1 = (Q \cup \{p, p_f\}, \Sigma, \Gamma \cup \{x_0\}, \delta_1, p, x_0, p_f)$
- We will prove that M and M_1 are equivalent.
- That is we will show that M accepts a string w if and only if M_1 accepts the same string w.

Equivalence of Acceptance by Final State and Empty Stack

Proof cont..

- If M accepts a string w, then there exists a sequence of transitions that leads M to empty its stack after processing w. M_1 simulates the same transitions and eventually reaches the final state p_f with only the x_0 marker in the stack, thereby accepting w
- If M_1 accepts a string w, then there exists a sequence of transitions that leads M_1 to its final state p_f with only the x_0 marker in the stack after processing w. Since M_1 simulates all transitions of M, the same sequence of transitions leads M to empty its stack, accepting w
- Therefore, $L(M) = N(M_1)$, and PDA M_1 accepts the same language L as PDA M, proving the equivalence in the case where M accepts by empty stack.

Proof cont..

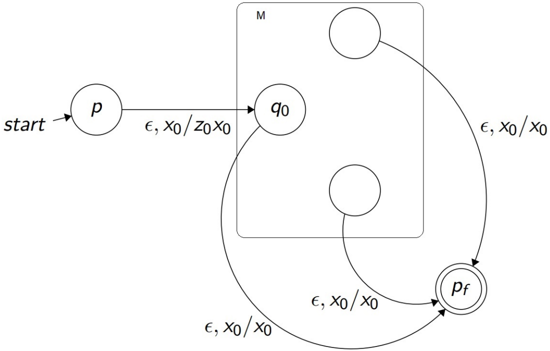


Figure: Transition diagram of M'

Equivalence of Acceptance by Final State and Empty Stack

CASE 2: PDA M accepts by final state (acceptance by final state (M) to acceptance by empty stack. $(M_1\ ))$

Theorem

If $L = L(M)$ for some PDA $M = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0, f)$ then there is a PDA M_1 such that $L = N(M_1)$

Proof:

- Assume that PDA M accepts a language L by reaching a final state after processing an input string.
- To show equivalence, we will construct another PDA, M_1, that accepts the same language L using an empty stack.
- To do this, we introduce two new states,p_0 and p and a new stack symbol x_0 not in the original stack alphabet of M.

Equivalence of Acceptance by Final State and Empty Stack

Proof cont..

- The new PDA M_1 contains all the transitions of M, and in addition, it includes some more transitions:
  - $\delta_1(p_0, \epsilon, x_0) = (q_0, z_0x_0)$ This transition allows M_1 to enter the initial configuration of M with the bottom-of-stack marker x_0 below the symbols of $M's$ stack.
  - For every accepting state $q \in F$ in PDA M, add an ϵ-transition from q to p in PDA M_1.

    $$\delta_1(q, \epsilon, \gamma) \; contain \; (p, \epsilon)$$

    where γ any stack symbol
  - For all stack symbol $\gamma \in \Gamma \cup \{x_0\}$

    $$\delta_1(p, \epsilon, \gamma) \; contain \; (p, \epsilon)$$

Equivalence of Acceptance by Final State and Empty Stack

Proof cont..

- Pushing z_0 onto the stack, p_0 enters the state q_0, which is the initial state of PDA M. Then, after consuming its input w, PDA M enters one of its final states.
- To construct PDA $M1$, for each accepting state q in PDA M, we add a transition to the new state p on epsilon (ϵ) with any stack symbol and delete that stack symbol. This process allows PDA M_1 to simulate the behavior of PDA M and recognize when M empties its stack.
- As a result, whenever PDA M enters a final state after consuming the input w, PDA M_1 will empty its stack after processing the same input w.
- Hence, PDA M_1 accepts the same language L as PDA M, proving the equivalence of acceptance by final state and empty stack.

Equivalence of Acceptance by Final State and Empty Stack

Proof cont..

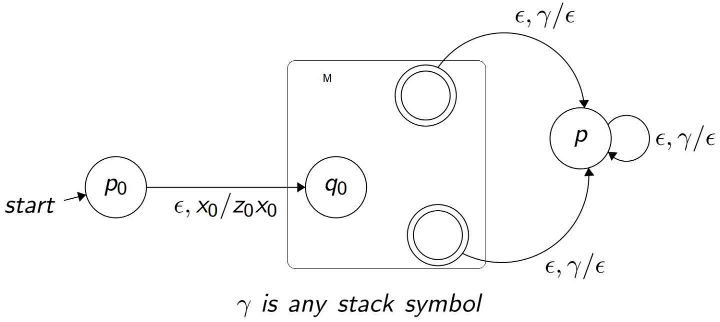


Figure: Transition diagram of M'

Deterministic PDA (DPDA)

- A PDA is said to be deterministic if all derivation(ID) in the design has to give only single move.
  - That is the PDA is deterministic in the sense that at most one move is possible the given state, input symbol and stack symbol

Formally we say a PDA $M = (Q, \Sigma, \delta, \Gamma, q_0, z_0, F)$ is deterministic if

1. $\delta(q, a, X)$ has at most one member for any $q \in Q, a \in \Sigma \cup \epsilon$ and $X \in \Gamma$
2. If $\delta(q, a, X)$ is nonempty, for some $a \in \Sigma$ then $\delta(q, \epsilon, X)$ must be empty

*The DPDA's accept a class of languages that is between the regular languages and the CFL's.

Deterministic PDA (DPDA)

Example:

- Consider the language $L = \{0^n 1^n | n \geq 1\}$. It turns out that this language can be recognized by a deterministic PDA.
    1. The PDA starts in the initial state and begins to read the input string.
    2. While reading consecutive 0's, it pushes them onto the stack, effectively storing the count of 0's encountered.
    3. Upon encountering a 1, the PDA transitions to another state, where it starts popping the 0's from the stack each time it reads a 1.
    4. If the PDA tries to pop more 0's than the number of 0's it has encountered while reading the input, it reaches a state where the stack is empty before consuming all the input. In this case, the PDA halts and rejects the input since it cannot be in the form $0^m 1^m$, where m is the number of 0's encountered.
    5. If the PDA successfully pops all the 0's from the stack, and the entire input has been read, it reaches the initial symbol at the bottom of the stack.
    6. When the PDA reaches the initial symbol at the bottom of the stack after reading the entire input, it accepts the input since the number of 0's and 1's is equal and in the correct order.

Example:

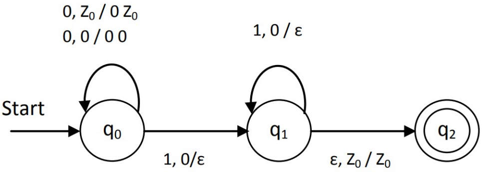


Figure: Deterministic PDA accepting $\{0^n1^n | n > 1\}$

Deterministic PDA (DPDA)

S. No	DPDA(Deterministic Pushdown Automata)	NPDA(Non-deterministic Pushdown Automata)
1.	It is less powerful than NPDA.	It is more powerful than DPDA.
2.	It is possible to convert every DPDA to a corresponding NPDA.	It is not possible to convert every NPDA to a corresponding DPDA.
3.	The language accepted by DPDA is a subset of the language accepted by NDPA.	The language accepted by NPDA is not a subset of the language accepted by DPDA.
4.	The language accepted by DPDA is called DCFL(Deterministic Context-free Language) which is a subset of NCFL(Non-deterministic Context-free Language) accepted by NPDA.	The language accepted by NPDA is called NCFL(Non-deterministic Context-free Language).

Figure: Difference Between NPDA and DPDA

Equivalence of PDA's and CFL's

The goal is to prove that the following three classes of the languages are all the same class.

1. The context-free languages (The language defined by CFG's).
2. The languages that are accepted by empty stack by some PDA.
3. The languages that are accepted by final state by some PDA.

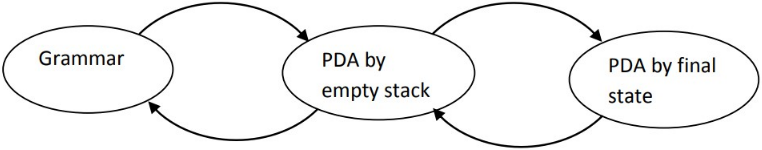


Figure: Organization of constructions showing equivalence of three ways of defining the CFL's

We have already shown that (2) and (3) are the same. Now, we prove that (1) and (2) are same.

Conversion of CFG to PDA

- If L is a context-free language, then there exists a PDA M such that $L = N(M)$.

Procedure

- Let $L = L(G)$, where $G = (V, T, P, S)$ is a context free grammar.
- We construct a PDA $M = M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ such that $L(M) = L(M)$ by empty stack
- The machine constructed has only one state q
- All terminals are the input Symbol
- The set of nonterminals and the set of terminals as its stack symbol

Conversion of CFG to PDA

Procedure cont..

- $M = M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$
- where

$$\begin{aligned} Q &= \{q\} \\ \Sigma &= T \\ \Gamma &= V \cup T \\ q_0 &= q \\ z_0 &= S \\ F &= \phi \end{aligned}$$

Where δ is defined by the following rules

1. If $A \rightarrow \beta$ is in P $\beta \in (V \cup T)^*$ Then
   - $\delta(q, \epsilon, A)$ contain (q, β)
2. For each $a \in T$
   - $\delta(q, a, a)$ contains (q, ϵ)

Example: Construct a pda M equivalent to the following context free grammar:

$$
\begin{aligned}
S &\rightarrow 0BB \\
B &\rightarrow 0S|1S|0.
\end{aligned}
$$

Test whether 010^4 is in $N(M)$.

Solution: Define pda A as follows:

$$A = (\{q\}, \{0, 1\}, \delta, \{S, B, 0, 1\}, q, S, \phi)$$

δ is defined by the following rules:

$$
\begin{aligned}
R1 &: \quad \delta(q, \epsilon, S) = \{(q, 0BB)\} \\
R2 &: \quad \delta(q, \epsilon, B) = \{(q, 0S), (q, 1S), (q, 0)\} \\
R3 &: \quad \delta(q, 0, 0) = \{(q, \epsilon)\} \\
R4 &: \quad \delta(q, 1, 1) = \{(q, \epsilon)\}
\end{aligned}
$$

String Checking

$$
\begin{aligned}
(q, 010^4, S) &\vdash (q, 010^4, 0BB) \quad byRuleR1 \\
&\vdash (q, 10^4, BB) \quad byRuleR3 \\
&\vdash (q, 10^4, 1SB) \quad byRuleR2 \\
&\vdash (q, 0^4, SB) \quad byRuleR4 \\
&\vdash (q, 0^4, 0BBB) \quad byRuleR1 \\
&\vdash (q, 0^3, BBB) \quad byRuleR3 \\
&\vdash^* (q, 0^3, 000) \quad byRuleR2 \\
&\vdash^* (q, \epsilon, \epsilon) \quad byRuleR3
\end{aligned}
$$

Conversion of PDA to CGF

- Will do this in two steps
  1. Every NPDA can be simulated by an NPDA with one state
  2. Every NPDA with one state has an equivalent CFG.

Conversion of PDA to CGF

Converting many-state PDA to one-state PDA

- Let's say P is a many state PDA with transition relation δ.
- We'll create a single state PDA P1 with transition relation δ_1.
- The stack symbols of P1 look like $[pXq]$ where p and q are states of P and X is a stack symbol from P
- P1 has only one state which we'll call 1 for distinction.
- For each transition in δ, create one or more transitions in δ_1.

Conversion of PDA to CGF

1 δ pops a symbol (and moves from p to q)

$$\delta(p, a, X) = \{(q, \epsilon)\} \implies \delta_1(1, a, [pXq]) = \{(1, \epsilon)\}$$

2 δ replaces a symbol

$$\delta(p, a, X) = \{(q, Y)\} \implies \forall_b \delta_1(1, a, [pXb]) = \{(1, [qYb])\}$$

3 δ pushes 2 or more symbols

$$\delta(p, a, X) = \{(q, YZ)\} \implies \forall_g, b\delta_1(1, a, [pXb]) = \{(1, [qYg][gZb])\}$$

*If δ pushes more symbols, e.g. $WXYZ$, then we must make a guess for each symbol, and the RHS will look like: $[qWg_1][g_1Xg_2][g_2Yg_3][g_3Zb]$

Conversion of PDA to CGF

Example

- Let P be a PDA that accepts $\{0^n 1 0^n | n \geq 1\}$ and has three states:
  - State p reads 0's from the input and pushes them onto the stack. It may stay in state p or move to state q.
  - State q reads a single 1 from the input and moves to state r.
  - State r reads 0's from the input and pops 0's from the stack.
- δ looks like:

$$\begin{aligned}
\delta(p, 0, Z0) &= \{(p, 0Z0), (q, 0Z0)\} \\
\delta(p, 0, 0) &= \{(p, 00), (q, 00)\} \\
\delta(q, 1, 0) &= \{(r, 0)\} \\
\delta(r, 0, 0) &= \{(r, \epsilon)\} \\
\delta(r, 0, Z0) &= \{(r, \epsilon)\}
\end{aligned}$$

Conversion of PDA to CGF

Example cont..

Here are the stack symbols of P1:

* [p0p], [p0q], [p0r]
* [p1p], [p1q], [p1r]
* [pZ_0p], [pZ_0q], [pZ_0r]
* [q0p], [q0q], [q0r]
* [q1p], [q1q], [q1r]
* [qZ_0p], [qZ_0q], [qZ_0r]
* [r0p], [r0q], [r0r]
* [r1p], [r1q], [r1r]
* [rZ_0p], [rZ_0q], [rZ_0r]

Conversion of PDA to CGF

Here are three examples of transitions in δ and the transitions in δ_1 that they generate.

1. $\delta(p, 0, 0) \rightarrow (q, 00)$

$$\begin{aligned}
\delta_1(1, 0, [p0p]) &= \{(1, [q0p][p0p]), (1, [q0q][q0p]), (1, [q0r][r0p])\} \\
\delta_1(1, 0, [p0q]) &= \{(1, [q0p][p0q]), (1, [q0q][q0q]), (1, [q0r][r0q])\} \\
\delta_1(1, 0, [p0r]) &= \{(1, [q0p][p0r]), (1, [q0q][q0r]), (1, [q0r][r0r])\}
\end{aligned}$$

2. $\delta(q, 1, 0) \rightarrow (r, 0)$

$$\begin{aligned}
\delta_1(1, 1, [q0p]) &= \{(1, [r0p])\} \\
\delta_1(1, 1, [q0q]) &= \{(1, [r0q])\} \\
\delta_1(1, 1, [q0r]) &= \{(1, [r0r])\}
\end{aligned}$$

- $\delta(r, 0, 0) \rightarrow (r, \epsilon)$

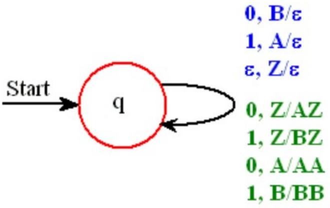
$$\delta_1(1, 0, [r0r]) = (1, \epsilon)$$

Single state PDA to grammar G

- For a single State PDA $N = (Q, \Sigma, \Gamma, \delta, q, Z_0)$, we construct a grammar $G = (V, T, P, S)$, such that
  - $L(G) = L(N)$, where $V = \Gamma, T = \Sigma$ *and* $S = Z_0$. the following rules outline the set of productions, P :

$$\begin{aligned} For\ every\ (q, a, Z) \quad &= \quad (q, \gamma),\ we\ add\ Z \rightarrow a\gamma,\ where \\ & \quad\ a \in \Sigma \cup \{\epsilon\}, Z \in \Gamma, \gamma \in \Gamma^* \end{aligned}$$

Example: Consider the PDA $P_N = (\{q\}, \{0, 1\}, \{Z, A, B\}, \delta_N, q, Z)$ in Figure The corresponding context-free grammar $G = (V, \{0, 1\}, P, S)$ is given by:



Solution:

- $V = \{S, Z, A, B\}$.

Conversion of PDA to CGF

Example cont..

P:

$$S \rightarrow Z$$

$$Z \rightarrow 0AZ \text{ (} since\ \delta_N(q, 0, Z)\ contains\ (q, AZ))$$

$$Z \rightarrow 1BZ \text{ (} since\ \delta_N(q, 1, Z)\ contains\ (q, BZ))$$

$$A \rightarrow 0AA \text{ (} since\ \delta_N(q, 0, A)\ contains\ (q, AA))$$

$$B \rightarrow 1BB \text{ (} since\ \delta_N(q, 1, B)\ contains\ (q, BB))$$

Example cont..

$$A \rightarrow 1 \text{ (} \mathit{since}\ \delta_N(q,1,A)\ \mathit{contains}\ (q,\epsilon))$$

$$B \rightarrow 0 \text{ (} \mathit{since}\ \delta_N(q,0,B)\ \mathit{contains}\ (q,\epsilon))$$

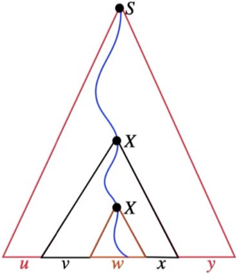
$$Z \rightarrow \epsilon \text{ (} \mathit{since}\ \delta_N(q,\epsilon,Z)\ \mathit{contains}\ (q,\epsilon))$$

Pumping Lemma for context-free languages

There is a pumping lemma for CFLs similar to the one for regular sets. It can be used in the same way to show that certain sets are not context-free.

Pumping Lemma for context-free languages

- Let L be any CFL. Then there is a constant n, depending only on L, such that if z is in L and $|z| \geq n$, then we may write $z = uvwxy$ such that
  1. $|vx| \geq 1, i.e\ vx \neq \epsilon$
  2. $|vwx| \leq n,\ and$
  3. for all $i \geq 0, uv^i wx^i y$ is in L.



Example: Consider the language $L = \{a^p | p \ is \ prime\}$. Suppose L were context free and let n be the constant. $n \in N$

- $L = \{a, a^3, a^5, a^7, ...\}$
  - Consider $z = a^5$.
    - $|z| = 5 \ \ 5 \geq n$
  - Write $z = uvwxy$ so as to satisfy the conditions of the pumping lemma.

    1. $|vx| \geq 1, i.e \ vx \neq \epsilon$
    2. $|vwx| \leq n, \ and$
    3. for all $i \geq 0, uv^i wx^i y$ is in L.

  Assume $u = a, v = a, w = a, x = a, y = a$
- put i=3: then $uv^i wx^i y$ become
  - $a(a)^3 a(a)^3 a = aaaaaaaaa$
  - $|aaaaaaaaa| = 9$, 9 is not a prime number, which is not belong to L, so this contradicts our assumption
  - So L is not context free.

Closure Properties of Context Free Languages

Closure Properties of Context Free Languages

- A CFL is basically a set of strings
- **Closure properties** of any set S is the operations which take element/elements of S as input and produce an element of S as output
- Thus, closure properties of CFLs are those set operations which take CFLs/CFL as input and produces a CFL as output

Closure Properties

1. Context-free languages are closed under union, concatenation and Kleene closure.
2. The context-free languages are closed under substitution.
3. The CFL's are closed under homomorphism.
4. The CFL's are closed under inverse homomorphism.
5. The CFL's are not closed under intersection.
6. The CFL's are not closed under complementation.
7. If L is a CFL and R is a regular set, then $L \cap R$ is a CFL

Closure Properties of Context Free Languages

Theorem: CFLs are closed under union

- If L_1 and L_2 are CFLs, then $L_1 \cup L_2$ is a CFL.

Proof

- Let L1 and L2 be generated by the CFG, $G_1 = (V_1, T_1, P_1, S_1)$ and $G_2 = (V_2, T_2, P_2, S_2)$, respectively.
- Without loss of generality, subscript each nonterminal of G_1 with a 1, and each nonterminal of G_2 with a 2 (so that $V_1 \cap V_2 = \phi$).
- Define the CFG, G, that generates $L_1 \cup L_2$ as follows: $G = (V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \rightarrow S_1 | S_2\}, S)$.
- A derivation starts with either $S \implies S_1$ *or* $S \implies S_2$.
- Subsequent steps use productions entirely from G_1 or entirely from G_2.
- Each word generated thus is either a word in L_1 or a word in L_2.

Example:

- Let $L1$ be Palindrome, defined by
  - $S_1 \rightarrow aS_1a|bS_1b|a|b|\epsilon$
- Let L2 be $\{a^nb^n|n \geq 0\}$ defined by:
  - $S_2 \rightarrow aS_2b|\epsilon$
- Then the union language is defined by:

$$\begin{array}{rcl} S & \rightarrow & S_1|S_2 \\ S_1 & \rightarrow & aS_1a|bS_1b|a|b|\epsilon \\ S_2 & \rightarrow & aS_2b|\epsilon \end{array}$$

Theorem: CFLs are closed under concatenation

- If L_1 *and* L_2 are CFLs, then L_1L_2 is a CFL.

Proof

- Let L_1 and L_2 be generated by the CFG, $G_1 = (V_1, T_1, P_1, S_1)$ and $G_2 = (V_2, T_2, P_2, S_2)$, respectively.
- Without loss of generality, subscript each nonterminal of G_1 with a 1, and each nonterminal of G_2 with a 2 (so that $V_1 \cap V_2 = \phi$).
- Define the CFG, G, that generates L_1L_2 as follows: $G = (V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \to S_1S_2\}, S)$.
- Each word generated thus is a word in L_1 followed by a word in L_2.

Example

- Let L1 be Palindrome, defined by:
  - $S_1 \rightarrow aS_1a|bS_1b|a|b|\epsilon$
- Let L2 be $\{a^nb^n|n \geq 0\}$ defined by:
  - $S_2 \rightarrow aS_2b|\epsilon$
- Then the concatenation language is defined by:

$$\begin{aligned} S &\rightarrow S_1S_2 \\ S_1 &\rightarrow aS_1a|bS_1b|a|b|\epsilon \\ S_2 &\rightarrow aS_2b|\epsilon \end{aligned}$$

Closure Properties of Context Free Languages

Theorem: CFLs are closed under Kleene star

- If L_1 is a CFL, then $L_1^*$is a CFL.

Proof

- Let L_1 be generated by the CFG, $G_1 = (V_1, T_1, P_1, S_1)$.
- Without loss of generality, subscript each nonterminal of G_1 with a 1.
- Define the CFG, G, that generates L_1^* as follows:
  - $G = (V_1 \cup \{S\}, T_1, P_1 \cup \{S \rightarrow S_1 S | \epsilon\}, S)$.
- Each word generated is either ϵ or some sequence of words in L_1.
- Every word in L_1^* (i.e., some sequence of 0 or more words in L_1) can be generated by G.

Example

- Let L_1 be $\{a^n b^n | n \geq 0\}$ defined by:
- $S \rightarrow aSb|\epsilon$
- Then L_1^* is generated by:

$$\begin{aligned} S &\rightarrow S_1 S|\epsilon \\ S_1 &\rightarrow aS_1 b|\epsilon \end{aligned}$$

None of these example grammars is necessarily the most compact CFG for the language it generates.

Closure Properties of Context Free Languages

Theorem: CFLs are not closed under intersection

- If L_1 and L_2 are CFLs, then $L_1 \cap L_2$ may not be a CFL

Proof

- $L_1 = \{a^n b^n a^m | n, m \geq 0\}$ is generated by the following CFG:

$$\begin{aligned} S &\rightarrow XA \\ X &\rightarrow aXb|\epsilon \\ A &\rightarrow Aa|\epsilon \end{aligned}$$

- $L_2 = \{a^n b^m a^m | n, m \geq 0\}$ is generated by the following CFG:

$$\begin{aligned} S &\rightarrow AX \\ X &\rightarrow aXb|\epsilon \\ A &\rightarrow Aa|\epsilon \end{aligned}$$

- $L_1 \cap L_2 = \{a^n b^n a^n | n \geq 0\}$, which is known not to be a CFL (pumping lemma).

Closure Properties of Context Free Languages

Theorem: CFLs are not closed under complement

- If L_1 is a CFL, then $\bar{L_1}$ may not be a CFL.

Proof

- Assume the complement of every CFL is a CFL
- Let L_1 and L_2 be 2 CFLs.
- Since CFLs are close under union, and we are assuming they are closed under complement, $\overline{\bar{L_1} \cup \bar{L_2}} = L1 \cap L2$ is a CFL.
- However, we know there are CFLs whose intersection is not a CFL.
- Therefore, our assumption that CFLs are closed under complement is false.

Example This does not mean that the complement of a CFL is never a CFL

- Let $L_1 = \{a^n b^n a^n | n \geq 0\}$, which is not a CFL.
- $\bar{L_1}$ is a CFL.
- We show this by constructing it as the union of 5 CFLs.
  - $M_{pq} = (a^+)(a^n b^n)(a^+) = \{a^p b^q a^r | p > q\}$
  - $M_{qp} = (a^n b^n)(b^+)(a^+) = \{a^p b^q a^r | p < q\}$
  - $M_{qr} = (a^+)(b^+)(b^n a^n) = \{a^p b^q a^r | q > r\}$
  - $M_{qr} = (a^+)(b^n a^n)(a^+) = \{a^p b^q a^r | q < r\}$
  - $M = a^+ b^+ a^+ =$ all words not of the form $a^p b^q a^r$
- Let $L = M \cup M_{pq} \cup M_{qp} \cup M_{qr} \cup M_{qr}$.
- Since $M \subseteq L$, Lcontains only words of the form $a^p b^q a^r$

Example cont..

- $\bar{L}$ cannot contain words of the form $a^p b^q a^r$, where $p < q$.
- $\bar{L}$ cannot contain words of the form $a^p b^q a^r$, where $p > q$.
- Therefore $\bar{L}$ only contains words of the form $a^p b^q a^r$, where $p = q$.
- $\bar{L}$ cannot contain words of the form $a^p b^q a^r$, where $q < r$.
- $\bar{L}$ cannot contain words of the form $a^p b^q a^r$, where $q > r$.
- Therefore $\bar{L}$ only contains words of the form $a^p b^q a^r$, where $q = r$.
- Since $p = q$ and $q = r$, $\bar{L}$ contains words of the form $a^n b^n a^n$, which is not context-free.

Theorem: The intersection of a CFL and an RL is a CFL.

- IfL_1 is a CFL and L_2 is regular, then $L_1 \cap L_2$ is a CFL

Proof

- We do this by constructing a PDA I to accept the intersection that is based on a PDA A for L_1 and a FA F for L_2.
- Convert A, if necessary, so that all input is read before accepting.
- Construct a set Y of all $A's$ states $y_1, y_2, ...,$ and a set X of all $F's$ states $x_1, x_2, ...$
- Construct $\{(y, x)|\forall_y \in Y, \forall_x \in X\}$.
- The start state of I is (y_0, x_0), where y_0 is the label of $A's$ start state, and x_0 is $F's$ initial state.
- Regarding the next state function, the x component changes only when the PDA is in a READ state:
  - If in (y_i, x_j) and y_i is not a READ state, its successor is (y_k, x_j),where y_k is the appropriate successor of y_i .
  - If in (y_i, x_j) and $y_i$is a READ state, reading a, its successor is(y_k, x_l), where y_k is the appropriate successor of y_i on an a, $\delta(x_j, a) = x_l$

- $I's$ ACCEPT states are those where the y component is ACCEPT and the x component is final.
- If the y component is ACCEPT and the x component is not final, the state in I is REJECT (or omitted, implying a crash).

Theorem: Context free languages are closed under Homomorphism

Example:

- Suppose L is a CFL over alphabet Σ, and h is a homomorphism on Σ.
- Let s be the substitution that replaces each symbol a in Σ by the language consisting of the one string that is $h(a)$
- i.e $s(a) = \{h(a)\}$ for all a in Σ. then $h(L) = s(L)$