

Finite Automata Theory and Formal Languages

TMV027/DIT321– LP4 2016

Lecture 11
Ana Bove

May 9th 2016

Overview of today's lecture:

- Inference, derivations and parse trees;
- Ambiguity in grammars;
- Regular grammars.

Recap: Context-Free Grammars

- Defined the so-called context-free languages;
- Developed in the mid-1950s by Noam Chomsky;
- Defined as 4-tuples $G = (V, T, \mathcal{R}, S)$;
- Notions:
 - Recursive inference;
 - Derivations: $\Rightarrow, \xRightarrow{lm}, \xRightarrow{rm}, \Rightarrow^*$;
 - Sentential forms: $S \Rightarrow^* \alpha, S \xRightarrow{lm} \alpha, S \xRightarrow{rm} \alpha$;
 - Parse trees, their heights and yields;
- $\mathcal{L}(G) = \{w \in T^* \mid S \xRightarrow{G} w\}$;
- Proofs about grammars.

Inference, Derivations and Parse Trees

Given a CFG $G = (V, T, \mathcal{R}, S)$, the following statements are equivalents:

- ① The recursive inference procedure determines that string w is in the language of the variable A ;
- ② $A \Rightarrow^* w$;
- ③ $A \xRightarrow{lm}^* w$;
- ④ $A \xRightarrow{rm}^* w$;
- ⑤ There is a parse tree with root A and yield w .

Note: The equivalences of 2–5 are also valid when the string w contains variables.

Inference, Derivations and Parse Trees (Cont.)

Showing $3 \Rightarrow 2$ and $4 \Rightarrow 2$ is trivial.

The next 7 slides contain the main ideas in the proofs of $1 \Rightarrow 5$, $5 \Rightarrow 3$, $5 \Rightarrow 4$ and $2 \Rightarrow 1$.

We will intuitively follow the proofs using the following grammar

$$\begin{aligned} E &\rightarrow 0 \mid 1 \mid E + E \mid \text{if } B \text{ then } E \text{ else } E \\ B &\rightarrow \text{True} \mid \text{False} \mid E < E \mid E == E \end{aligned}$$

and the string “*if 0 < 1 then 1 + 1 else 0*”.

From Recursive Inference to Parse Trees ($1 \Rightarrow 5$)

Theorem: Let $G = (V, T, \mathcal{R}, S)$ be a CFG and $A \in V$.

If the recursive inference tells us that w is in the language of A then there is a parse tree with root A and yield w .

Proof: By course-of-value induction on the number of steps in the recursive inference. $P(n)$: if w is obtained from A by a recursive inference with n steps, then there is a parse tree with root A and yield w .

Base case: 1 step: Then we have a production $A \rightarrow w$ and it is trivial to build the tree.

Inductive Step: $n + 1$ steps: Our IH is that the statement is true for strings x and variables B such that x is in the language of B can be recursively inferred in at most n steps.

From Recursive Inference to Parse Trees ($1 \Rightarrow 5$) (Cont.)

Suppose the last step in the inference of w uses the production $A \rightarrow X_1 X_2 \dots X_k$.

We break w up as $w_1 w_2 \dots w_k$ where:

- 1 If X_i is a terminal then $w_i = X_i$;
- 2 If X_i is a variable then w_i is the string which was inferred to be in the language of X_i .
This inference took at most n steps and we can apply IH.
Then we have a tree with root X_i and yield w_i .

We can now construct a tree with root A , first line of children X_1, X_2, \dots, X_k , and then the trees given by the IH when X_i is a variable or the corresponding terminal when X_i is a terminal.

The yield of the tree is the concatenation of the yields of the children of A which is $w_1 w_2 \dots w_k = w$.

From Trees to Leftmost Derivations (5 \Rightarrow 3)

Theorem: Let $G = (V, T, \mathcal{R}, S)$ be a CFG.

If there is a parse tree with root A and yield w then $A \xRightarrow{*}^{lm} w$ in G .

Proof: By course-of-value induction in the height of the tree.

$P(n)$: If there is a parse tree of height n with root A and yield w then $A \xRightarrow{*}^{lm} w$ in G .

Base case: Height 1: Must be a production $A \rightarrow w$ and then $A \xRightarrow{*}^{lm} w$ and $A \xRightarrow{*}^{lm} w$.

Inductive Step: Height $n > 1$: Let X_1, X_2, \dots, X_k be the children of the root A . Note that there must be a production $A \rightarrow X_1 X_2 \dots X_k$. Now:

- 1 If X_i is a terminal, define $w_i = X_i$;
- 2 If X_i is a variable, then it is the root of a subtree. Let w_i be the yield of X_i . This subtree has height less than n and then by IH $X_i \xRightarrow{*}^{lm} w_i$.

Observe that $w = w_1 w_2 \dots w_k$.

To construct a leftmost derivation for w we start at $A \xRightarrow{*}^{lm} X_1 X_2 \dots X_k$.

From Trees to Leftmost Derivations (5 \Rightarrow 3) (Cont.)

By induction on i we show that $A \xRightarrow{*}^{lm} w_1 w_2 \dots w_{i-1} X_i X_{i+1} \dots X_k$.

Base case: For $i = 0$ we already have $A \xRightarrow{*}^{lm} X_1 X_2 \dots X_k$.

Inductive Step: Assume $A \xRightarrow{*}^{lm} w_1 w_2 \dots w_{i-1} X_i X_{i+1} \dots X_k$.

- 1 If X_i is a terminal we do nothing.
So $A \xRightarrow{*}^{lm} w_1 w_2 \dots w_{i-1} w_i X_{i+1} \dots X_k$;
- 2 If X_i is a variable, by IH we have that $X_i \xRightarrow{*}^{lm} \alpha_1 \xRightarrow{*}^{lm} \alpha_2 \dots \xRightarrow{*}^{lm} w_i$.

Apply this sequence of derivations to go from $A \xRightarrow{*}^{lm} w_1 w_2 \dots w_{i-1} X_i X_{i+1} \dots X_k$ to $A \xRightarrow{*}^{lm} w_1 w_2 \dots w_{i-1} w_i X_{i+1} \dots X_k$.

Observe that each step is a leftmost derivation!

When $i = k$ then we have constructed a leftmost derivation

$A \xRightarrow{*}^{lm} w_1 w_2 \dots w_{i-1} w_i w_{i+1} \dots w_k$.

From Trees to Rightmost Derivations (5 \Rightarrow 4)

Theorem: Let $G = (V, T, \mathcal{R}, S)$ be a CFG.

If there is a parse tree with root A and yield w then $A \xRightarrow{*}^{rm} w$ in G .

Proof: The proof is similar to that of the previous theorem.

The difference is that we first need to expand X_k then X_{k-1} and so on until we get to X_1 .

From Derivations to Recursive Inference (2 \Rightarrow 1)

Theorem: Let $G = (V, T, \mathcal{R}, S)$ be a CFG.

If $A \xRightarrow{*} w$ then the recursive inference procedure applied to G determines that w is in the language of A .

Proof: By course-of-value induction on the length of the derivation of $A \xRightarrow{*} w$.

$P(n)$: if $A \xRightarrow{n} w$ then the recursive inference procedure applied to G determines that w is in the language of A .

Base case: 1 step: Then $A \rightarrow w$ is a production and the base part of the recursive inference will find that w is in the language of A .

Inductive Step: $n + 1$ steps: Our IH is that the statement holds for any derivation of at most n steps.

From Derivations to Recursive Inference ($2 \Rightarrow 1$) (Cont.)

Let $A \Rightarrow X_1 X_2 \dots X_k \Rightarrow^* w$ be the first step.

Now, we can break w as $w = w_1 w_2 \dots w_k$ where:

- 1 If X_i is a terminal then $X_i = w_i$;
- 2 If X_i is a variable then $X_i \Rightarrow^* w_i$.

This derivation takes at most n steps and then by IH w_i is inferred to be in the language of X_i .

If $A \Rightarrow X_1 X_2 \dots X_k \Rightarrow^* w$ then there must be a production $A \rightarrow X_1 X_2 \dots X_k$ and we know that each w_i is either X_i or belongs to the language of X_i .

The last step of the recursive inference procedure must use these facts to obtain that $w_1 w_2 \dots w_k$ is in the language of A .

Ambiguity in Natural (English) Language

Dictionary definition:

Ambiguity: a word or expression that can be understood in two or more possible ways.

What do I mean when I say ...

- Kids make nutritious snacks;
- The lady hit the man with an umbrella;
- He gave her cat food;
- The man saw the boy with the binoculars;
- They are hunting dogs.

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 2 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?

Example: Ambiguous Grammars

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 words like $1 + 1 * 0$:

- 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!

$$\textcircled{1} E \Rightarrow E + E \Rightarrow 1 + E \Rightarrow 1 + 0$$

$$\textcircled{2} E \Rightarrow E + E \Rightarrow E + 0 \Rightarrow 1 + 0$$

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

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 is not reflected in the grammar:
* has stronger precedence than +;
- 2 How to associate an operator is not reflected in the grammar:
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

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 terms connected by +.

In this way, terms and expressions will associate to the left.

Unambiguous Grammar for Expressions

We have then the following *unambiguous* grammar:

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

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

Example: We have $E \Rightarrow^* 1 + 1 * 0$ 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.

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 16 must be ambiguous since we have 2 leftmost derivations for $1 + 0 * 1$:

$$\begin{aligned} \textcircled{1} \quad E &\xRightarrow{lm} E + E \xRightarrow{lm} 1 + E \xRightarrow{lm} 1 + E * E \xRightarrow{lm} 1 + 0 * E \xRightarrow{lm} 1 + 0 * 1 \\ \textcircled{2} \quad E &\xRightarrow{lm} E * E \xRightarrow{lm} E + E * E \xRightarrow{lm} 1 + E * E \xRightarrow{lm} 1 + 0 * E \xRightarrow{lm} 1 + 0 * 1 \end{aligned}$$

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 slides 6–7 we constructed a leftmost derivation from a parse tree. Observe that if the trees have a node where different productions are used then so will the leftmost derivations.

Example: Balanced Parentheses

The following grammar of parenthesis expressions is ambiguous

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

since $E \xrightarrow{lm} \epsilon$ and $E \xrightarrow{lm} EE \xrightarrow{lm} \epsilon E \xrightarrow{lm} \epsilon\epsilon = \epsilon$.

Let us consider the following grammar instead:

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

We want to prove that:

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 course-of-value induction on $|w|$.

$P(n)$: for $w \in \mathcal{L}(S)$ with $|w| = n$ then $w\mathcal{L}(S) \subseteq \mathcal{L}(S)$.

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

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

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

Since $S \rightarrow (S)S$ is a production then $(\mathcal{L}(S))\mathcal{L}(S) \subseteq \mathcal{L}(S)$.

$u \in \mathcal{L}(S)$, then

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

Example: Balanced Parentheses (Cont.)

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

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

$\mathcal{L}(S) \subseteq \mathcal{L}(E)$: By course-of-value induction on $|w|$.

$P(n)$: for $w \in \mathcal{L}(S)$ with $|w| = n$ then $w \in \mathcal{L}(E)$.

Base case is trivial.

Let $|w| = n + 1$. We have that $w = (u)v$ 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 course-of-value induction on the length of $E \Rightarrow^* x$.

$P(n)$: if $E \Rightarrow^n x$ then $x \in \mathcal{L}(S)$.

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

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

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

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 $S \xRightarrow{lm}^* w$.

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

Otherwise, either $w =)v$ and there is no derivation or

$$w = (v \text{ and we have that } S \xRightarrow{lm} (S)S.$$

We now should prove (by induction on $|u|$) that

Lemma: *Given u , for any k , there is at most one leftmost derivation*

$$S(\)S)^k \xRightarrow{lm}^* u.$$

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 $S \xRightarrow{lm} (S)S \xRightarrow{lm}^* (v = w$.

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:

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

(See grammar for \mathcal{L} in Lecture 10, slide 18.)

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 Grammars

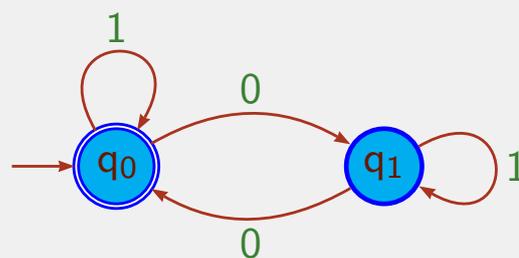
Definition: A grammar where all rules are of the form $A \rightarrow aB$ or $A \rightarrow \epsilon$ is called *left regular*.

Definition: A grammar where all rules are of the form $A \rightarrow Ba$ or $A \rightarrow \epsilon$ is called *right regular*.

Note: We will see that regular grammars generate the regular languages.

Example: Regular Grammars

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



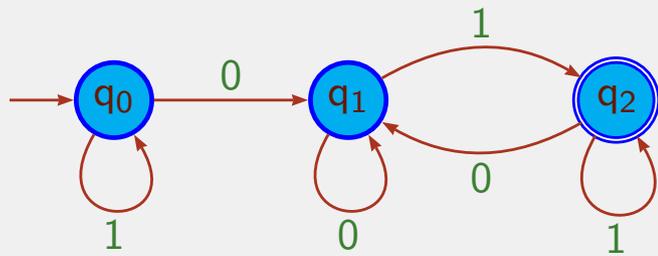
Exercise: What could the left regular grammar be for this language?

Let q_0 be the start variable.

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

Example: Regular Grammars

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



Exercise: What could the left regular grammar be for this language?

Let q_0 be the start variable.

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

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

Exercise: What could the right regular grammar be for this language?

Let q_2 be the start variable.

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

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

Overview of Next Lecture

Sections 7–7.2:

- Regular grammars and Chomsky hierarchy;
- Simplifications and normal forms for CFL;
- Pumping lemma for CFL.