Theory of Computation

(Context-Free Grammars)

Pramod Ganapathi
Department of Computer Science
State University of New York at Stony Brook

January 24, 2021

Contents

Contents

Context-Free Grammars (CFG)

Computer program compilation

C++ program:

```
#include <iostream>
using namespace std;
int main()
{
    if (true)
    {
        cout << "Hi 1";
        else
            cout << "Hi 2";
    }
    return 0;
}
```

C++ program:

```
#include <iostream>
using namespace std;

int main()
{
    if (true)
        cout << "Hi 1";
        else
            cout << "Hi 2";

    return 0;
}
```

Computer program compilation

C++ program:

```
#include <iostream>
using namespace std;
int main()
{
    if (true)
    {
        cout << "Hi 1";
        else
            cout << "Hi 2";
    }
    return 0;
}
```

Output:

```
error:  expected ‘}’ before ‘else’
```

C++ program:

```
#include <iostream>
using namespace std;

int main()
{
    if (true)
        cout << "Hi 1";
        else
            cout << "Hi 2";

    return 0;
}
```

Output:

```
Hi 1
```

Computer program compilation

C++ program:

```
#include <iostream>
using namespace std;
int main()
{
    if (true)
    {
        cout << "Hi 1";
        else
            cout << "Hi 2";
    }
    return 0;
}
```

Output:

`error: expected ‘}’ before ‘else’`

C++ program:

```
#include <iostream>
using namespace std;

int main()
{
    if (true)
        cout << "Hi 1";
        else
            cout << "Hi 2";

    return 0;
}
```

Output:

`Hi 1`

- DFA cannot check the syntax of a computer program.
- We need context-free grammars – a computational model more powerful than finite automata to check the syntax of most structures in a computer program.

Construct CFG for $L = \{a^n b^n \mid n \geq 0\}$

Problem

- Construct a CFG that accepts all strings from the language $L = \{a^n b^n \mid n \geq 0\}$

Construct CFG for $L = \{a^n b^n \mid n \geq 0\}$

Problem

- Construct a CFG that accepts all strings from the language $L = \{a^n b^n \mid n \geq 0\}$

Solution

- Language $L = \{\epsilon, ab, aabb, aaabbb, aaaabbbb, \dots\}$
- CFG G.

  $S \rightarrow aSb$

  $S \rightarrow \epsilon$

Construct CFG for $L = \{a^n b^n \mid n \geq 0\}$

Solution (continued)

- CFG G.
  $S \rightarrow aSb \mid \epsilon$
- Accepting ϵ. $\triangleright$ 1-step computation
  $S \Rightarrow \epsilon \qquad (\because S \rightarrow \epsilon)$
- Accepting ab. $\triangleright$ 2-steps computation
  $S \Rightarrow aSb \qquad (\because S \rightarrow aSb)$
  $\Rightarrow ab \qquad (\because S \rightarrow \epsilon)$
- Accepting $aabb$. $\triangleright$ 3-steps computation
  $S \Rightarrow aSb \qquad (\because S \rightarrow aSb)$
  $\Rightarrow aaSbb \qquad (\because S \rightarrow aSb)$
  $\Rightarrow aabb \qquad (\because S \rightarrow \epsilon)$
- Accepting $aaabbb$. $\triangleright$ 4-steps computation
  $S \Rightarrow aSb \qquad (\because S \rightarrow aSb)$
  $\Rightarrow aaSbb \qquad (\because S \rightarrow aSb)$
  $\Rightarrow aaaSbbb \qquad (\because S \rightarrow aSb)$
  $\Rightarrow aaabbb \qquad (\because S \rightarrow \epsilon)$

Construct CFGs

Problems

Construct CFGs to accept all strings from the following languages:

- $R = a^*$
- $R = a^+$
- $R = a^*b^*$
- $R = a^+b^+$
- $R = a^* \cup b^*$
- $R = (a \cup b)^*$
- $R = a^*b^*c^*$

Construct CFG for palindromes over $\{a, b\}$

Problem

- Construct a CFG that accepts all strings from the language $L = \{w \mid w = w^R \text{ and } \Sigma = \{a, b\}\}$

Construct CFG for palindromes over $\{a, b\}$

Problem

- Construct a CFG that accepts all strings from the language $L = \{w \mid w = w^R \text{ and } \Sigma = \{a, b\}\}$

Solution

- Language $L = \{\epsilon, a, b, aa, bb, aaa, aba, bab, bbb, aaaa, abba, baab, bbbb, \ldots\}$
- CFG G.
  $S \rightarrow aSa \mid bSb \mid a \mid b \mid \epsilon$

Construct CFG for palindromes over $\{a, b\}$

Solution (continued)

- CFG G. $S \to aSa \mid bSb \mid a \mid b \mid \epsilon$
- Accepting ϵ. $S \Rightarrow \epsilon$ ▷ 1 step
  Accepting a. $S \Rightarrow a$
  Accepting b. $S \Rightarrow b$
- Accepting aa. $S \Rightarrow aSa \Rightarrow aa$ ▷ 2 steps
  Accepting bb. $S \Rightarrow bSb \Rightarrow bb$
- Accepting aaa. $S \Rightarrow aSa \Rightarrow aaa$ ▷ 2 steps
  Accepting aba. $S \Rightarrow aSa \Rightarrow aba$
  Accepting bab. $S \Rightarrow bSb \Rightarrow bab$
  Accepting bbb. $S \Rightarrow bSb \Rightarrow bbb$
- Accepting $aaaa$. $S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aaaa$ ▷ 3 steps
  Accepting $abba$. $S \Rightarrow aSa \Rightarrow abSba \Rightarrow abba$
  Accepting $baab$. $S \Rightarrow bSb \Rightarrow baSab \Rightarrow baab$
  Accepting $bbbb$. $S \Rightarrow bSb \Rightarrow bbSbb \Rightarrow bbbb$

Construct CFG for non-palindromes over $\{a, b\}$

Problem

- Construct a CFG that accepts all strings from the language $L = \{w \mid w \neq w^R \text{ and } \Sigma = \{a, b\}\}$

Construct CFG for non-palindromes over $\{a, b\}$

Problem

- Construct a CFG that accepts all strings from the language $L = \{w \mid w \neq w^R \text{ and } \Sigma = \{a, b\}\}$

Solution

- Language $L = \{\epsilon, ab, ba, aab, abb, baa, bba, \ldots\}$
- CFG G.
  $S \rightarrow aSa \mid bSb \mid aAb \mid bAa$
  $A \rightarrow Aa \mid Ab \mid \epsilon$

Construct CFG for non-palindromes over $\{a, b\}$

Solution (continued)

- CFG G.

  $S \to aSa \mid bSb \mid aAb \mid bAa$

  $A \to Aa \mid Ab \mid \epsilon$

- Accepting $abbbbaaba$. ▷ 7-step derivation

  $S \Rightarrow aSa$

  $\Rightarrow abSba$

  $\Rightarrow abbAaba$

  $\Rightarrow abbAaaba$

  $\Rightarrow abbAbaaba$

  $\Rightarrow abbAbbaaba$

  $\Rightarrow abbbbaaba$

What is a context-free grammar (CFG)?

- Grammar = A set of rules for a language
- Context-free = LHS of productions have only 1 nonterminal

Definition

A context-free grammar (CFG) G is a 4-tuple
$G = (N, \Sigma, S, P)$, where,

1. N: A finite set (set of nonterminals/variables).
2. Σ: A finite set (set of terminals).
3. P : A finite set of productions/rules of the form $A \rightarrow \alpha$, $A \in N, \alpha \in (N \cup \Sigma)^*$. ▷ Time (computation) ▷ Space (computer memory)
4. S: The start nonterminal (belongs to N).

Derivation, acceptance, and rejection

Definitions

- Derivation.
  $\alpha A\gamma \Rightarrow \alpha\beta\gamma \qquad (\because A \rightarrow \beta)$ ▷ 1-step derivation
- Acceptance.
  G accepts string w iff
  $S \Rightarrow_G^* w$ ▷ multistep derivation
- Rejection.
  G rejects string w iff
  $S \not\Rightarrow_G^* w$ ▷ no derivation

What is a context-free language (CFL)?

Definition

- If $G = (N, \Sigma, S, P)$ is a CFG, the language generated by G is $L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$
- A language L is a context-free language (CFL) if there is a CFG G with $L = L(G)$.

Construct CFG for $L = \{w \mid n_a(w) = n_b(w)\}$

Problem

- Construct a CFG that accepts all strings from the language $L = \{w \mid n_a(w) = n_b(w)\}$

Construct CFG for $L = \{w \mid n_a(w) = n_b(w)\}$

Problem

- Construct a CFG that accepts all strings from the language $L = \{w \mid n_a(w) = n_b(w)\}$

Solution

- Language $L = \{\epsilon, ab, ba, ba, aabb, abab, abba, bbaa, \ldots\}$
- CFGs.
  1. $S \rightarrow SaSbS \mid SbSaS \mid \epsilon$
  2. $S \rightarrow aSbS \mid bSaS \mid \epsilon$
  3. $S \rightarrow aSb \mid bSa \mid SS \mid \epsilon$
- Derive the following 4-letter strings from G.
  $aabb, abab, abba, bbaa, baba, baab$
- Write G as a 4-tuple.
- What is the meaning/interpretation/logic of the grammar?

Construct CFGs

Problem

Construct CFGs that accepts all strings from the following languages

1. $L = \{w \mid n_a(w) > n_b(w)\}$
2. $L = \{w \mid n_a(w) = 2n_b(w)\}$
3. $L = \{w \mid n_a(w) \neq n_b(w)\}$

Construct CFGs

Problem

Construct CFGs that accepts all strings from the following languages

1. $L = \{w \mid n_a(w) > n_b(w)\}$
2. $L = \{w \mid n_a(w) = 2n_b(w)\}$
3. $L = \{w \mid n_a(w) \neq n_b(w)\}$

Solutions

1. $S \to aS \mid bSS \mid SSb \mid SbS \mid a$
2. $S \to SS \mid bAA \mid AbA \mid AAb \mid \epsilon$
   $A \to aS \mid SaS \mid Sa \mid a$
3. ?

Union, concatenation, and star are closed on CFL's

Properties

- If L_1 and L_2 are context-free languages over an alphabet Σ, then $L_1 \cup L_2$, $L_1 L_2$, and L_1^* are also CFL's.

Union, concatenation, and star are closed on CFL's

Properties

- If L_1 and L_2 are context-free languages over an alphabet Σ, then $L_1 \cup L_2$, $L_1 L_2$, and L_1^* are also CFL's.

Construction

Let $G_1 = (N_1, \Sigma, S_1, P_1)$ be CFG for L_1.
Let $G_2 = (N_2, \Sigma, S_2, P_2)$ be CFG for L_2.

- Union.
  Let $G_u = (N_u, \Sigma, S_u, P_u)$ be CFG for $L_1 \cup L_2$.
  $N_u = N_1 \cup N_2 \cup \{S_u\}$; $P_u = P_1 \cup P_2 \cup \{S_u \to S_1 \mid S_2\}$
- Concatenation.
  Let $G_c = (N_c, \Sigma, S_c, P_c)$ be CFG for $L_1 L_2$.
  $N_u = N_1 \cup N_2 \cup \{S_c\}$; $P_c = P_1 \cup P_2 \cup \{S_c \to S_1 S_2\}$
- Kleene star.
  Let $G_s = (N_s, \Sigma, S_s, P_s)$ be CFG for L_1^*.
  $N_s = N_1 \cup \{S_s\}$; $P_s = P_1 \cup \{S_s \to S_s S_1 \mid \epsilon\}$

Union is closed on CFL's

Problem

- If L_1 and L_2 are CFL's then $L_3 = L_1 \cup L_2$ is a CFL.
- If L_1 and $L_3 = L_1 \cup L_2$ are CFL's, is L_2 a CFL?

Union is closed on CFL's

Problem

- If L_1 and L_2 are CFL's then $L_3 = L_1 \cup L_2$ is a CFL.
- If L_1 and $L_3 = L_1 \cup L_2$ are CFL's, is L_2 a CFL?

Solution

- L_2 may or may not be a CFL.

  $L_1 = \Sigma^*$ ▷ CFL

  $L_3 = L_1 \cup L_2 = \Sigma^*$ ▷ CFL

  $L_2 = \{a^n \mid n \text{ is prime}\}$ ▷ Non-CFL

Reversal is closed on CFL's

Property

- If L is a CFL, then L^R is a CFL.

Reversal is closed on CFL's

Property

- If L is a CFL, then L^R is a CFL.

Construction

- Let $G = (N, \Sigma, S, P)$ be CFG for L.
  Let $G_r = (N, \Sigma, S, P_r)$ be CFG for L^R. Then
- Reversal.
  P_r = productions from P such that all symbols on the right hand side of every production is reversed.
  i.e., If $A \rightarrow \alpha$ is in P, then $A \rightarrow \alpha^R$ is in P_r
- Example.
  Grammar for accepting L is $S \rightarrow aSb \mid ab$.
  Grammar for accepting L^R is $S \rightarrow bSa \mid ba$.

Intersection is not closed on CFL's

Problem

- Show that L_1, L_2 are CFL's and $L = L_1 \cap L_2$ is a non-CFL.

$$\begin{aligned} L &= \{a^i b^j c^k \mid i = j \text{ and } j = k\} \\ &= \{a^i b^i c^k \mid i, k \geq 0\} \cap \{a^i b^j c^j \mid i, j \geq 0\} \\ &L_1 \cap L_2 \end{aligned}$$

Intersection is not closed on CFL's

Problem

- Show that L_1, L_2 are CFL's and $L = L_1 \cap L_2$ is a non-CFL.

$$\begin{aligned} L &= \{a^i b^j c^k \mid i = j \text{ and } j = k\} \\ &= \{a^i b^i c^k \mid i, k \geq 0\} \cap \{a^i b^j c^j \mid i, j \geq 0\} \\ &\quad L_1 \cap L_2 \end{aligned}$$

Solution

- L_1 is a CFL.
  $L_1 = \{a^i b^i c^k \mid i, k \geq 0\} = \{a^i b^i \mid i \geq 0\}\{c^k \mid k \geq 0\}$
  $= L_3 L_4 =$ CFL $\quad (\because L_3, L_4$ are CFL's$)$
- L_2 is a CFL.
  $L_2 = \{a^i b^j c^j \mid i, j \geq 0\} = \{a^i \mid i \geq 0\}\ \{b^j c^j \mid j \geq 0\}$
  $= L_5 L_6 =$ CFL $\quad (\because L_5, L_6$ are CFL's$)$
- L is a non-CFL.
  Use pumping lemma for CFL's.

Complementation is not closed on CFL's

Problem

- Show that complementation is not closed on CFL's.

Complementation is not closed on CFL's

Problem

- Show that complementation is not closed on CFL's.

Solution

Proof by contradiction.

- Suppose complementation is closed under CFL's.
  i.e., if L is a CFL, then $\bar{L}$ is a CFL.
- Consider the equation $L_1 \cap L_2 = \overline{(\overline{L_1} \cup \overline{L_2})}$.
  Closure on complementation implies closure on intersection.
- But, intersection is not closed on CFL's.
- Contradiction!
- Hence, complementation is not closed on CFL's.

Complementation is not closed on CFL's

Problem

- Show that $\bar{L}$ is a CFL and L is a non-CFL.

$$\bar{L} = \Sigma^* - \{ww \mid w \in \Sigma^*\} = \Sigma^* - L$$

Complementation is not closed on CFL's

Problem

- Show that $\bar{L}$ is a CFL and L is a non-CFL.

$$\bar{L} = \Sigma^* - \{ww \mid w \in \Sigma^*\} = \Sigma^* - L$$

Solution

- $\bar{L}$ is a CFL.
  $S \to A \mid B \mid AB \mid BA$
  $A \to EAE \mid a$
  $B \to EBE \mid b$
  $E \to a \mid b$
  Why does this grammar work?
- L is a non-CFL.
  Use pumping lemma for CFL's.

Set difference is not closed on CFL's

Problem

- Show that set difference is not closed on CFL's.

Set difference is not closed on CFL's

Problem

- Show that set difference is not closed on CFL's.

Solution

Proof by contradiction.

- Suppose set difference is closed under CFL's.
  i.e., if L_1, L_2 are CFL's, then $L_1 - L_2$ is a CFL.
- Consider the equation $L_1 \cap L_2 = L_1 - (L_1 - L_2)$.
  Closure on set difference implies closure on intersection.
- But, intersection is not closed on CFL's.
- Contradiction!
- Hence, set difference is not closed on CFL's.

Summary: Closure properties of CFL's

Operation	Closed on CFL's?
Union $(L_1 \cup L_2)$	✓
Concatenation $(L_1 L_2)$	✓
Kleene star (L^*)	✓
Reversal (L^R)	✓
Intersection $(L_1 \cap L_2)$	✗
Complementation $(\bar{L})$	✗
Set difference $(L_1 - L_2)$	✗

Construct CFG for $L = \{a^i b^j c^k \mid j = i + k\}$

Problem

- Construct a CFG that accepts all strings from the language $L = \{a^i b^j c^k \mid j = i + k\}$

Construct CFG for $L = \{a^i b^j c^k \mid j = i + k\}$

Problem

- Construct a CFG that accepts all strings from the language $L = \{a^i b^j c^k \mid j = i + k\}$

Solution

- Language $L = \{\epsilon, ab, bc, a^2b^2, b^2c^2, ab^2c, \ldots\}$
- $L = \{a^i b^j c^k \mid j = i + k\}$
  $= \{a^i b^{i+k} c^k\}$ ($\because$ substitute for j)
  $= \{a^i b^i b^k c^k\}$ ($\because$ expand)
  $= \{a^i b^i\}\{b^k c^k\}$ ($\because$ split the concatenated languages)
  $= L_1 L_2$
- Solve the problem completely by constructing CFG's for L_1, L_2, and then $L_1 L_2$.
- Divide-and-conquer. We can solve a complicated problem if we can break the problem into several simpler subproblems and solve those simpler problems.
- Construct CFG for the variant where $j \neq i + k$.

Construct CFG for $L = \{a^i b^j c^k \mid j \neq i + k\}$

Problem

- Construct a CFG that accepts all strings from the language $L = \{a^i b^j c^k \mid j \neq i + k\}$

Construct CFG for $L = \{a^i b^j c^k \mid j \neq i + k\}$

Problem

- Construct a CFG that accepts all strings from the language $L = \{a^i b^j c^k \mid j \neq i + k\}$

Solution

- Language $L = \{\epsilon, a, b, c, ac, a^2, b^2, c^2, \ldots\}$
- $L = \{a^i b^j c^k \mid j \neq i + k\}$
  $= \{a^i b^j c^k \mid j > (i + k)\} \cup \{a^i b^j c^k \mid j < (i + k)\}$
  $= L_1 \cup L_2$
- Can we represent L_1 and L_2 using simpler languages?

Construct CFG for $L = \{a^i b^j c^k \mid j \neq i + k\}$

Solution (continued)

- Case 1. $L_1 = \{a^i b^j c^k \mid j > i + k\}$
  $= \{a^i b^j c^k \mid j = i + m + k \text{ and } m \geq 1\}$
  $= \{a^i b^{i+m+k} c^k \mid m \geq 1\}$
  $= \{a^i b^i\} \cdot \{b^m \mid m \geq 1\} \cdot \{b^k c^k\}$
  $= \{a^i b^i\} \cdot \{bb^n\} \cdot \{b^k c^k\}$
  $= L_{11} \cdot L_{12} \cdot L_{13}$
  We know how to construct CFG's for L_{11}, L_{12}, L_{13}
- Case 2. $L_2 = \{a^i b^j c^k \mid j < i + k\}$
  $= \{a^i b^j c^k \mid j < i \text{ or } i \leq j < i + k\}$
  $= \{a^i b^j c^k \mid j < i\} \cup \{a^i b^j c^k \mid i \leq j < i + k\}$
  $= L_{21} \cup L_{22}$
  How to proceed?

Construct CFG for $L = \{a^i b^j c^k \mid j \neq i + k\}$

Solution (continued)

- Case 3. $L_{21} = \{a^i b^j c^k \mid j < i\}$
  $= \{a^i b^j c^k \mid i = m + j \text{ and } m \geq 1\}$
  $= \{a^{m+j} b^j c^k \mid m \geq 1\}$
  $= \{a^m \mid m \geq 1\} \cdot \{a^j b^j\} \cdot \{c^k\}$
  $= L_{211} \cdot L_{212} \cdot L_{213}$
  We know how to construct CFG's for $L_{211}, L_{212}, L_{213}$
- Case 4. $L_{22} = \{a^i b^j c^k \mid i \leq j < i + k\}$
  $= \{a^i b^j c^k \mid j \geq i \text{ and } k > j - i\}$
  $= \{a^i b^{i+(j-i)} c^{(j-i)+m} \mid (j - i) \geq 0 \text{ and } m \geq 1\}$
  $= \{a^i b^i\} \cdot \{b^{j-i} c^{j-i} \mid (j - i) \geq 0\} \cdot \{c^m \mid m \geq 1\}$
  $= \{a^i b^i\} \cdot \{b^i c^i\} \cdot \{c^m \mid m \geq 1\}$
  $= L_{221} \cdot L_{222} \cdot L_{223}$
  We know how to construct CFG's for $L_{221}, L_{222}, L_{223}$

Construct CFG for $bba(ab)^* \mid (ab \mid ba^*b)^*ba$

Problem

- Construct a CFG that accepts all strings from the language corresponding to R.E. $bba(ab)^* \mid (ab \mid ba^*b)^*ba$.

Construct CFG for $bba(ab)^* \mid (ab \mid ba^*b)^*ba$

Problem

- Construct a CFG that accepts all strings from the language corresponding to R.E. $bba(ab)^* \mid (ab \mid ba^*b)^*ba$.

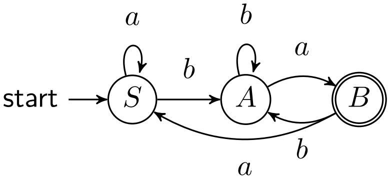
Solution

- Language $L = \{ba, bba, abba, bbba, \ldots\}$
  This is a regular language.
- CFG G.
  $S \to S_1 \mid S_2$
  $S_1 \to S_1ab \mid bba$ ▷ Generates $bba(ab)^*$
  $S_2 \to TS_2 \mid ba$ ▷ Generates $(ab \mid ba^*b)^*ba$
  $T \to ab \mid bUb$ ▷ Generates $ab \mid ba^*b$
  $U \to aU \mid \epsilon$ ▷ Generates a^*

Construct CFG for strings of a DFA

Problem

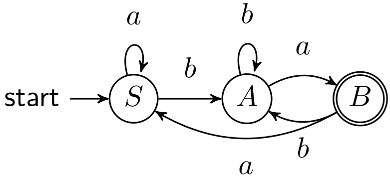
- Construct a CFG that accepts all strings accepted by the following DFA.



Construct CFG for strings of a DFA

Problem

- Construct a CFG that accepts all strings accepted by the following DFA.



Solution

- Language $L = \{(a \mid b)^* ba\}$ $\triangleright$ Strings ending with ba
  $= \{ba, aba, bba, aaba, abba, baba, bbba, \ldots\}$
  This is a regular language.
- How to construct CFG for this DFA?
  Approach 1: Compute R.E. Construct CFG for the R.E.
  Approach 2: Construct CFG from the DFA using transitions.

Construct CFG for strings of a DFA

Solution (continued)

- Idea.
  For every transition $\delta(Q, a) = R$, add a production $Q \to aR$.
  What does this mean? Why should it work?

Construct CFG for strings of a DFA

Solution (continued)

- Idea.
  For every transition $\delta(Q, a) = R$, add a production $Q \to aR$.
  What does this mean? Why should it work?
- CFG. $\triangleright$ 3 states = 3 nonterminals
  $S \to aS \mid bA$
  $A \to bA \mid aB$
  $B \to bA \mid aS \mid \epsilon$ $\triangleright$ ϵ-production for halting state
- Accepting bbaaba.
  $S \xrightarrow{b} A \xrightarrow{b} A \xrightarrow{a} B \xrightarrow{a} S \xrightarrow{b} A \xrightarrow{a} B$
  $S \Rightarrow bA \Rightarrow bbA \Rightarrow bbaB \Rightarrow bbaaS \Rightarrow bbaabA \Rightarrow bbaabaB$
  $\Rightarrow bbaaba$

What is a regular grammar/language?

Definitions

- A context-free grammar $G = (N, \Sigma, S, P)$ is called a regular grammar if every production is of the form $A \to aB$ or $A \to \epsilon$, where $A, B \in N$ and $a \in \Sigma$.
- A language $L \in \Sigma^*$ is called a regular language iff $L = L(G)$ for some regular grammar G.

Construct CFG for understanding human languages

Problem

- Construct a CFG to understand some structures in the English language.

Solution

- CFG:
  $\langle \text{Sentence} \rangle \rightarrow \langle \text{NounPhrase} \rangle \ \langle \text{VerbPhrase} \rangle$
  $\langle \text{NounPhrase} \rangle \rightarrow \langle \text{ComplexNoun} \rangle | \langle \text{ComplexNoun} \rangle \langle \text{PrepPhrase} \rangle$
  $\langle \text{VerbPhrase} \rangle \rightarrow \langle \text{ComplexVerb} \rangle | \langle \text{ComplexVerb} \rangle \langle \text{PrepPhrase} \rangle$
  $\langle \text{PrepPhrase} \rangle \rightarrow \langle \text{Prep} \rangle \ \langle \text{ComplexNoun} \rangle$
  $\langle \text{ComplexNoun} \rangle \rightarrow \langle \text{Article} \rangle \ \langle \text{Noun} \rangle$
  $\langle \text{ComplexVerb} \rangle \rightarrow \langle \text{Verb} \rangle \mid \langle \text{Verb} \rangle \ \langle \text{NounPhrase} \rangle$
  $\langle \text{Article} \rangle \rightarrow \text{a} \mid \text{the}$
  $\langle \text{Noun} \rangle \rightarrow \text{boy} \mid \text{girl} \mid \text{flower}$
  $\langle \text{Verb} \rangle \rightarrow \text{touches} \mid \text{likes} \mid \text{sees}$
  $\langle \text{Prep} \rangle \rightarrow \text{with}$

Construct CFG for understanding human languages

Solution (continued)

- Accepting "a girl likes".
  $\langle \text{Sentence} \rangle \Rightarrow \langle \text{NounPhrase} \rangle \langle \text{VerbPhrase} \rangle$
  $\Rightarrow \langle \text{ComplexNoun} \rangle \langle \text{VerbPhrase} \rangle$
  $\Rightarrow \langle \text{Article} \rangle \langle \text{Noun} \rangle \langle \text{VerbPhrase} \rangle$
  $\Rightarrow$ a $\langle \text{Noun} \rangle \langle \text{VerbPhrase} \rangle$
  $\Rightarrow$ a girl $\langle \text{VerbPhrase} \rangle$
  $\Rightarrow$ a girl $\langle \text{ComplexVerb} \rangle$
  $\Rightarrow$ a girl $\langle \text{Verb} \rangle$
  $\Rightarrow$ a girl likes
- Derive "a girl with a flower likes the boy".

Construct CFG for strings with valid parentheses

Problem

- Construct a CFG that accepts all strings from the language $L = \{\epsilon, (), ()(), (()), ()()(), (()()), ()(()), (())(), ((())), \ldots\}$

Construct CFG for strings with valid parentheses

Problem

- Construct a CFG that accepts all strings from the language $L = \{\epsilon, (), ()(), (()), ()()(), (()()), ()(()), (())(), ((())), \ldots\}$

Solution

- Applications. Compilers check for syntactic correctness in:
  1. Computer programs written by you that possibly contain nested code blocks with { }, ( ), and [ ].
  2. Web pages written by you that contain nested code blocks with `<div></div>`, `<table></table>`, and `<ul></ul>`.
- Language $L = \{w \mid w \in \{(,)\}^*$ such that $n_((w) = n_)(w)$ and and in any prefix $p_{i<|w|}$ of w, $n_((p_i) \geq n_)(p_i)\}$
- What is the CFG?

Construct CFG for strings with valid parentheses

Solution (continued)

Multiple correct ways to write the CFG:

1. $S \rightarrow S(S)S \mid \epsilon$
2. $S \rightarrow SS \mid (S) \mid \epsilon$
3. $S \rightarrow S(S) \mid \epsilon$
4. $S \rightarrow (S)S \mid \epsilon$
5. $S \rightarrow SR) \mid \epsilon$
   $R \rightarrow (\mid RR)$
6. $S \rightarrow (RS \mid \epsilon$
   $R \rightarrow) \mid (RR$

- Are some CFG's better than the others?
  If so, better in what?

Construct CFG for valid arithmetic expressions

Problem

- Construct a CFG that accepts all valid arithmetic expressions from $\Sigma = \{(,), +, \times, n\}$, where n represents any integer.

Construct CFG for valid arithmetic expressions

Problem

- Construct a CFG that accepts all valid arithmetic expressions from $\Sigma = \{(,), +, \times, n\}$, where n represents any integer.

Solution

- Language $L = \{15 + 85, 57 \times 3, (27 + 46) \times 10, \ldots\}$
- Abstraction: Denote n to mean any integer.
  Valid expressions: $(n + n) + n \times n$, etc
  Invalid expressions: $+n$, $(n+)n$, $()$, $n \times n)$, etc
- Hint: Use some ideas from the parenthesis problem

Construct CFG for valid arithmetic expressions

Solution (continued)

Multiple correct ways to write the CFG:

1. $E \to E + E \mid E \times E \mid (\ E\ ) \mid n$
2. $E \to E + T \mid T$ ▷ expression

   $T \to T \times F \mid F$ ▷ term

   $F \to (\ E\ ) \mid n$ ▷ factor
3. $E \to TE'$

   $E' \to +TE' \mid \epsilon$

   $T \to FT'$

   $T' \to \times FT' \mid \epsilon$

   $F \to (\ E\ ) \mid n$

- Can you derive $(n \times n)$?
- Are some CFG's better than the others? If so, better in what?

What is a derivation?

Definition

- A derivation in a context-free grammar is a leftmost derivation (LMD) if, at each step, a production is applied to the leftmost variable-occurrence in the current string. A rightmost derivation (RMD) is defined similarly.

Example

- CFG: $E \to E + E \mid E \times E \mid (\ E \ ) \mid n$
  Accepting $n + (n)$.
  LMD: $E \Rightarrow E + E \Rightarrow n + E \Rightarrow n + (E) \Rightarrow n + (n)$
  RMD: $E \Rightarrow E + E \Rightarrow E + (E) \Rightarrow E + (n) \Rightarrow n + (n)$

What is an ambiguous grammar?

Definition

- A context-free grammar G is ambiguous if for at least one $w \in L(G)$, w has more than one derivation tree (or, equivalently, more than one leftmost derivation).
- Intuition: A CFG is ambiguous if it generates a string in several different ways.

Arithmetic expression: Ambiguous grammar

Problem

- Show that the following CFG is ambiguous:
  $E \to E + E \mid E \times E \mid (\ E\ ) \mid n$

Arithmetic expression: Ambiguous grammar

Problem

- Show that the following CFG is ambiguous:
  $E \to E + E \mid E \times E \mid (\ E \ ) \mid n$

Solution

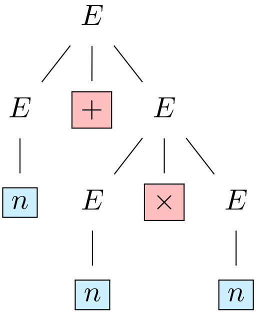
- Consider the strings $n + n \times n$ or $n + n + n$.
  There are two derivation trees for each of the strings.
- Accepting $n + n \times n$.
  LMD 1: $E \Rightarrow E + E \Rightarrow n + E \Rightarrow n + E \times E \Rightarrow n + n \times E$
  $\Rightarrow n + n \times n$
  LMD 2: $E \Rightarrow E \times E \Rightarrow E + E \times E \Rightarrow n + E \times E \Rightarrow n + n \times E$
  $\Rightarrow n + n \times n$
- Accepting $n + n + n$.
  LMD 1: $E \Rightarrow E + E \Rightarrow n + E \Rightarrow n + E + E \Rightarrow n + n + E$
  $\Rightarrow n + n + n$
  LMD 2: $E \Rightarrow E + E \Rightarrow E + E + E \Rightarrow n + E + E \Rightarrow n + n + E$
  $\Rightarrow n + n + n$

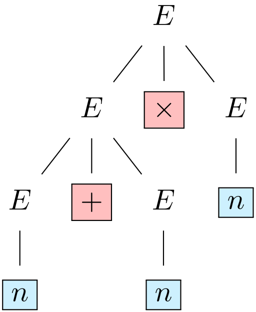
Arithmetic expression: Ambiguous grammar

Solution (continued)

Two derivation (or parse) trees $\implies$ Ambiguity
(Reason 1: The precedence of different operators isn't enforced.)

- LMD 1: $E \Rightarrow E + E \Rightarrow n + E \Rightarrow n + E \times E \Rightarrow n + n \times E \Rightarrow n + n \times n$
- LMD 2: $E \Rightarrow E \times E \Rightarrow E + E \times E \Rightarrow n + E \times E \Rightarrow n + n \times E \Rightarrow n + n \times n$





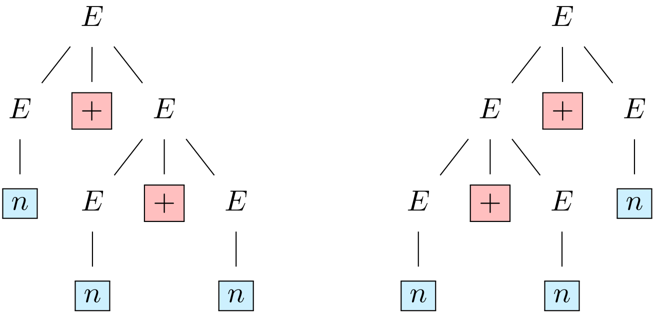
Arithmetic expression: Ambiguous grammar

Solution (continued)

Two derivation (or parse) trees $\Longrightarrow$ Ambiguity

(Reason 2: Order of operators of same precedence isn't enforced.)

- LMD 1: $E \Rightarrow E + E \Rightarrow n + E \Rightarrow n + E + E \Rightarrow n + n + E \Rightarrow n + n + n$
- LMD 2: $E \Rightarrow E + E \Rightarrow E + E + E \Rightarrow n + E + E \Rightarrow n + n + E \Rightarrow n + n + n$



Arithmetic expression: Ambiguous grammar

Problem

- Consider the following ambiguous grammar:
  $E \to E + E \mid E \times E \mid (\ E\ ) \mid n$
  How many different derivations (or LMDs) are possible for the string $n + n + \cdots + n$, where n is repeated k times?

Arithmetic expression: Ambiguous grammar

Problem

- Consider the following ambiguous grammar:
  $E \to E + E \mid E \times E \mid (\ E \ ) \mid n$
  How many different derivations (or LMDs) are possible for the string $n + n + \cdots + n$, where n is repeated k times?

Solution

- Let $d(k) =$ number of derivations for k operands. Then
  $d(1) = 1$
  $d(2) = 1$
  $d(3) = 2$
  $d(4) = 5$ How?
- How do you compute $d(k)$?
  $d(k) = \sum_{i=1}^{k-1} d(i)d(k-i)$

If-else ladder: Ambiguous grammar

Problem

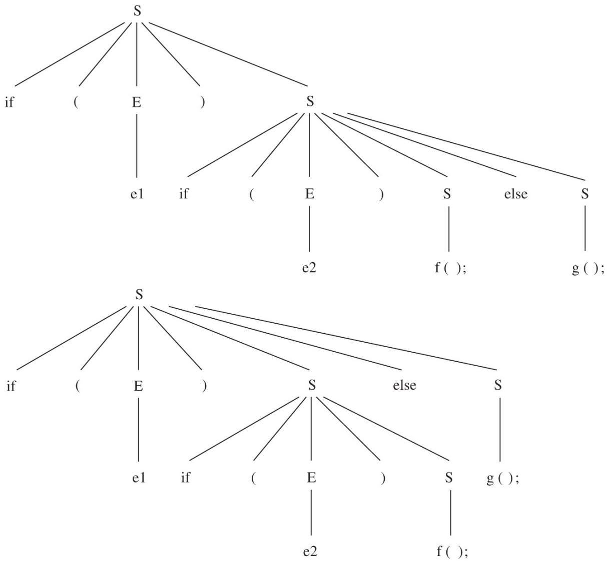
- Show that the following CFG is ambiguous:
  $S \rightarrow$ if $(\ E\ )\ S \mid$ if $(\ E\ )\ S$ else $S \mid O$
  where, $S =$ statement, $E =$ expression, $O =$ other statement.

Solution

- Consider the string: if (e_1) if (e_2) F(); else G();
  There are two derivation trees for the string.
- Can you identify the two derivation trees for the string?

If-else ladder: Ambiguous grammar

Solution (continued)



What is the output of this program?

C++ program:

```
#include <iostream>
using namespace std;

int main()
{
    if (true)
        if (false)
            ;
    else
        cout << "Hi!";

    return 0;
}
```

What is the output of this program?

C++ program:

```
#include <iostream>
using namespace std;

int main()
{
    if (true)
        if (false)
            ;
    else
        cout << "Hi!";

    return 0;
}
```

Output:

Hi!

If-else ladder: Unambiguous grammar

Problem

- Can you come up with an unambiguous grammar for the language accepted by the following ambiguous grammar?
  $S \rightarrow$ if (E) S | if (E) S else S | O
  where, $S =$ statement, $E =$ expression, $O =$ other statement.

Solution

- $S \rightarrow S_1 \mid S_2$
  $S_1 \rightarrow$ if (E) S_1 else S_1 | O
  $S_2 \rightarrow$ if (E) S | if (E) S_1 else S_2
- How do you prove that the grammar is really unambiguous?

What is an inherently ambiguous language?

Definition

- A context-free language is called inherently ambiguous if there exists no unambiguous grammar to generate the language.

What is an inherently ambiguous language?

Definition

- A context-free language is called inherently ambiguous if there exists no unambiguous grammar to generate the language.

Examples

Proofs?

- $L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$
- $L = \{a^i b^i c^j d^j\} \cup \{a^i b^j c^j d^i\}$

Language generated by a grammar

Problem

- Prove that the following grammar G generates all strings of balanced parentheses and only such strings.
  $S \rightarrow (S)S \mid \epsilon$

Language generated by a grammar

Problem

- Prove that the following grammar G generates all strings of balanced parentheses and only such strings.
  $S \rightarrow (S)S \mid \epsilon$

Solution

- $L(G) =$ language generated by the grammar G.
  $L =$ language of balanced parentheses.
- Show that $L(G) = L$. Two cases.
  Case 1. Show that every string derivable from S is balanced. i.e., $L(G) \subseteq L$.
  Case 2. Show that every balanced string is derivable from S. i.e., $L \subseteq L(G)$.

Language generated by a grammar

Solution (continued)

Case 1. Show that every string derivable from S is balanced.

Let $n =$ number of steps in derivation.

- Basis.

  The only string derivable from S in 1 step is ϵ and ϵ is balanced.
- Induction.

  Suppose all strings with derivation fewer than n steps produce balanced parentheses.

  Consider a LMD of at most n steps.

  That derivation must be of the form

  $$S \Rightarrow (S)S \Rightarrow^* (x)S \Rightarrow^* (x)y \qquad \text{(LMD)}$$

  Derivations of x and y take fewer than n steps.

  So, x and y are balanced.

  Therefore, the string $(x)y$ must be balanced.

Language generated by a grammar

Solution (continued)

Case 2. Show that every balanced string is derivable from S.

Let $2n =$ length of a balanced string.

- Basis.

  A 0-length string is ϵ, which is balanced.

- Induction.

  Assume that every balanced string of length less than $2n$ is derivable from S. Consider a balanced string w of length $2n$ such that $n \geq 1$. String w must begin with a left parenthesis. Let (x) be the shortest nonempty prefix of w having an equal number of left and right parentheses. Then, w can be written as $w = (x)y$, where, both x and y are balanced. Since x and y are of length less than $2n$, they are derivable from S. Thus, we can find a derivation of the form

  $S \Rightarrow (S)S \Rightarrow^* (x)S \Rightarrow^* (x)y$ (LMD)

  proving that $w = (x)y$ must also be derivable from S.

What is Chomsky normal form (CNF)?

Definition

- A context-free grammar is said to be in Chomsky normal form (CNF) if every production is of one of these three types:
  $A \to BC$ (where B, C are nonterminals and they cannot be the start nonterminal S)
  $A \to a$ (where a is a terminal symbol)
  $S \to \epsilon$
- Why should we care for CNF?
  For every context-free grammar G, there is another CFG G_{CNF} in Chomsky normal form such that $L(G_{\text{CNF}}) = L(G)$.

Example

- $S \to AA \mid \epsilon$
  $A \to AA \mid a$

Converting a CFG to CNF

Algorithm rule	Before rule	After rule
1. Start nonterminal must not appear on the RHS	$S \to ASABS$	$S_0 \to S$ $S \to ASABS$
2. Remove productions like $A \to \epsilon$	$R \to ARA$ $A \to a \mid \epsilon$	$R \to ARA$ $R \to AR \mid RA \mid A$ $A \to a$
3. Remove productions like $A \to B$	$A \to B$ $B \to CDD$	$A \to CDD$
4. Convert to CNF	$A \to BCD$	$A \to BC'$ $C' \to CD$

CFG-TO-CNF(G)

1. Start nonterminal must not appear on RHS
2. Remove ϵ productions
3. Remove unit productions
4. Convert to CNF

Converting a CFG to CNF

Problem

- Convert the following CFG to CNF.

  $S \to ASA \mid aB$

  $A \to B \mid S$

  $B \to b \mid \epsilon$

Converting a CFG to CNF

Problem

- Convert the following CFG to CNF.

  $S \to ASA \mid aB$

  $A \to B \mid S$

  $B \to b \mid \epsilon$

Solution

- Start nonterminal must not appear on the right hand side

  $S_0 \to S$

  $S \to ASA \mid aB$

  $A \to B \mid S$

  $B \to b \mid \epsilon$

- Remove $B \to \epsilon$

  $S_0 \to S$

  $S \to ASA \mid aB \mid a$

  $A \to B \mid S \mid \epsilon$

  $B \to b$

Converting a CFG to CNF

Solution (continued)

- Remove $A \to \epsilon$

  $S_0 \to S$
  $S \to ASA \mid SA \mid AS \mid S \mid aB \mid a$
  $A \to B \mid S$
  $B \to b$

- Remove $A \to B$

  $S_0 \to S$
  $S \to ASA \mid SA \mid AS \mid S \mid aB \mid a$
  $A \to S \mid b$
  $B \to b$

- Remove $S \to S$ $\triangleright$ Do nothing

  $S_0 \to S$
  $S \to ASA \mid SA \mid AS \mid aB \mid a$
  $A \to S \mid b$
  $B \to b$

Converting a CFG to CNF

Solution (continued)

- Remove $A \to S$

  $S_0 \to S$
  $S \to ASA \mid SA \mid AS \mid aB \mid a$
  $A \to ASA \mid SA \mid AS \mid aB \mid a \mid b$
  $B \to b$

- Remove $S_0 \to S$

  $S_0 \to ASA \mid SA \mid AS \mid aB \mid a$
  $S \to ASA \mid SA \mid AS \mid aB \mid a$
  $A \to ASA \mid SA \mid AS \mid aB \mid a \mid b$
  $B \to b$

- Convert $ASA \to AA_1$

  $S_0 \to AA_1 \mid SA \mid AS \mid aB \mid a$
  $S \to AA_1 \mid SA \mid AS \mid aB \mid a$
  $A \to AA_1 \mid SA \mid AS \mid aB \mid a \mid b$
  $A_1 \to SA$
  $B \to b$

Converting a CFG to CNF

Solution (continued)

- Introduce $A_2 \to a$
  $S_0 \to AA_1 \mid SA \mid AS \mid A_2B \mid a$
  $S \to AA_1 \mid SA \mid AS \mid A_2B \mid a$
  $A \to AA_1 \mid SA \mid AS \mid A_2B \mid a \mid b$
  $A_1 \to SA$
  $A_2 \to a$
  $B \to b$
- This grammar is now in Chomsky normal form.

What is Griebach normal form (GNF)?

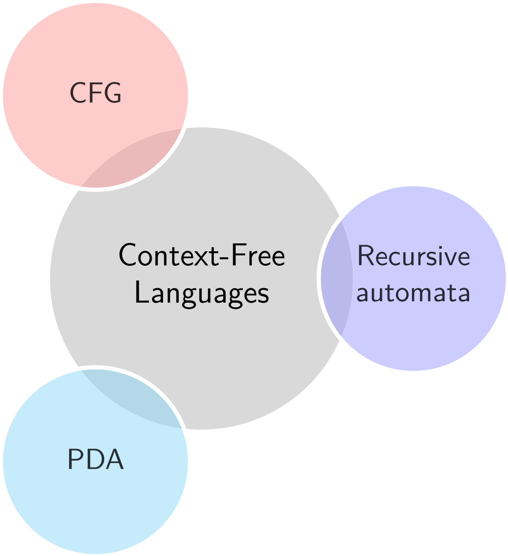
Definition

- A context-free grammar is said to be in Griebach normal form (GNF) if every production is of the following type:
  $A \rightarrow aA_1A_2\ldots A_d$ (where a is a terminal symbol and $A_1, A_2, \ldots, A_d$ are nonterminals)
  $S \rightarrow \epsilon$ (Not always included)
- Why should we care for GNF?
  For every context-free grammar G, there is another CFG G_{GNF} in Griebach normal form such that $L(G_{\mathsf{GNF}}) = L(G)$.
  A string of length n has a derivation of exactly n steps.

Example

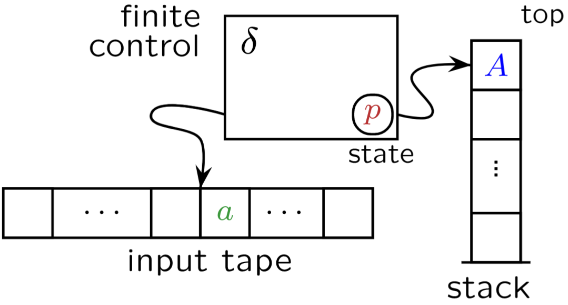
- $S \rightarrow aA \mid bB$
  $B \rightarrow bB \mid b$
  $A \rightarrow aA \mid a$

Equivalence of different computation models



Pushdown Automata (PDA)

Pushdown automaton



Source: Wikipedia

- PDA has access to a stack of unlimited memory

What is a pushdown automaton (PDA)?

- Nondetermistic = Events cannot be determined precisely
- Pushdown = Using stack of infinite memory
- Automaton = Computing machine

What is a pushdown automaton (PDA)?

- Nondetermistic = Events cannot be determined precisely
- Pushdown = Using stack of infinite memory
- Automaton = Computing machine

Definition

A pushdown automaton (PDA) P is a 6-tuple
$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where,

1. Q: A finite set (set of states).
2. Σ: A finite set (input alphabet).
3. Γ: A finite set (stack alphabet).
4. $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon)$ is the transition function.

   $\triangleright$ Time (computation)
5. q_0: The start state (belongs to Q).
6. F: The set of accepting/final states, where $F \subseteq Q$.

Stack $\triangleright$ Space (computer memory)

What is a context-free language?

Definition

- A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ accepts a string $w \in \Sigma^*$ iff

$$(q_0, w, \$) \vdash_M^* (q_f, \epsilon, \alpha)$$

  for some $\alpha \in \Gamma^*$ and some $q_f \in F$.
  A PDA rejects a string iff it does not accept it.
- We say that a PDA M accepts a language L if $L = \{w \mid M \text{ accepts } w\}$.
- A language is called a context-free language if some PDA accepts or recognizes it.

Construct PDA for $L = \{a^n b^n\}$

Problem

- Construct a PDA that accepts all strings from the language $L = \{a^n b^n\}$

Construct PDA for $L = \{a^n b^n\}$

Problem

- Construct a PDA that accepts all strings from the language $L = \{a^n b^n\}$

Solution

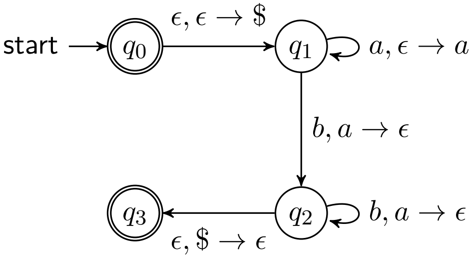
PDA()

1. while next input character is a do
2. push a
3. while next input character is b do
4. pop a

Construct PDA for $L = \{a^n b^n\}$

Solution (continued)

- Transition $(i, s_1 \to s_2)$ means that when you see input character i, replace s_1 with s_2 as the top of stack.



Construct PDA for $L = \{a^n b^n\}$

Solution (continued)

- PDA P is specified as
  Set of states is $Q = \{q_0, q_1, q_2, q_3\}$
  Set of input symbols is $\Sigma = \{a, b\}$
  Set of stack symbols is $\Gamma = \{a, \$\}$
  Start state is q_0
  Set of accept states is $F = \{q_0, q_3\}$
  Transition function δ is: (Empty cell is ϕ)

Input	a			b			ϵ		
Stack	a	$\$$	ϵ	a	$\$$	ϵ	a	$\$$	ϵ
q_0									$\{(q_1, \$)\}$
q_1			$\{(q_1, a)\}$	$\{(q_2, \epsilon)\}$					
q_2				$\{(q_2, \epsilon)\}$				$\{(q_3, \epsilon)\}$	
q_3									

Construct PDA for $L = \{a^n b^n\}$

Solution (continued)

Step	State	Stack	Input	Action
1	q_0		$aaabbb$	push $\$$
2	q_1	$\$$	$aaabbb$	push a
3	q_1	$\$a$	$aabbb$	push a
4	q_1	$\$aa$	$abbb$	push a
5	q_1	$\$aaa$	bbb	pop a
6	q_2	$\$aa$	bb	pop a
7	q_2	$\$a$	b	pop a
8	q_2	$\$$		pop $\$$
9	q_3			accept

Step	State	Stack	Input	Action
1	q_0		$aababb$	push $\$$
2	q_1	$\$$	$aababb$	push a
3	q_1	$\$a$	$ababb$	push a
4	q_1	$\$aa$	$babb$	pop a
5	q_2	$\$a$	abb	crash
6	q_ϕ	$\$a$	bb	
7	q_ϕ	$\$a$	b	
8	q_ϕ	$\$a$		reject

Construct PDA for $L = \{ww^R \mid w \in \{a, b\}^*\}$

Problem

- Construct a PDA that accepts all strings from the language $L = \{ww^R \mid w \in \{a, b\}^*\}$

Construct PDA for $L = \{ww^R \mid w \in \{a,b\}^*\}$

Problem

- Construct a PDA that accepts all strings from the language $L = \{ww^R \mid w \in \{a,b\}^*\}$

Solution

PDA()

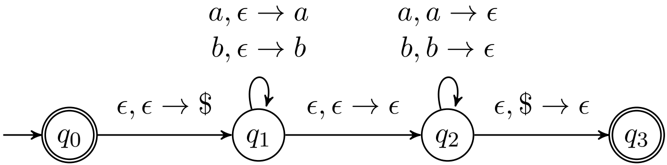
1. while next input character is a or b do
2. push the symbol
3. Nondeterministically guess the mid point of the string
4. while next input character is a or b do
5. pop the symbol

Construct PDA for $L = \{ww^R \mid w \in \{a, b\}^*\}$

Problem

- Construct a PDA that accepts all strings from the language $L = \{ww^R \mid w \in \{a, b\}^*\}$

Solution (continued)



Construct PDA for $L = \{a^i b^j c^k \mid i = j$ or $i = k\}$

Problem

- Construct a PDA that accepts all strings from the language $L = \{a^i b^j c^k \mid i = j \text{ or } i = k\}$

Construct PDA for $L = \{a^i b^j c^k \mid i = j$ or $i = k\}$

Problem

- Construct a PDA that accepts all strings from the language $L = \{a^i b^j c^k \mid i = j \text{ or } i = k\}$

Solution

$\mathrm{PDA}()$

1. while next input character is a do push a
2. Nondeterministically guess whether a's $=$ b's or a's $=$ c's

Case 1. a's $=$ b's.

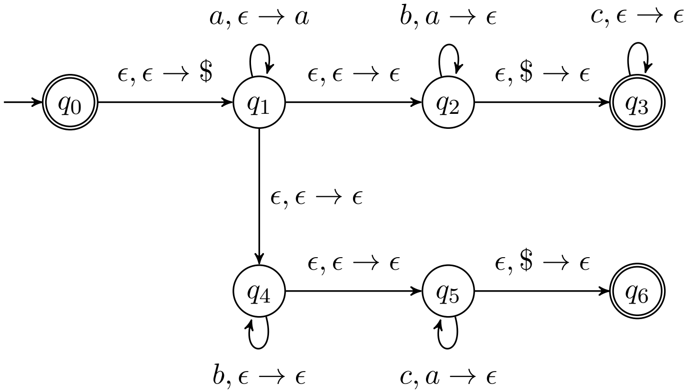
1. while next input character is b do pop a
2. while next input character is c do nothing

Case 2. a's $=$ c's.

1. while next input character is b do nothing
2. while next input character is c do pop a

Construct PDA for $L = \{a^i b^j c^k \mid i = j$ or $i = k\}$

Solution (continued)



Non-Context-Free Languages

Pumping lemma for context-free languages

Theorem

Suppose L is a context-free language over alphabet Σ. Then there is a natural number s so that for every long string $w \in L$ satisfying $|w| \geq s$, the string w can be split into five strings $w = uvxyz$ such that the following three conditions are true.

- $|vxy| \leq s$.
- $|vy| \geq 1$.
- For every $i \geq 0$, the string uv^ixy^iz also belongs to L.

$L = \{a^n b^n c^n\}$ is a non-CFL

Problem

- Prove that $L = \{a^n b^n c^n\}$ is not CFL.

$L = \{a^n b^n c^n\}$ is a non-CFL

Problem

- Prove that $L = \{a^n b^n c^n\}$ is not CFL.

Solution

- Suppose L is CFL. Then it must satisfy pumping property.
- Suppose $w = a^s b^s c^s$.
- Let $w = uvxyz$ where $|vxy| \leq s$ and $|vy| \geq 1$.
- Then $uv^i xy^i z$ must belong to L for all $i \geq 0$.
- We will show that $uxz \notin L$ for all possible cases.
- Three cases:

  Case 1. vxy consists of exactly 1 symbol (a's or b's or c's).

  Case 2. vxy consist of exactly 2 symbols (ab's or bc's).

  Case 3. vxy consist of exactly 3 symbols (abc's).

  This case is impossible. Why?

$L = \{a^n b^n c^n\}$ is a non-CFL

Solution (continued)

Case 1. vxy consists of exactly 1 symbol (a's or b's or c's).

Three subcases:

- Subcase i. vxy consists only of a's.
  Let $w = uvxyz = a^s b^s c^s$.
  uxz is not in L.
  Reason: $uxz = a^{s-(|v|+|y|)} b^s c^s \notin L$ as $(|v| + |y|) > 0$.
  uxz has fewer a's than b's or c's.
- Subcase ii. vxy consists only of b's.
  Similar to Subcase i.
- Subcase iii. vxy consists only of c's.
  Similar to Subcase i.

$L = \{a^n b^n c^n\}$ is a non-CFL

Solution (continued)

Case 2. vxy consist of exactly 2 symbols (ab's or bc's).

Two subcases:

- Subcase i. vxy consist only of a's and b's.
  Let $w = uvxyz = a^s b^s c^s$.
  uxz is not in L.
  Reason: $uxz = a^{k_1} b^{k_2} c^s \notin L$
  where $k_1 + k_2 = 2s - (|v| + |y|) < 2s$ as $(|v| + |y|) > 0$.
  uxz has either fewer a's or fewer b's than c's.
- Subcase ii. vxy consist only of b's and c's.
  Similar to Subcase i.

$L = \{ww \mid w \in \{a,b\}^*\}$ is a non-CFL

Problem

- Prove that $L = \{ww \mid w \in \{a,b\}^*\}$ is not CFL.

$L = \{ww \mid w \in \{a,b\}^*\}$ is a non-CFL

Problem

- Prove that $L = \{ww \mid w \in \{a,b\}^*\}$ is not CFL.

Solution

- Suppose L is CFL. Then it must satisfy pumping property.
- Suppose $w = a^s b^s a^s b^s$.
- Let $w = uvxyz$ where $|vxy| \leq s$ and $|vy| \geq 1$.
- Then uv^ixy^iz must belong to L for all $i \geq 0$.
- We will show that $uxz \notin L$ for all possible cases.
- Two cases:
  Case 1. vxy consists of exactly 1 symbol (a's or b's).
  Case 2. vxy consist of exactly 2 symbols (ab's or ba's).

$L = \{ww \mid w \in \{a,b\}^*\}$ **is a non-CFL**

Solution (continued)

Case 1. vxy consists of exactly 1 symbol (a's or b's).

Three subcases:

- Subcase i. vxy consists only of a's.
  Let $w = uvxyz = a^s b^s a^s b^s$.
  uxz is not in L.
  Reason: $uxz = a^{s-(|v|+|y|)} b^s a^s b^s \notin L$ as $(|v| + |y|) > 0$.
  uxz has fewer a's than b's.
- Subcase ii. vxy consists only of b's.
  Similar to Subcase i.

$L = \{ww \mid w \in \{a,b\}^*\}$ is a non-CFL

Solution (continued)

Case 2. vxy consist of exactly 2 symbols (ab's or ba's).

Two subcases:

- Subcase i. vxy consist only of a's and b's.
  Let $w = uvxyz = a^s b^s a^s b^s$.
  uxz is not in L.
  Reason: $uxz = a^{k_1} b^{k_2} a^s b^s \notin L$
  where $k_1 + k_2 = 2s - (|v| + |y|) < 2s$ as $(|v| + |y|) > 0$.
  uxz is not in the form of ww.
- Subcase ii. vxy consist only of b's and a's.
  Similar to Subcase i.

$L = \{a^n \mid n$ is a square$\}$ is a non-CFL

Problem

- Prove that $L = \{a^n \mid n \text{ is a square}\}$ is not CFL.

$L = \{a^n \mid n$ is a square$\}$ is a non-CFL

Problem

- Prove that $L = \{a^n \mid n \text{ is a square}\}$ is not CFL.

Solution

- Suppose L is CFL. Then it must satisfy pumping property.
- Suppose $w = a^{s^2}$.
- Let $w = uvxyz$ where $|vxy| \leq s$ and $|vy| \geq 1$.
- Then uv^ixy^iz must belong to L for all $i \geq 0$.
- But, $uv^2xy^2z \notin L$.
  Reason: Let $|vy| = k$. Then, $k \in [1, s]$.
  $uv^2xy^2z = a^{s^2+|vy|} = a^{s^2+k} \notin L$.
  Because, $s^2 < s^2 + k < (s+1)^2$ as $k \in [1, s]$.
- Contradiction! Hence, L is not CFL.

$L = \{a^n \mid n$ is a power of $2\}$ is a non-CFL

Problem

- Prove that $L = \{a^n \mid n$ is a power of $2\}$ is not CFL.

$L = \{a^n \mid n$ is a power of $2\}$ is a non-CFL

Problem

- Prove that $L = \{a^n \mid n$ is a power of $2\}$ is not CFL.

Solution

- Suppose L is CFL. Then it must satisfy pumping property.
- Suppose $w = a^{2^s}$, where s is the pumping length.
- Let $w = uvxyz$ where $|vxy| \leq s$ and $|vy| \geq 1$.
- Then uv^ixy^iz must belong to L for all $i \geq 0$.
- But, $uv^2xy^2z \notin L$.
  Reason: Let $|vy| = k$, where $k \in [1, s]$.
  Then, $uv^2xy^2z = a^{2^s+k} \notin L$.
  Because, $2^s < 2^s + k < 2^{s+1}$.
- Contradiction! Hence, L is not CFL.

$L = \{a^n \mid n$ is prime$\}$ is a non-CFL

Problem

- Prove that $L = \{a^n \mid n \text{ is prime}\}$ is not CFL.

$L = \{a^n \mid n$ is prime$\}$ is a non-CFL

Problem

- Prove that $L = \{a^n \mid n \text{ is prime}\}$ is not CFL.

Solution

- Suppose L is CFL. Then it must satisfy pumping property.
- Suppose $w = a^m$, where m is prime and $m \geq s$.
- Let $w = uvxyz$ where $|vxy| \leq s$ and $|vy| \geq 1$.
- Then uv^ixy^iz must belong to L for all $i \geq 0$.
- But, $uv^{m+1}xy^{m+1}z \notin L$.
  Reason: Let $|vy| = k$. Then, $k \in [1, s]$.
  $uv^{m+1}xy^{m+1}z = a^{m+m|vy|} = a^{m+mk} = a^{m(k+1)} \notin L$.
- Contradiction! Hence, L is not CFL.

Membership problem: A decision problem on CFL's

Problem

- Given a CFG G and a string w, is $w \in L(G)$?

Membership problem: A decision problem on CFL's

Problem

- Given a CFG G and a string w, is $w \in L(G)$?

Solution

- This is a difficult problem. Why?
  Nondeterminism cannot be eliminated unlike in finite automata.
- Algorithmically solvable.
  CYK algorithm (for grammars in CNF)
  Earley parser
  GLR parser

More decision problems involving CFL's

Decision problems

Algorithmically solvable.

- Given a CFG G, is $L(G)$ nonempty?
- Given a CFG G, is $L(G)$ infinite?
- Given a CFG G, is G a regular grammar?
- Given a CFG G, is $L(G)$ a regular language?

Algorithmically unsolvable.

- Given a CFG G, is $L(G) = \Sigma^*$?
- Given a CFG G, is G ambiguous?
- Given a CFG G, is $L(G)$ inherently ambiguous?
- Given two CFG's G_1 and G_2, is $L(G_1) = L(G_2)$?
- Given two CFG's G_1 and G_2, is $L(G_1) \subseteq L(G_2)$?
- Given two CFG's G_1 and G_2, is $L(G_1) \cap L(G_2)$ nonempty?