Finite Automata and Formal Languages

TMV026/DIT321– LP4 2012

Lecture 12
Ana Bove

May 3rd 2012

Overview of today's lecture:

- Abstract Syntax
- Ambiguity in Grammars
- Chomsky Hierarchy

Example

Construct a grammar for the following language:

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

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

Concrete 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 the data but they may contain features such as parentheses which are syntactically significant but that are implicit in the structure of the abstract syntax tree.

Example: Abstract Syntax of Simple Expressions

Given the grammar

$$\begin{array}{lcl} E & \rightarrow & 0 \mid 1 \mid E + E \mid E * E \mid \text{if } B \text{ then } E \text{ else } E \mid (E) \\ B & \rightarrow & \text{True} \mid \text{False} \mid E < E \mid E == E \end{array}$$

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

```
data Exp = Z | O | Plus Exp Exp | Mult Exp Exp |
           IfThenElse BExp Exp Exp

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

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

Ambiguous Grammars

Example: Consider the following grammar

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

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:

$$\begin{aligned} C &\rightarrow \text{if } b \text{ then } C \text{ else } C \\ C &\rightarrow \text{if } b \text{ then } C \\ C &\rightarrow s \end{aligned}$$

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

1. if b then (if b then s else s)
2. if b then (if b then s) 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

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

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 connected by $*$;
- An *expression* is a sequence of one or more of terms connected by $+$.

In addition, terms and expressions will associate to the left.

Unambiguous Grammar for Expressions

We have then the following grammar:

$$\begin{array}{lcl} F & \rightarrow & 0 \mid 1 \mid (E) \\ T & \rightarrow & F \mid T * F \\ E & \rightarrow & T \mid E + T \end{array}$$

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

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 11 slides 27–28 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 $E^k \overset{lm}{\Rightarrow}{}^* w$.*

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

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

Proof: By induction on $|w|$. Let i be either a or b.

Base case: If $|w| = 1$ then $w = i$. Then k must be 1 and $E \overset{lm}{\Rightarrow} i$.

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

1. $w = *v$: The derivation must be of the form $EE^k \overset{lm}{\Rightarrow} *EEE^k \overset{lm}{\Rightarrow}{}^* *v$.
   Then $E^{k+2} \overset{lm}{\Rightarrow}{}^* v$ 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 $EE^k \overset{lm}{\Rightarrow} +EEE^k \overset{lm}{\Rightarrow}{}^* +v$.
   Similarly as above.

3. $w = iv$: The derivation must be of the form $EE^k \overset{lm}{\Rightarrow} iE^k \overset{lm}{\Rightarrow}{}^* iv$.
   Then $E^k \overset{lm}{\Rightarrow}{}^* v$ and we conclude by IH as before.

Example: Balanced Parentheses

The following grammar of parenthesis expressions is ambiguous

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

Let us consider the following grammar instead:

$$S \to \epsilon \mid (S)S$$

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 $w\mathcal{L}(S) \subseteq \mathcal{L}(S)$.

Base 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 $|u|, |v| \leqslant n$.
By IH we have that $v\mathcal{L}(S) \subseteq \mathcal{L}(S)$.
Since $u \in \mathcal{L}(S)$ and $S \to (S)S$ is a production then $(\mathcal{L}(S))\mathcal{L}(S) \subseteq \mathcal{L}(S)$.
Now

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

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 $|w|$. Base case is trivial.
Let $|w| = n+1$. We have that $w = (u)v$ with $u, v \in \mathcal{L}(S)$ and $|u|, |v| \leqslant n$.
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 $E \Rightarrow^* w$.
If $E \Rightarrow \epsilon = w$ then $w \in \mathcal{L}(S)$.
If $E \Rightarrow EE \Rightarrow^* uv = w$ then by IH $u, v \in \mathcal{L}(S)$ and by previous lemma then $w = uv \in \mathcal{L}(S)\mathcal{L}(S) \subseteq \mathcal{L}(S)$.
If $E \Rightarrow (E) \Rightarrow^* (u) = w$ then by IH $u \in \mathcal{L}(S)$ and $w = (u)\epsilon \in \mathcal{L}(S)$.

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 $S \overset{lm}{\Rightarrow}^* w$.
If $w = \epsilon$ then it is trivial.
Otherwise, $w =)v$ and there is no derivation or $w = (v$ and we have that $S \overset{lm}{\Rightarrow} (S)S$.
We now should prove (by induction on $|u|$) that

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

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

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

Example: Balanced Parentheses (Cont.)

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

Proof: By induction on $|u|$.
If $|u| = 0$ then $u = \epsilon$. Here k must be 0 and we have $S\langle)S\rangle^0 = S \Rightarrow \epsilon$.
If $|u| = n + 1$ then $u = (x$ or $u =)x$ with $|x| = n$. By IH, for any k, there is at most one leftmost derivation $S\langle)S\rangle^k \overset{lm}{\Rightarrow}^* x$. We have 2 cases:

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

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

(See grammar for $\mathcal{L}$ in slide 1.)

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!

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:

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

We must prove by induction on $|w|$ that $p \Rightarrow^* wq$ iff $\hat{\delta}(p, w) = q$ and $p \Rightarrow^* w$ 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 $A \to aB$ or $A \to \epsilon$ is called *left regular* (and *right regular* if all rules are of the form $A \to Ba$ or $A \to \epsilon$).

Regular Languages and Context-Free Languages

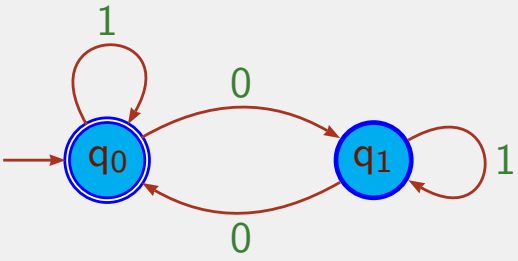
We prove by induction on $|w|$ that $p \Rightarrow^* wq$ iff $\hat{\delta}(p, w) = q$ and $p \Rightarrow^* w$ iff $\hat{\delta}(p, w) \in F$.

Base case:] 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.

Inductive step: 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:



The left regular grammar for this language is

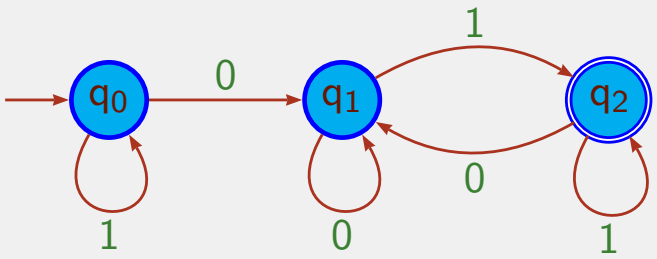
$$\begin{aligned} q_0 &\rightarrow \epsilon \mid 0q_1 \mid 1q_0 \\ q_1 &\rightarrow 0q_0 \mid 1q_1 \end{aligned}$$

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

$$q_0 \to 0q_1 \mid 1q_0 \qquad q_1 \to 0q_1 \mid 1q_2 \qquad q_2 \to \epsilon \mid 0q_1 \mid 1q_2$$

with q_0 as the start variable.

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

The right regular grammar for this language is

$$q_0 \to \epsilon \mid q_01 \qquad q_1 \to q_00 \mid q_10 \mid q_20 \qquad q_2 \to q_11 \mid q_21$$

with q_2 as the start variable.

$$q_2 \Rightarrow q_11 \Rightarrow q_201 \Rightarrow q_1101 \Rightarrow q_10101 \Rightarrow q_000101 \Rightarrow q_0100101 \Rightarrow 100101$$

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 $\alpha A\beta \to \alpha\gamma\beta$. α and β may be empty, but γ 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 $A \to Ba$, $A \to aB$ or $A \to \epsilon$.

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