Finite Automata Theory and Formal Languages

TMV027/DIT321– LP4 2013

Lecture 11

Ana Bove

May 2nd 2013

Overview of today's lecture:

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

Inference, Derivations and Parse Trees

Given a CFG $G = (V, T, \mathcal{R}, S)$ we will show the following equivalences:

1. The recursive inference procedure determines that string w is in the language of the variable A;
2. $A \Rightarrow^* w$;
3. $A \overset{lm}{\Rightarrow}^* w$;
4. $A \overset{rm}{\Rightarrow}^* w$;
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

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

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.*
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 induction on the number of steps in the recursive inference.

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 inferred in at most n steps.

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

We break w up as $w_1 w_2 \ldots w_k$ where ...

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

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 a tree with root A, first line of children $X_1, X_2, \dots, X_k$, and 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 $A \overset{lm}{\Rightarrow}^* w$ in G.*

Proof: By induction in the height of the tree.

Base case: Height 1: There must be a production $A \to w$ and then $A \overset{lm}{\Rightarrow} w$ and $A \overset{lm}{\Rightarrow}^* w$.

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 $A \to X_1 X_2 \dots X_k$. 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 $X_i \overset{lm}{\Rightarrow}^* w_i$.

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

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

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

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

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

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

1. If X_i is a terminal we do nothing. So $A \overset{lm}{\Rightarrow}^* w_1 w_2 \dots w_{i-1} w_i X_{i+1} \dots X_k$;
2. If X_i is a variable, by IH we have that $X_i \overset{lm}{\Rightarrow} \alpha_1 \overset{lm}{\Rightarrow} \alpha_2 \dots \overset{lm}{\Rightarrow} w_i$. Apply this sequence of derivations to go from $A \overset{lm}{\Rightarrow}^* w_1 w_2 \dots w_{i-1} X_i X_{i+1} \dots X_k$ to $A \overset{lm}{\Rightarrow}^* w_1 w_2 \dots w_{i-1} w_i X_{i+1} \dots X_k$. Observe that each step is a leftmost derivation!

When $i = k$ then we have constructed a leftmost derivation $A \overset{lm}{\Rightarrow}^* w_1 w_2 \dots w_{i-1} w_i w_{i+1} \dots w_k$.

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 $A \overset{rm}{\Rightarrow}^* w$ 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 $A \Rightarrow^* w$ then the recursive inference procedure applied to G determines that w is in the language of A.*

Proof: By induction on the length of the derivation of $A \Rightarrow^* w$.

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.

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

Now, we can break w as $w = w_1 w_2 \ldots w_k$ where ...

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

Now, we can break w as $w = w_1 w_2 \ldots w_k$ 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 $A \Rightarrow X_1 X_2 \ldots X_k \Rightarrow^* w$ then there must be a production $A \rightarrow X_1 X_2 \ldots X_k$ 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 $w_1 w_2 \ldots w_k$ 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 way.
```

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;
- I convinced her kids were funny.

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 word.

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

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

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 15 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 slides 5–6 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: 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 can 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: Balanced Parentheses

The following grammar of parenthesis expressions is ambiguous

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

Let us consider the following grammar instead:

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

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 $|w|$ 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 $|u|, |v| \leqslant n$. Hence the IH hold for u and v.

The, 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 course of value 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 course of value induction on the length of $E \Rightarrow^* x$.

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

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

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 Lecture 10, slide 16.)

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

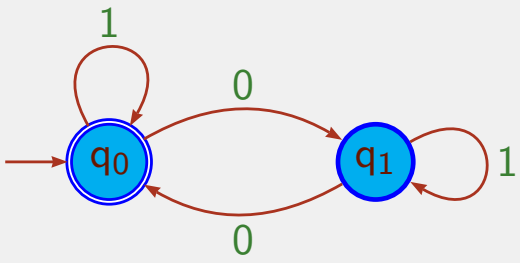
Definition: A grammar where all rules are of the form $A \to aB$ or $A \to \epsilon$ is called *left regular*.

Definition: A grammar where all rules are of the form $A \to Ba$ or $A \to \epsilon$ 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:



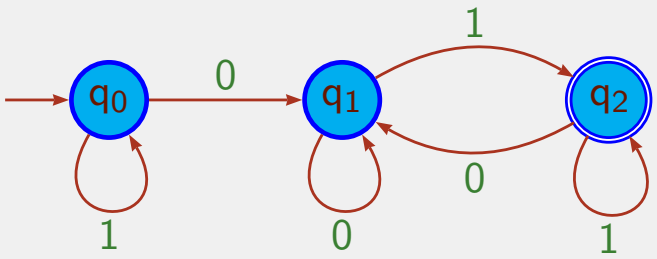
Exercise: What could be the left regular grammar for this language?

Let q_0 be the start variable.

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

Example: Regular Grammars

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



Exercise: What could be the left regular grammar for this language? Let q_0 be the start variable.

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

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

Exercise: What could be the left regular grammar for this language? Let q_2 be the start variable.

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

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

Overview of Next Lecture

OBS: Next lecture on Tuesday at *10:00*!

Sections 7–7.2:

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