

# Finite Automata and Formal Languages

TMV026/DIT321 – LP4 2011

Lecture 11

May 3rd 2011

Overview of today's lecture:

- Abstract Syntax
- Ambiguity in Grammars
- Chomsky Hierarchy

---

CFG – Abstract Syntax – Ambiguous Grammars – Chomsky Hierarchy

---

## Parse Trees and Abstract Syntax

*Concrete syntax* describes the way documents are written while *abstract syntax* describes the pure structure of a document.

The abstract syntax of some data is its structure described as a data type.

A parse tree also describe the structure of data.

Parse trees are similar to abstract syntax trees but they also contain features such as parentheses which are syntactically significant but also implicit in the structure of the abstract syntax tree.

## Example: Abstract Syntax of Simple Expressions

Given the grammar

$$E \rightarrow 0 \mid 1 \mid E + E \mid \text{if } B \text{ then } E \text{ else } E$$

$$B \rightarrow \text{True} \mid \text{False} \mid E < E \mid E == E$$

its abstract syntax can be defined by the following data types:

```
data Exp = Z | 0 | Plus Exp Exp | IfThenElse BExp Exp Exp
```

```
data BExp = T | F | Less Exp Exp | Eq Exp Exp
```

```
bexp = Less Z (Plus 0 (Plus Z Z))
```

## Ambiguous Grammars

**Example:** Consider the following grammar

$$E \rightarrow E + E \mid E * E \mid 0 \mid 1$$

The sentential form  $E + E * E$  has the following 2 possible derivations

1.  $E \Rightarrow E + E \Rightarrow E + E * E$
2.  $E \Rightarrow E * E \Rightarrow E + E * E$

Observe the difference of the corresponding parse tree for each derivation.

Intuitively, there are 2 possible meanings for the words.

What would be the result of  $1 + 1 * 0$  in each case?

1.  $1 + (1 * 0) = 1$
2.  $(1 + 1) * 0 = 0$

## Ambiguous Grammars

**Definition:** A CFG grammar  $G = (V, T, \mathcal{R}, S)$  is *ambiguous* if there is at least a string  $w \in T^*$  for which we can find two (or more) parse trees, each with root  $S$  and yield  $w$ .

If each string has at most one parse tree we say that the grammar is *unambiguous*.

**Note:** The existence of different derivations for a certain string does not necessarily mean the existence of different parse trees.

$$1. E \Rightarrow E + E \Rightarrow 1 + E \Rightarrow 1 + 0$$

$$2. E \Rightarrow E + E \Rightarrow E + 0 \Rightarrow 1 + 0$$

## Example: Ambiguous Grammar

The following (simplified part of a) grammar produces ambiguity in programming languages with conditionals:

$$C \rightarrow \text{if } b \text{ then } C \text{ else } C$$

$$C \rightarrow \text{if } b \text{ then } C$$

$$C \rightarrow s$$

The expression “if  $b$  then if  $b$  then  $s$  else  $s$ ” can be interpreted in the following 2 different ways:

$$1. \text{if } b \text{ then (if } b \text{ then } s \text{ else } s)$$

$$2. \text{if } b \text{ then (if } b \text{ then } s) \text{ else } s$$

How should the parser of this language understand the expression?

## Removing Ambiguity from Grammars

Unfortunately, there is no algorithm that can tell us if a grammar is ambiguous.

In addition, there is no algorithm that can remove ambiguity in a grammar.

Some context-free languages have *only* ambiguous grammars. These languages are called *inherently ambiguous*.

In these cases removal of ambiguity is impossible.

For the other cases, there are well-known techniques for eliminating ambiguity.

## Problems with the Grammar of Expressions (Slide 3)

**Observe:** There are 2 causes of ambiguity in the following grammar

$$E \rightarrow E + E \mid E * E \mid 0 \mid 1$$

1. The precedence of the operators was not taken into account.  
\* has stronger precedence than + but this is not reflected in the grammar.
2. A sequence of identical operator can be grouped either from the right or from the left.

We will have 2 parse trees for  $E + E + E$ .

Even if the operator is associative in the language we define, we need to pick one way of grouping the operator.

## Solution for the Grammar of Expressions (Slide 3)

To enforce precedence we introduce different variables representing those expressions with the same “binding strength”. Namely:

- A *factor* is an expression that cannot be broken apart by any adjacent operators: either 0 or 1, or a parenthesised expression.
- A *term* is an expression that cannot be broken by the + operator, that is a sequence of one or more factors.
- An *expression* is a sequence of terms connected by +.

Terms and expressions will associate to the left.

## Unambiguous Grammar for Expressions

We have then the following grammar:

$$F \rightarrow (E) \mid 0 \mid 1$$

$$T \rightarrow F \mid T * F$$

$$E \rightarrow T \mid E + T$$

We have now either  $E \Rightarrow^* 1 + 1 * 0$  with the usual meaning or  $E \Rightarrow^* (1 + 1) * 0$  if we want to change the precedence of the operators.

Even  $E \Rightarrow^* 1 + 0 + 1$  has now only one derivation.

**Note:** It is not obvious that this is an unambiguous grammar!

## Leftmost/Rightmost Derivations and Ambiguity

We have seen that derivations might not be unique even if the grammar is unambiguous.

However, in an unambiguous grammars both the leftmost and the rightmost derivations will be unique.

**Example:** The grammar of slide 3 must be ambiguous since we have 2 leftmost derivations for  $1 + 0 * 1$ :

1.  $E \xRightarrow{lm} E + E \xRightarrow{lm} 1 + E \xRightarrow{lm} 1 + E * E \xRightarrow{lm} 1 + 0 * E \xRightarrow{lm} 1 + 0 * 1$
2.  $E \xRightarrow{lm} E * E \xRightarrow{lm} E + E * E \xRightarrow{lm} 1 + E * E \xRightarrow{lm} 1 + 0 * E \xRightarrow{lm} 1 + 0 * 1$

**Note:** In general we have

Number of leftmost derivations = number of rightmost derivations =  
number of parse trees.

## Leftmost/Rightmost Derivations and Ambiguity

**Theorem:** Let  $G = (V, T, \mathcal{R}, S)$  be a CFG and let  $w \in T^*$ .  $w$  has 2 distinct parse trees iff  $w$  has 2 distinct leftmost (rightmost) derivations from  $S$ .

**Proof:** We sketch the proof dealing with leftmost derivations.

**If)** Start the tree with  $S$ . Examine each step in the derivation. Only the leftmost variable will be replaced. This variable corresponds to the leftmost node in the tree being constructed. The production used determines the children of this subtree. 2 different derivations will produce a subtree with different children.

**Only-if)** In Lecture 10 slides 26–27 we constructed a leftmost derivation form a parse tree. Observe that if the trees have a node where different productions are used then so will the leftmost derivations.

## Example: The Polish Notation

Consider the following grammar for arithmetical expressions:

$$E \rightarrow * E E \mid + E E \mid a \mid b$$

**Theorem:** *This grammar is not ambiguous.*

**Proof:** By induction on  $|w|$  we prove the following lemma:

**Lemma:** *For any  $k$ , there is at most one leftmost derivation of  $E^k \xRightarrow{lm}^* w$ .*

It follows from this result that we have the following property:

**Corollary:** *If  $*u_1u_2 = *v_1v_2 \in \mathcal{L}(E)$  then  $u_1 = v_1$  and  $u_2 = v_2$ . Similarly if  $+u_1u_2 = +v_1v_2 \in \mathcal{L}(E)$  then  $u_1 = v_1$  and  $u_2 = v_2$ .*

In addition, the result also says that if  $w \in \mathcal{L}(E)$  then there is a unique parse tree for  $w$ .

## Example: The Polish Notation

**Proof:** (Of the lemma) By induction on  $|w|$ . Let  $i$  be either  $a$  or  $b$ .

**Basis case:** If  $|w| = 1$  then  $w = i$ . Then  $k$  must be 1 and  $E \xRightarrow{lm}^* i$ .

**Inductive step:** If  $|w| = n + 1$  with  $n > 0$  then we have 3 cases:

1.  $w = *v$ : The derivation must be of the form  $EE^k \xRightarrow{lm}^* *EE^k \xRightarrow{lm}^* *v$ .

Then  $E^{k+2} \xRightarrow{lm}^* v$  with  $|v| = n$ .

By IH we know this derivation is unique and so must be that of  $w$ .

2.  $w = +v$ : The derivation must be of the form  $EE^k \xRightarrow{lm}^* +EE^k \xRightarrow{lm}^* +v$ .

Similarly as above.

3.  $w = iv$ : The derivation must be of the form  $EE^k \xRightarrow{lm}^* iE^k \xRightarrow{lm}^* iv$ .

Then  $E^k \xRightarrow{lm}^* v$  and we conclude by IH as before.

## Example: Balanced Parentheses

The following grammar of parenthesis expressions is ambiguous

$$E \rightarrow \epsilon \mid EE \mid (E)$$

Let us consider the following grammar instead:

$$S \rightarrow (S)S \mid \epsilon$$

We have:

**Lemma:**  $\mathcal{L}(S) = \mathcal{L}(E)$ .

**Theorem:** *The grammar for  $S$  is not ambiguous.*

## Example: Balanced Parentheses (Cont.)

**Lemma:**  $\mathcal{L}(S)\mathcal{L}(S) \subseteq \mathcal{L}(S)$ .

**Proof:** By induction on  $|w|$  we prove that if  $w \in \mathcal{L}(S)$  then  $w\mathcal{L}(S) \subseteq \mathcal{L}(S)$ .

**Basis case:** If  $|w| = 0$  then  $\epsilon\mathcal{L}(S) = \mathcal{L}(S)$ .

**Inductive step:** If  $|w| = n + 1$  then  $w = (u)v$  with  $u, v \in \mathcal{L}(S)$  and  $|u|, |v| \leq n$ .

By IH we have that  $v\mathcal{L}(S) \subseteq \mathcal{L}(S)$ .

Now

$$w\mathcal{L}(S) = (u)v\mathcal{L}(S) \subseteq (u)\mathcal{L}(S) \subseteq \mathcal{L}(S)$$

since  $u \in \mathcal{L}(S)$  and  $(\mathcal{L}(S))\mathcal{L}(S) \subseteq \mathcal{L}(S)$  because  $S \rightarrow (S)S$  is a production.



**Example: Balanced Parentheses (Cont.)**

**Lemma:**  $\mathcal{L}(S) = \mathcal{L}(E)$ .

**Proof:** Let  $w \in \mathcal{L}(S)$  and  $x \in \mathcal{L}(E)$ .

$\mathcal{L}(S) \subseteq \mathcal{L}(E)$ : By induction on  $|w|$ . Basis case is trivial.

Otherwise  $w = (u)v$  with  $u, v \in \mathcal{L}(S)$  and  $|u|, |v| \leq n$ .

By IH  $u, v \in \mathcal{L}(E)$ . Using the productions of  $E$  we conclude that  $w \in \mathcal{L}(E)$ .

$\mathcal{L}(E) \subseteq \mathcal{L}(S)$ : By induction on the length of  $E \Rightarrow^* x$ .

If  $E \Rightarrow \epsilon = x$  then  $x \in \mathcal{L}(S)$ .

If  $E \Rightarrow EE \Rightarrow^* yz = x$  then by IH  $y, z \in \mathcal{L}(S)$  and by previous lemma then  $x = yz \in \mathcal{L}(S)\mathcal{L}(S) \subseteq \mathcal{L}(S)$ .

If  $E \Rightarrow (E) \Rightarrow^* (y) = x$  then by IH  $y \in \mathcal{L}(S)$  and  $x = (y)\epsilon \in \mathcal{L}(S)$ .

**Example: Balanced Parentheses (Cont.)**

**Theorem:** *The grammar for  $S$  is not ambiguous.*

**Proof:** Not trivial. Let  $w \in \{(\,)\}^*$ .

One tries to show that there is at most one leftmost derivation  $S \xRightarrow{lm}^* w$ .

If  $w = \epsilon$  then it is trivial.

Otherwise,  $w = )v$  and there is no derivation or  $w = (v$  and we have that  $S \xRightarrow{lm} (S)S$ .

We now prove that

**Lemma:** *Given  $u$ , for any  $k$ , there is at most one leftmost derivation  $S(\ )S)^k \xRightarrow{lm}^* u$ .*

Now we use this lemma with  $v$  and  $k = 1$  to conclude that there is at most one leftmost derivation  $S)S \xRightarrow{lm}^* v$ .

Then, there is at most one leftmost derivation  $S \xRightarrow{lm} (S)S \xRightarrow{lm}^* (v = w$ .

## Example: Balanced Parentheses (Cont.)

**Lemma:** Given  $u$ , for any  $k$ , there is at most one leftmost derivation  $S\langle\rangle S^k \xRightarrow{lm}^* u$ .

**Proof:** By induction on  $|u|$ .

If  $|u| = 0$  then  $u = \epsilon$ . Here  $k$  must be 0 and we have  $S\langle\rangle S^0 = S \Rightarrow \epsilon$ .

If  $|u| = n + 1$  then  $u = (x \text{ or } u =)x$  with  $|x| = n$ . By IH there is at most one leftmost derivation  $S\langle\rangle S^k \xRightarrow{lm}^* x$  for any  $k$ . We have 2 cases:

- $u = (x$ : Then  $S\langle\rangle S^k \xRightarrow{lm} (S)S\langle\rangle S^k = (S\langle\rangle S)^{k+1} \xRightarrow{lm}^* (x$   
for a derivation  $S\langle\rangle S^{k+1} \xRightarrow{lm}^* x$  which, if it exists, it is unique.
- $u = )x$ : Then  $S\langle\rangle S^k \xRightarrow{lm} \epsilon\langle\rangle S^k = )S\langle\rangle S^{k-1} \xRightarrow{lm}^* )x$   
for a derivation  $S\langle\rangle S^{k-1} \xRightarrow{lm}^* x$  which, if it exists, it is unique.

## Inherent Ambiguity

**Definition:** A context-free language  $\mathcal{L}$  is said to be *inherently ambiguous* if *all* its grammars are ambiguous.

**Note:** It is enough that 1 grammar for the  $\mathcal{L}$  is unambiguous for  $\mathcal{L}$  to be unambiguous.

## Example: Inherent Ambiguity

The following language is inherently ambiguous:

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

$\mathcal{L}$  is context-free and generated by the following grammar:

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

Strings of the form  $a^n b^n c^n d^n$  for  $n > 0$  have 2 different leftmost derivations.

See pages 214–215 in the book for the intuition of why  $\mathcal{L}$  is inherent ambiguous. The proof is complex!

## Chomsky Hierarchy

This hierarchy of grammars was described by Noam Chomsky in 1956:

**Type 0:** Unrestricted grammars.

They generate exactly all languages that can be recognised by a Turing machine.

**Type 1:** Context-sensitive grammars.

Rules are of the form  $\alpha A \beta \rightarrow \alpha \gamma \beta$ .  $\alpha$  and  $\beta$  may be empty, but  $\gamma$  must be non-empty.

**Type 2:** Context-free grammars.

Are used to produce the syntax of most programming languages.

**Type 3:** Regular grammars.

Rules are of the form  $A \rightarrow Ba$ ,  $A \rightarrow aB$  or  $A \rightarrow \epsilon$ .

We have that  $\text{Type 3} \subset \text{Type 2} \subset \text{Type 1} \subset \text{Type 0}$ .

## Regular Languages and Context-Free Languages

**Theorem:** If  $\mathcal{L}$  is a regular language then  $\mathcal{L}$  is context-free.

**Proof:** If  $\mathcal{L}$  is a regular language then  $\mathcal{L} = \mathcal{L}(D)$  for a DFA  $D$ .

Let  $D = (Q, \Sigma, \delta, q_0, F)$ .

We define a CFG  $G = (Q, \Sigma, \mathcal{R}, q_0)$  where  $\mathcal{R}$  is the set of productions:

- $p \rightarrow aq$  if  $\delta(p, a) = q$
- $p \rightarrow \epsilon$  if  $p \in F$

We must prove by induction on  $|w|$  that  $p \Rightarrow^* wq$  iff  $\hat{\delta}(p, w) = q$  and  $p \Rightarrow^* \epsilon$  iff  $\hat{\delta}(p, w) \in F$ . Then, in particular  $w \in \mathcal{L}(G)$  iff  $w \in \mathcal{L}(D)$ .

**Note:** A grammar where all rules are of the form  $A \rightarrow aB$  or  $A \rightarrow \epsilon$  is called *left regular* (and *right regular* if all rules are of the form  $A \rightarrow Ba$  or  $A \rightarrow \epsilon$ ).

## Regular Languages and Context-Free Languages

- If  $|w| = 0$  then  $w = \epsilon$ .

Given the rules in the grammar,  $p \Rightarrow^* q$  only when  $p = q$  and  $p \Rightarrow^* \epsilon$  only when  $p \rightarrow \epsilon$ .

We have  $\hat{\delta}(p, \epsilon) = p$  by definition of  $\hat{\delta}$  and  $p \in F$  by the way we defined the grammar.

- Suppose  $|w| = n + 1$ , then  $w = av$ .  $\hat{\delta}(p, av) = \hat{\delta}(\delta(p, a), v)$  with  $|v| = n$ .

By IH  $\delta(p, a) \Rightarrow^* vq$  iff  $\hat{\delta}(\delta(p, a), v) = q$ .

By construction we have a rule  $p \rightarrow a\delta(p, a)$ .

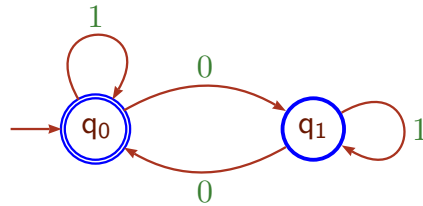
Then  $p \Rightarrow a\delta(p, a) \Rightarrow^* avq$  iff  $\hat{\delta}(p, av) = \hat{\delta}(\delta(p, a), v) = q$ .

By IH  $\delta(p, a) \Rightarrow^* v$  iff  $\hat{\delta}(\delta(p, a), v) \in F$ .

Now  $p \Rightarrow a\delta(p, a) \Rightarrow^* av$  iff  $\hat{\delta}(p, av) = \hat{\delta}(\delta(p, a), v) \in F$ .

### Example

A DFA that generates the language over  $\{0, 1\}$  with an even number of 0's is



The left regular grammar for this language is

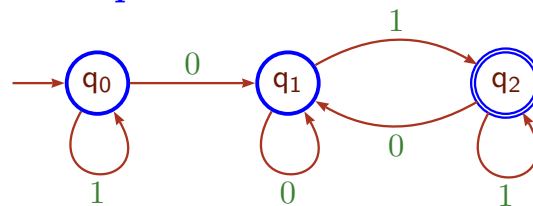
$$\begin{aligned}
 q_0 &\rightarrow \epsilon \mid 0q_1 \mid 1q_0 \\
 q_1 &\rightarrow 0q_0 \mid 1q_1
 \end{aligned}$$

with  $q_0$  as the start variable.

Are the strings 01011 and 01010 in the language?

### Example

Consider the following DFA over  $\{0, 1\}$ :



The left regular grammar for this language is

$$q_0 \rightarrow 0q_1 \mid 1q_0 \quad q_1 \rightarrow 0q_1 \mid 1q_2 \quad q_2 \rightarrow \epsilon \mid 0q_1 \mid 1q_2$$

with  $q_0$  as the start variable.

$$q_0 \Rightarrow 1q_0 \Rightarrow 10q_1 \Rightarrow 100q_1 \Rightarrow 1001q_2 \Rightarrow 10010q_1 \Rightarrow 100101q_2 \Rightarrow 1001011q_2 \Rightarrow 1001011$$

The right regular grammar for this language is

$$q_0 \rightarrow \epsilon \mid q_01 \quad q_1 \rightarrow q_00 \mid q_10 \mid q_20 \quad q_2 \rightarrow q_11 \mid q_21$$

with  $q_2$  as the start variable.

$$q_2 \Rightarrow q_21 \Rightarrow q_111 \Rightarrow q_2011 \Rightarrow q_11011 \Rightarrow q_101011 \Rightarrow q_0001011 \Rightarrow q_01001011 \Rightarrow 1001011$$