Es

$\Sigma = \{a, b, c\}$

$L = \{ a^i b^j c^{i+j} \mid i, j \geq 1 \}$ → not regular

variable ≠ symbol !

$S \to aSc \mid aBc$ → Embedding wrapping

$B \to bc \mid bBc$

$\underbrace{a \ldots a \underbrace{b \ldots b c}\ldots c}$

It's impossible to connect something in this way : (in CFG)

$a \ldots a b \ldots b c \ldots c$

Crossing

Ambiguous CFGs

In the CFG

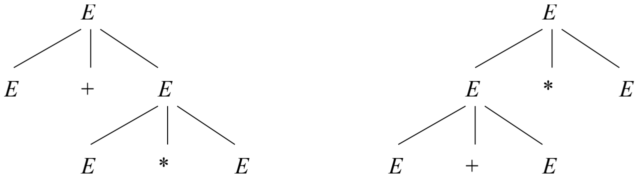
1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$
5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow I\ a$
8. $I \rightarrow I\ b$
9. $I \rightarrow I\ 0$
10. $I \rightarrow I\ 1$

the sentential form $E + E * E$ has two derivations

$$E \Rightarrow E + E \Rightarrow E + E * E$$
$$E \Rightarrow E * E \Rightarrow E + E * E$$

Ambiguous CFGs

Associated parse trees for the derivations of $E + E * E$



(a) (b)

The two derivations correspond to different **precedences** for operators sum and multiplication

Ambiguous CFGs

The existence of different derivations for a string is not problematic, if these correspond to a single parse tree

Example : In our CFG for arithmetic expressions, the string $a + b$ has at least two derivations

$$E \Rightarrow E + E \Rightarrow I + E \Rightarrow a + E \Rightarrow a + I \Rightarrow a + b$$
$$E \Rightarrow E + E \Rightarrow E + I \Rightarrow I + I \Rightarrow I + b \Rightarrow a + b$$

This is NOT ambigous

These two have the same parse-tree

However, the associated parse trees are the same, and string $a + b$ is **not** ambiguous

Ambiguous CFGs

Let $G = (V, T, P, S)$ be a CFG. G is **ambiguous** if there exists a string in $L(G)$ with more than one parse tree

If every string in $L(G)$ has only one parse tree, G is said to be **unambiguous**

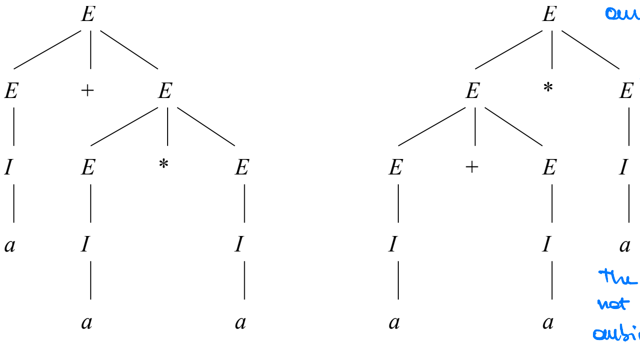
The ambiguity is **problematic** in many applications where the syntactic structure of a sentence is used to interpret its meaning

Example: compilers for programming languages

Example

In the CFG for arithmetic expressions, the terminal string $a + a * a$ has two parse trees

also $a+a+a$ is ambigous



(a) (b)

This doesn't mean that the algebric result is ambigous

the fact that it's not algebraically ambigous, doesn't mean that it's not CFG ambigous

$a+a+a$ is not algebraically ambigous ONLY thanks to the properties of $+$

Canonical derivations

A parse tree is associated with a **unique** leftmost derivation

A leftmost derivation is associated with a **unique** parse tree

More than one leftmost derivations always imply more than one parse trees

Similary for rightmost derivations

Inherent ambiguity

A CFL L is **inherently ambiguous** when every CFG such that $L(G) = L$ is ambiguous

Example : Let us consider the language

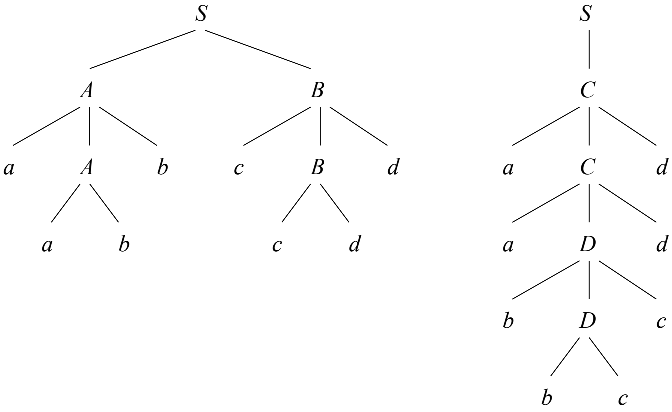
$$L = \{a^n b^n c^m d^m \mid n \geqslant 1,\ m \geqslant 1\} \cup \{a^n b^m c^m d^n \mid n \geqslant 1,\ m \geqslant 1\}$$

L can be generated by a CFG with the following productions

$$\begin{aligned} S &\rightarrow AB \mid C \\ A &\rightarrow aAb \mid ab \\ B &\rightarrow cBd \mid cd \\ C &\rightarrow aCd \mid aDd \\ D &\rightarrow bDc \mid bc \end{aligned}$$

Inherent ambiguity

There are two parse trees for the string *aabbccdd*



(a) (b)

Inherent ambiguity

Associated leftmost derivations

$$S \underset{lm}{\Rightarrow} AB \underset{lm}{\Rightarrow} aAbB \underset{lm}{\Rightarrow} aabbB \underset{lm}{\Rightarrow} aabbcBd \underset{lm}{\Rightarrow} aabbccdd$$

$$S \underset{lm}{\Rightarrow} C \underset{lm}{\Rightarrow} aCd \underset{lm}{\Rightarrow} aaDdd \underset{lm}{\Rightarrow} aabDcdd \underset{lm}{\Rightarrow} aabbccdd$$

It is possible to show that **every** CFG generating L provides a similar ambiguity for the string $aabbccdd$ (not in the textbook)

Language L is therefore inherently ambiguous

Exercises

- Provide an example showing that the CFG with productions

$$S \rightarrow aS \mid aSbS \mid \epsilon$$

  is ambiguous
- Provide an example showing that the CFG with productions

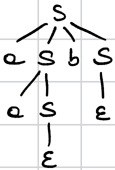
$$S \rightarrow aSbS \mid bSaS \mid \epsilon$$

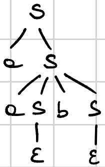
  is ambiguous. **Hint**: consider some string of length 4

1) Consider string aab:

$$S \Rightarrow aSbS \Rightarrow aaSbS \Rightarrow aa\varepsilon bS \Rightarrow aa\varepsilon b\varepsilon$$

$$S \Rightarrow aS \Rightarrow aaSbS \Rightarrow aa\varepsilon bS \Rightarrow aa\varepsilon b\varepsilon$$



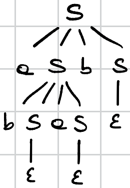


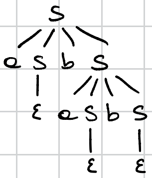
There are two trees ⇒ ambiguity

2) Consider the string abab

$$S \Rightarrow aSbS \Rightarrow abSaSbS \Rightarrow ab\varepsilon aSbS \Rightarrow ab\varepsilon a\varepsilon bS \Rightarrow ab\varepsilon a\varepsilon b\varepsilon$$

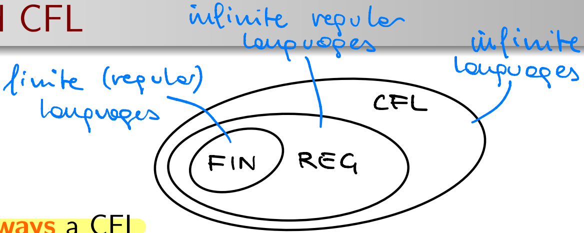
$$S \Rightarrow aSbS \Rightarrow aSbaSbS \Rightarrow a\varepsilon baSbS \Rightarrow a\varepsilon ba\varepsilon bS \Rightarrow a\varepsilon ba\varepsilon b\varepsilon$$





There are two trees $\Rightarrow$ ambiguity

Reguar languages and CFL



A regular language is **always** a CFL

From a regular expression or from an FA we can aways construct a CFG generating the same language

This is not in the textbook!

From regular expression to CFG

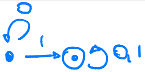
Let E be any regular expression. We use a variable for E (start symbol) and a variable for each subexpression of E

We use **structural induction** on the regular expression to build the productions of our CFG

- if $E = \mathbf{a}$, then add production $E \rightarrow a$
- if $E = \epsilon$, then add production $E \rightarrow \epsilon$
- if $E = \emptyset$, then production set is empty
- if $E = F + G$, then add production $E \rightarrow F \mid G$
- if $E = FG$, then add production $E \rightarrow FG$
- if $E = F^*$, then add production $E \rightarrow FE \mid \epsilon$

Example

Regular expression : $\mathbf{0}^*\mathbf{1}(\mathbf{0}+\mathbf{1})^*$ $\Rightarrow$



$\Rightarrow$

$$Q_0 \to 0Q_0 \mid 1Q_1$$
$$Q_1 \to 0Q_1 \mid 1Q_1 \mid \varepsilon$$

Use left-associativity for concatenation

CFG :

0^* $\quad$ $1(0+1)^*$

$$E \to AR$$
$$R \to BC \qquad 1 \qquad (0+1)^*$$
$$0^*$$
$$A \to 0A \mid \epsilon \to 0^0$$
$$B \to 1 \qquad (0+1)^+$$
$$C \to DC \mid \epsilon \to (0+1)^0$$
$$D \to 0 \mid 1$$
$$\to 0+1$$

From FA to CFG

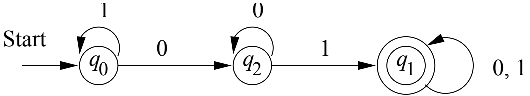
We use a variable Q for each state q of the FA. Initial symbol is Q_0

For each transition from state p to state q under symbol a, add production $P \rightarrow a\ Q$

If q is a final state, add production $Q \rightarrow \epsilon$

Example

Automaton :



CFG :

$$Q_0 \to 1Q_0 \mid 0Q_2$$
$$Q_2 \to 0Q_2 \mid 1Q_1$$
$$Q_1 \to 0Q_1 \mid 1Q_1 \mid \epsilon$$

String 1101 is accepted by the automaton. In the equivalent CFG, 1101 has the following derivation :

$$Q_0 \Rightarrow 1Q_0 \Rightarrow 11Q_0 \Rightarrow 110Q_2 \Rightarrow 1101Q_1 \Rightarrow 1101$$

Automata, Languages and Computation

Chapter 6 : Push-Down Automata

Master Degree in Computer Engineering
University of Padua
Lecturer : Giorgio Satta

Lecture based on material originally developed by :
Gösta Grahne, Concordia University

Introduction

A push-down automaton consists of

- an ϵ-NFA
- a **stack** representing the auxiliary memory

The stack can

- record an arbitrary number of symbols
- release symbols with a strict policy :
  last in, **first out**

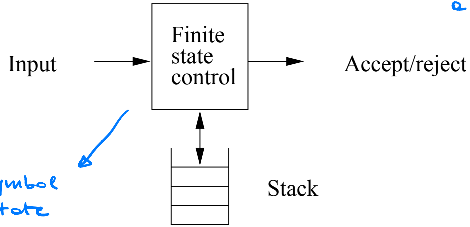
Push-down automata and context-free grammars are equivalent formalisms

Introduction

A transition of a push-down automaton

- consumes a single symbol from the input, or else is an ϵ-transition
- updates the current state
- replaces the **top-most** symbol of the stack stack with a string of symbols, including ϵ

each symbol has a different "push"



3 inputs: - symbol
- state
- stack

Introduction

More precisely, replacement of symbol X in the stack top-most position with string γ amounts to

special case of switch

- removing X if $\gamma = \epsilon$, also called **pop**
- replacing X if $\gamma = Y$, also called **switch**; if $\gamma = X$, the stack remains unaltered
- inserting new symbols if $|\gamma| > 1$; if $\gamma = ZX$ the transition is called **push**

I always read X first

First symbol of γ becomes top symbol of the new stack

Example

Let us consider the language (palindrome strings with even length)

$$L_{wwr} = \{ww^R \mid w \in \{0,1\}^*\}$$

generated by the CFG productions

$$P \to 0P0, \;\; P \to 1P1, \;\; P \to \epsilon$$

Example

A push-down automaton for L_{wwr} has three states, and operates as follows

Guess that you are reading w. Stay in state q_0, and push the input symbol onto the stack

Guess that you are at the boundary between w and w^R. Go to state q_1 using an ϵ-transition

You are now reading the first symbol of w^R. Compare it to the top of the stack. If they match, pop the stack and remain in state q_1. If they don't match, the automaton **halts**, i.e., it does not have a next move

If the stack is empty, go to state q_2 and **accept**