Finite Automata Theory and Formal Languages

TMV027/DIT321– LP4 2018

Lecture 11

Ana Bove

April 26th 2018

Recap: Regular Languages

- Decision properties of RL:
  - Is it empty?
  - Does it contain this word?
  - Contains ϵ, contains at most ϵ, is infinite?
- Equivalence of languages using table-filling algorithm;
- Minimisation of DFA using table-filling algorithm.

Overview of Today's Lecture

- Context-free grammars;
- Derivations;
- Parse trees;
- Proofs in grammars.

Contributes to the following learning outcome:

- Explain and manipulate the diff. concepts in automata theory and formal lang;
- Prove properties of languages, grammars and automata with rigorously formal mathematical methods;
- Design automata, regular expressions and context-free grammars accepting or generating a certain language;
- Describe the language accepted by an automata or generated by a regular expression or a context-free grammar;
- Determine if a certain word belongs to a language;
- Differentiate and manipulate formal descriptions of lang, automata and grammars.

Context-free Grammars

We have seen that not all languages are regular.

Context-free grammars (CFG) define the so called *context-free languages*.

They have been developed in the mid-1950s by Noam Chomsky.

We will see (next lecture) that

regular languages $\subset$ *context-free languages*

CFG play an important role in the description and design of programming languages and compilers, for example, for the implementation of parsers.

Example: Palindromes

Let us consider the language $\mathcal{P}$ of *palindromes* over $\Sigma = \{0, 1\}$.

That is, $\mathcal{P} = \{w \in \Sigma^* \mid w = \text{rev}(w)\}$.

Example of palindromes over Σ are ϵ, 0110, 00011000, 010.

We can use the Pumping Lemma for RL to show that $\mathcal{P}$ is not regular.

How can $\mathcal{P}$ be defined?

Consider the inductive definition of a (very similar) language $\mathcal{L}$:

- ϵ, 0 and 1 are in $\mathcal{L}$;
- if $w \in \mathcal{L}$ then $0w0$ and $1w1$ are also in $\mathcal{L}$.

Example: CFG for Palindromes

$$\begin{array}{lcl} P & \rightarrow & \epsilon \\ P & \rightarrow & 0 \\ P & \rightarrow & 1 \\ P & \rightarrow & 0P0 \\ P & \rightarrow & 1P1 \end{array}$$

The *variable* P represents the set of strings that are palindromes.

The *rules* explain how to construct the strings in the language.

Example: CFG for Simple Expressions

We can define several sets in a grammar.

Example: Here we define 2 sets: those representing simple numerical expressions (denoted by E) and those representing Boolean expressions (denoted by B).

Note that they depend on each other!

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

Compact Notation

We can have a more compact notation for the productions defining elements in a certain set.

Example: Palindromes can be defined as

$$P \to \epsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1$$

Example: The expressions can be defined as

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

Context-Free Grammars

Definition: A *context-free grammar* is a 4-tuple $G = (V, T, \mathcal{R}, S)$ where:

- V is a finite set of *variables* or *non-terminals*: each variable represents a language or set of strings;
- T is a finite set of *terminals*: think of T as the alphabet of the language we are defining;
- $\mathcal{R}$ is a finite set of *rules* or *productions* which recursively define the language. Each production consists of:
  - A variable being defined in the production;
  - The symbol "$\rightarrow$";
  - A string of 0 or more terminals and variables called the *body* of the production;
- S is the *start variable* and represents the language we are defining; other variables define the auxiliary classes needed to define our language.

Example: C++ Compound Statements

A context free grammar for statements:

$$(\{S, LC, C, E, Id, LLoD, L, D\}, \{a, \ldots, A, \ldots, 0, \ldots, (,), \ldots\}, \mathcal{R}, S)$$

with $\mathcal{R}$ as follows:

$$\begin{array}{lcl}
S & \rightarrow & \{LC\} \\
LC & \rightarrow & \epsilon \mid C\ LC \\
C & \rightarrow & S \mid \mathit{if}\ (E)\ C \mid \mathit{if}\ (E)\ C\ \mathit{else}\ C \mid \\
 & & \mathit{while}\ (E)\ C \mid \mathit{do}\ C\ \mathit{while}\ (E) \mid \mathit{for}\ (C\ E; E)\ C \mid \\
 & & \mathit{case}\ E : C \mid \mathit{switch}\ (E)\ C \mid \mathit{return}\ E; \mid \mathit{goto}\ Id; \\
 & & \mathit{break}; \mid \mathit{continue}; \\
E & \rightarrow & \ldots \\
Id & \rightarrow & L\ LLoD \\
LLoD & \rightarrow & L\ LLoD \mid D\ LLoD \mid \epsilon \\
L & \rightarrow & A \mid B \mid \ldots \mid Z \mid a \mid b \mid \ldots \mid z \\
D & \rightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{array}$$

Notation for Context-Free Grammars

We use the following convention when working with CFG:

- $a, b, c, \dots, 0, 1, \dots, (,), +, \%, \dots$ are terminal symbols;
- $A, B, C, \dots$ are variables (non-terminals);
- $w, x, y, z, \dots$ are strings of terminals;
- $X, Y, \dots$ are either a terminal or a variable;
- $\alpha, \beta, \gamma, \dots$ are strings of terminals and/or variables;
  In particular, they can also represent strings with *only* variables.

Working with Grammars

We use the productions of a CFG to infer that a string w is in the language of a given variable.

Example: Let us see if 0110110 is a palindrome.

We can do this in 2 (equivalent) ways:

Recursive inference: From body to head.

(Similar to how we would construct the element in the inductive set.)

1)	0	is palindrome by rule	$P \to 0$
2)	101	is palindrome using	1) and rule $P \to 1P1$
3)	11011	is palindrome using	2) and rule $P \to 1P1$
4)	0110110	is palindrome using	3) and rule $P \to 0P0$

Derivation: Notation: $\Rightarrow$

From head to body using the same rules in the opposite order.

$P \Rightarrow 0P0 \Rightarrow 01P10 \Rightarrow 011P110 \Rightarrow 0110110$

Formal Definition of a Derivation

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

Definition: Let $A \in V$ and $\alpha, \beta \in (V \cup T)^*$. Let $A \to \gamma \in \mathcal{R}$.

Then $\alpha A \beta \Rightarrow \alpha\gamma\beta$ is *one derivation step* (alternative $\alpha A \beta \overset{G}{\Rightarrow} \alpha\gamma\beta$).

Example: $B \Rightarrow E == E \Rightarrow 0 == E \Rightarrow 0 == E + E \Rightarrow 0 == E + 1 \Rightarrow 0 == 0 + 1$

Reflexive-transitive Closure of a Derivation

We can define a relation performing zero or more derivation steps.

Definition: The *reflexive-transitive closure* of $\Rightarrow$ is the relation $\Rightarrow^*$ (alternative $\overset{G}{\Rightarrow}{}^*$) defined as follows:

$$\frac{}{\alpha \Rightarrow^* \alpha} \qquad\qquad \frac{\alpha \Rightarrow^* \beta \qquad \beta \Rightarrow \gamma}{\alpha \Rightarrow^* \gamma}$$

Example: $B \Rightarrow^* B$
$B \Rightarrow^* 0 + E < E$
$B \Rightarrow^* 0 == 0 + 1$

Note: We denote $A \Rightarrow^n \alpha$ when α is derived from A in n steps.

Example: $B \Rightarrow^0 B$

Leftmost and Rightmost Derivations

At every derivation step we can choose to replace *any* variable by the right-hand side of one of its productions.

Two particular derivations are:

Leftmost derivation: Notations: $\overset{lm}{\Rightarrow}$ and $\overset{lm}{\Rightarrow}^*$.
At each step we choose to replace the *leftmost variable*.

Example: $B \overset{lm}{\Rightarrow}^* 0 == 0+1$

$$B \overset{lm}{\Rightarrow} E == E \overset{lm}{\Rightarrow} 0 == E \overset{lm}{\Rightarrow} 0 == E + E \overset{lm}{\Rightarrow} 0 == 0 + E \overset{lm}{\Rightarrow} 0 == 0 + 1$$

Rightmost derivation: Notations: $\overset{rm}{\Rightarrow}$ and $\overset{rm}{\Rightarrow}^*$.
At each step we choose to replace the *rightmost variable*.

Example: $B \overset{rm}{\Rightarrow}^* 0 == 0+1$

$$B \overset{rm}{\Rightarrow} E == E \overset{rm}{\Rightarrow} E == E + E \overset{rm}{\Rightarrow} E == E + 1 \overset{rm}{\Rightarrow} E == 0 + 1 \overset{rm}{\Rightarrow} 0 == 0 + 1$$

Sentential Forms

Derivations from the start variable have a special role.

Definition: Let $G = (V, T, \mathcal{R}, S)$ be a CFG and let $\alpha \in (V \cup T)^*$.
α is called a *sentential form* if $S \Rightarrow^* \alpha$.

We say *left sentential form* if $S \overset{lm}{\Rightarrow}^* \alpha$ or *right sentential form* if $S \overset{rm}{\Rightarrow}^* \alpha$.

Example: $010P010$ is a sentential form since

$$P \Rightarrow 0P0 \Rightarrow 01P10 \Rightarrow 010P010$$

Language of a Grammar

Definition: Let $G = (V, T, \mathcal{R}, S)$ be a CFG.
The *language of G*, denoted $\mathcal{L}(G)$ is the set of terminal strings that can be derived from the start variable.

$$\mathcal{L}(G) = \{w \in T^* \mid S \overset{G}{\Rightarrow^*} w\}$$

Definition: If $\mathcal{L}$ is the language of a certain context-free grammar, then $\mathcal{L}$ is said to be a *context-free language*.

Examples of Context Free Grammars

Example: Construct a CFG generating $\{0^i1^j \mid j \geqslant i \geqslant 0\}$.

$$S \to \epsilon \mid 0S1 \mid S1$$

Example: Construct a CFG generating $\{w \in \{0,1\}^* \mid \#_0(w) = \#_1(w)\}$.

$$S \to \epsilon \mid 0S1 \mid 1S0 \mid SS$$

Example: Construct a grammar for the following language:

$$\{a^nb^nc^md^m \mid n, m \geqslant 1\} \cup \{a^nb^mc^md^n \mid n, m \geqslant 1\}$$

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

Observations on Derivations

Observe: If we have $A \Rightarrow^* \gamma$ then we also have $\alpha A \beta \Rightarrow^* \alpha\gamma\beta$.

The same sequence of steps that took us from A to γ will also take us from $\alpha A \beta$ to $\alpha\gamma\beta$.

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

This same derivation justifies $E == E \Rightarrow^* E == 0 + 1$ since

$$E == E \Rightarrow E == E + E \Rightarrow E == 0 + E \Rightarrow E == 0 + 1$$

Observations on Derivations

Observe: If we have $A \Rightarrow X_1 X_2 \ldots X_k \Rightarrow^* w$, then we can break w into pieces $w_1, w_2, \ldots, w_k$ such that $X_i \Rightarrow^* w_i$. If X_i is a terminal then $X_i = w_i$.

This can be showed by proving (by induction on the length of the derivation) that if $X_1 X_2 \ldots X_k \Rightarrow^* \alpha$ then all positions of α that come from expansion of X_i are to the left of all positions that come from the expansion of X_j if $i < j$.

To obtain $X_i \Rightarrow^* w_i$ from $A \Rightarrow^* w$ we need to remove all positions of the sentential form (see next slide) that are to the left and to the right of all the positions derived from X_i, and all steps not relevant for the derivation of w_i from X_i.

Example: Let $B \Rightarrow E == E \Rightarrow E == E + E \Rightarrow E == E + 0 \Rightarrow$ $E == E + E + 0 \Rightarrow E == 0 + E + 0 \Rightarrow 1 == 0 + E + 0 \Rightarrow 1 == 0 + 1 + 0$.

The derivation from the middle E in the sentential form $E == E + E$ is $E \Rightarrow E + E \Rightarrow 0 + E \Rightarrow 0 + 1$.

Parse Trees

Parse trees are a way to represent derivations.

A parse tree reflects the internal structure of a word of the language.

Using parse trees it becomes very clear which is the variable that was replaced at each step.

In addition, it becomes clear which terminal symbols where generated/derived form a particular variable.

Parse trees are very important in compiler theory.

In a compiler, a *parser* takes the source code into its *abstract* tree, which is an abstract version of the parse tree.

This parse tree is the structure of the program.

Parse Trees

Definition: Let $G = (V, T, \mathcal{R}, S)$ be a CFG.
The *parse trees* for G are trees with the following conditions:

- Nodes are labelled with a variable;
- Leaves are either variables, terminals or ϵ;
- If a node is labelled A and it has children labelled $X_1, X_2, \ldots, X_n$ respectively from left to right, then it must exist in $\mathcal{R}$ a production of the form $A \to X_1X_2 \ldots X_n$.

Note: If a leaf is ϵ it should be the only child of its parent A and there should be a production $A \to \epsilon$.

Note: Of particular importance are the parse trees with root S.

Exercise: Construct the parse trees for $0 == E + 1$ and for 001100.

Height of a Parse Tree

Definition: The *height* of a parse tree is the maximum length of a path from the root of the tree to one of its leaves.

Observe: We count the *edges* in the tree, and not the number of nodes and the leaf in the path.

Example: The height of the parse tree for $0 == E + 1$ is 3 and the one for 001100 is 4.

Yield of a Parse Tree

Definition: A *yield* of a parse tree is the string resulted from concatenating all the leaves of the tree from left to right.

Observations:

- We will show than the yield of a tree is a string derived from the root of the tree;
- Of particular importance are the yields that consist of only terminals; that is, the leaves are either terminals or ϵ;
- When, in addition, the root is S then we have a parse tree for a string in the language of the grammar;
- We will see that yields can be used to describe the language of a grammar.

Context-Free Languages and Inductive Definitions

For each CFG there is a corresponding inductive definition (see slide 4).

Example: Consider the grammar for palindromes in slide 5 (and 7).

Now consider the following inductive definition:

Base Cases:
- The empty string is a palindrome;
- 0 and 1 are palindromes.

Inductive Steps: If w is a palindrome, then so are $0w0$ and $1w1$.

A natural way to do proofs on context-free languages defined with an inductive set is to follow this inductive structure.

How to prove properties when the language is defined with a grammar?

Proofs About a Grammar

When we want to prove something about a grammar we usually need to prove an statement for each of the variables in the grammar.

See notes on induction for an example!

(Compare this with proofs about FA where we needed statements for each state.)

Proofs about grammars are in general done by:

- (course-of-value) induction on the length of a certain string of the language;
- (course-of-value) induction on the length (number of steps) of a derivation of a certain string;
- induction on the structure of the strings/words in the language;
  (Recall words are either empty or not empty!)
- (course-of-value) induction on the height of the parse tree.

Example: Proof About a Grammar

Exercise: Consider the grammar $S \to \epsilon \mid 0S1 \mid 1S0 \mid SS$.
Prove that $\forall\, w.$ if $S \Rightarrow^* w$ then $\#_0(w) = \#_1(w)$.

Proof: Our $P(n) : \forall\, w.$ if $S \Rightarrow^n w$ then $\#_0(w) = \#_1(w)$.
Proof by course-of-value induction on the length n of the derivation.

Base cases: If length is 1 then we have $S \Rightarrow \epsilon$ and $\#_0(\epsilon) = \#_1(\epsilon)$ holds.

Inductive Steps: Our IH is that $\forall\, i. 1 \leqslant i \leqslant n, \forall\, w'.$ if $S \Rightarrow^i w'$ then $\#_0(w') = \#_1(w')$.
Let $S \Rightarrow^{n+1} w$ with $n > 0$. Then we have 3 cases:

- $S \Rightarrow 0S1 \Rightarrow^n w$: Here $w = 0w_1 1$ with $S \Rightarrow^n w_1$.
  By IH $\#_0(w_1) = \#_1(w_1)$ hence
  $\#_0(w) = \#_0(0w_1 1) = 1 + \#_0(w_1) = 1 + \#_1(w_1) = \#_1(w)$;
- $S \Rightarrow 1S0 \Rightarrow^n 1w_1 0$: Similar ...
- $S \Rightarrow SS \Rightarrow^n w$: Here $w = w_1 w_2$ with $S \Rightarrow^i w_1$ and $S \Rightarrow^j w_2$ for $i, j < n$.
  By IH $\#_0(w_1) = \#_1(w_1)$ and $\#_0(w_2) = \#_1(w_2)$.
  Hence $\#_0(w) = \#_0(w_1) + \#_0(w_2) = \#_1(w_1) + \#_1(w_2) = \#_1(w)$.

Example: Proof About a Grammar

Lemma: *Let G be the grammar presented on slide 5.*
Then $\mathcal{L}(G)$ is the set of palindromes over $\{0, 1\}$.

Proof: We will prove that $\forall\, w.$ if $w \in \{0, 1\}^*$, then $w \in \mathcal{L}(G)$ iff $w = \text{rev}(w)$.

If) Our $P(n) : \forall w.$ if w is a palindrome and $|w| = n$ then $w \in \mathcal{L}(G)$, that is, $P \Rightarrow^* w$.
Proof by course-of-value induction on n (that is, $|w|$).

Base cases: If $|w| = 0$ or $|w| = 1$ then w is ϵ, 0 or 1.
We have productions $P \to \epsilon$, $P \to 0$ and $P \to 1$.
Then $P \Rightarrow^* w$ so $w \in \mathcal{L}(G)$.

Inductive Steps: Assume w such that $|w| > 1$.
Our IH is that the property holds for any w' such that $|w'| < |w|$.
Since $w = \text{rev}(w)$ then $w = 0w'0$ or $w = 1w'1$, and $w' = \text{rev}(w')$.
$|w'| < |w|$ so by IH then $P \Rightarrow^* w'$.
If $w = 0w'0$ we have $P \Rightarrow 0P0 \Rightarrow^* 0w'0 = w$ so $w \in \mathcal{L}(G)$.
Similarly if $w = 1w'1$.

Example: Proof About a Grammar (Cont.)

Only-if) We prove that $\forall\, w$. if $w \in \mathcal{L}(G)$ then w is a palindrome.

Our $P(n)$: $\forall\, w$. if $P \Rightarrow^n w$ then $w = \text{rev}(w)$.

Proof by mathematical induction on n (length of the derivation of w).

Base case: If the derivation is in one step then we should have $P \Rightarrow \epsilon$, $P \Rightarrow 0$ and $P \Rightarrow 1$. In all cases we have $w = \text{rev}(w)$.

Inductive Step: Our IH is that $\forall\, w'$. if $P \Rightarrow^n w'$ with $n > 0$ then $w' = \text{rev}(w')$.

Assume $P \Rightarrow^{n+1} w$. The we have 2 cases:

- $P \Rightarrow 0P0 \Rightarrow^n 0w'0 = w$;
- $P \Rightarrow 1P1 \Rightarrow^n 1w'1 = w$.

Observe that in both cases $P \Rightarrow^n w'$ with $n > 0$.

Hence by IH $w' = \text{rev}(w')$ so $w = \text{rev}(w)$.

Overview of Next Lecture

Sections 5.2.3–5.2.6, 5.4:

- Inference, derivations and parse trees;
- Ambiguity in grammars;
- More about proofs on grammars and languages.

Overview of next Week

Mon 30	Tue 1	Wed 2	Thu 3	Fri 4
			Lec 13-15 HB3 CFG.	
		15-17 EL41 Consultation		