Finite Automata Theory and Formal Languages

TMV027/DIT321– LP4 2017

Lecture 13
Ana Bove

May 8th 2017

Overview of today's lecture:

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

Recap: Context-Free Grammars

- Equivalence between recursive inference, (leftmost/rightmost) derivations and parse trees;
- Ambiguous grammars;
- Inherent ambiguity;
- Proofs about grammars and languages.

Regular Grammars

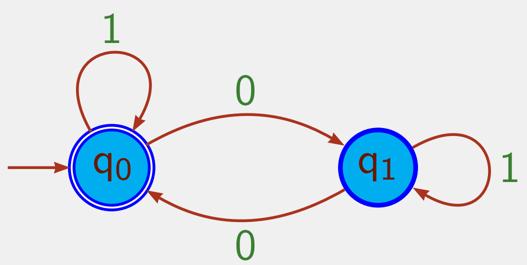
Definition: A grammar where all rules are of the form $A \rightarrow aB$ or $A \rightarrow \epsilon$ is called *left regular*.

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:



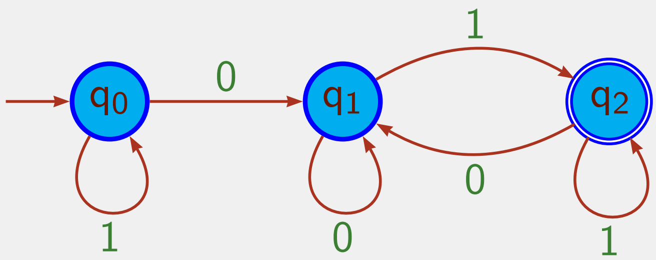
Exercise: What could the left regular grammar be for this language?

Let q_0 be the start variable.

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

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.

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

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

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

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

$$q_2 \Rightarrow q_1 1 \Rightarrow q_2 01 \Rightarrow q_1 101 \Rightarrow q_1 0101 \Rightarrow q_0 00101 \Rightarrow q_0 100101 \Rightarrow 100101$$

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 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 $w \in \mathcal{L}(D)$.

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 $p \to \epsilon$.
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 $w = av$.
Then $\hat{\delta}(p, av) = \hat{\delta}(\delta(p, a), v)$ with $|v| = n$.
By IH $\delta(p, a) \Rightarrow^* vq$ iff $\hat{\delta}(\delta(p, a), v) = q$.
By construction we have a rule $p \to a\delta(p, a)$.
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 $\hat{\delta}(p, av) = \hat{\delta}(\delta(p, a), v) \in F$.

Chomsky Hierarchy

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

Type 0: *Unrestricted grammars*
Rules are of the form $\alpha \to \beta$, α must be non-empty.
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*
Rules are of the form $A \to \alpha$, α can be empty.
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.

Generating, Reachable, Useful and Useless Symbols

Let $G = (V, T, \mathcal{R}, S)$ be a CFG.
Let $X \in V \cup T$ and let $\alpha, \beta \in (V \cup T)^*$.

Definition: X is *reachable* if $S \Rightarrow^* \alpha X \beta$.
(This is similar to accessible states in FA.)

Definition: X is *generating* if $X \Rightarrow^* w$ for some $w \in T^*$.

Definition: The symbol X is *useful* if $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$ for some $w \in T^*$.
Note: A symbol that is useful should be generating and reachable.

Definition: X is *useless* iff it is not useful.

We shall "simplify" the grammars by eliminating useless symbols.

Computing the Generating Symbols

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

The following inductive procedure computes the generating symbols of G:

Base Case: All elements of T are generating;

Inductive Step: If a production $A \to \alpha$ is such that all symbols of α are known to be generating, then A is also generating. Observe that α could be ϵ.

(The inductive step is to be applied until no new symbols are found generating.)

Theorem: *The procedure above finds all and only the generating symbols of a grammar.*

Proof: See Theorem 7.4 in the book.

Example: Generating Symbols

Consider the grammar over $\{a\}$ given by the rules:

$$\begin{array}{lcl} S & \to & aS \mid W \mid U \\ W & \to & aW \\ U & \to & a \\ V & \to & aa \end{array}$$

a is generating.

U and V are generating since $U \to a$ and $V \to aa$.

S is generating since $S \to U$.

No other symbol is found generating so W is not generating.

After eliminating the non-generating symbols and their productions we get

$$S \to aS \mid U \qquad U \to a \qquad V \to aa$$

Computing the Reachable Symbols

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

The following inductive procedure computes the reachable symbols of G:

Base Case: The start variable S is reachable;

Inductive Step: If A is reachable and we have a production $A \to \alpha$ then all symbols in α are reachable.

(The inductive step is to be applied until no new symbols are found reachable.)

Theorem: *The procedure above finds all and only the reachable symbols of a grammar.*

Proof: See Theorem 7.6 in the book.

Example: Reachable Symbols

Consider the grammar given by the rules:

$$S \to aB \mid BC \qquad C \to b$$
$$A \to aA \mid c \mid aDb \qquad D \to B$$
$$B \to DB \mid C$$

S is reachable.

Hence a, B and C are reachable.

Then b and D are reachable.

No other symbol are found reachable so A and c are not reachable.

After eliminating the non-reachable symbols and their productions we get

$$S \to aB \mid BC \qquad C \to b$$
$$B \to DB \mid C \qquad D \to B$$

Eliminating Useless Symbols

It is important in which order we check generating and reachable symbols!

Example: Consider the following grammar

$$S \to AB \mid a \qquad A \to b$$

If we first check for generating symbols and then for reachability we get

$$S \to a$$

If we first check for reachability and then for generating we get

$$S \to a \qquad A \to b$$

Eliminating Useless Symbols

Theorem: *Let $G = (V, T, \mathcal{R}, S)$ be a CFG and let $\mathcal{L}(G) \neq \emptyset$.*
Let $G' = (V', T', \mathcal{R}', S)$ be constructed as follows:

1. *First, eliminate all non-generating symbols and all productions involving one or more of those symbols;*
2. *Then, eliminate all non-reachable symbols and all productions involving one or more of those symbols.*

Then G' has no useless symbols and $\mathcal{L}(G) = \mathcal{L}(G')$.

Proof: See Theorem 7.2 in the book.

Example: Eliminating Useless Symbols

Consider the grammar given by the rules:

$$
\begin{array}{lcllcl}
S & \to & gAe \mid aYB \mid CY & A & \to & bBY \mid ooC \\
B & \to & dd \mid D & C & \to & jVB \mid gI \\
D & \to & n & U & \to & kW \\
V & \to & baXXX \mid oV & W & \to & c \\
X & \to & fV & Y & \to & Yhm
\end{array}
$$

After eliminating non-generating symbols:

$$
\begin{array}{lcllcl}
S & \to & gAe & A & \to & ooC \\
B & \to & dd \mid D & C & \to & gI \\
D & \to & n & U & \to & kW \\
 & & & W & \to & c
\end{array}
$$

After eliminating non-reachable symbols:

$$S \to gAe \qquad A \to ooC \qquad C \to gI$$

What is the language generated by the grammar?

Nullable Variables

Definition: A variable A is *nullable* if $A \Rightarrow^* \epsilon$.

Note: Observe that only variables are nullable!

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

The following inductive procedure computes the nullable variables of G:

Base Case: If $A \to \epsilon$ is a production then A is nullable;

Inductive Step: If $B \to X_1 X_2 \ldots X_k$ is a production and all the X_i are nullable then B is also nullable.

(The inductive step is to be applied until no new symbols are found nullable.)

Theorem: *The procedure above finds all and only the nullable variables of a grammar.*

Proof: See Theorem 7.7 in the book.

Eliminating ϵ-Productions

Definition: An *ϵ-production* is a production of the form $A \to \epsilon$.

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

The following procedure eliminates the ϵ-production of G:

1. Determine all nullable variables of G;
2. Build $\mathcal{P}$ with all the productions of $\mathcal{R}$ plus a rule $A \to \alpha\beta$ whenever we have $A \to \alpha B \beta$ and B is nullable.
   Note: If $A \to X_1 X_2 \ldots X_k$ and all X_i are nullable, we do not include the case where all the X_i are absent;
3. Construct $G' = (V, T, \mathcal{R}', S)$ where $\mathcal{R}'$ contains all the productions in $\mathcal{P}$ except for the ϵ-productions.

Theorem: *The grammar G' constructed from the grammar G as above is such that $\mathcal{L}(G') = \mathcal{L}(G) - \{\epsilon\}$.*

Proof: See Theorem 7.9 in the book.

Example: Eliminating ϵ-Productions

Example: Consider the grammar given by the rules:

$$S \to aSb \mid SS \mid \epsilon$$

By eliminating ϵ-productions we obtain

$$S \to ab \mid aSb \mid S \mid SS$$

Example: Consider the grammar given by the rules:

$$S \to AB \qquad A \to aAA \mid \epsilon \qquad B \to bBB \mid \epsilon$$

By eliminating ϵ-productions we obtain

$$S \to A \mid B \mid AB \qquad A \to a \mid aA \mid aAA \qquad B \to b \mid bB \mid bBB$$

Eliminating Unit Productions

Definition: A *unit production* is a production of the form $A \to B$.
(This is similar to ϵ-transitions in a ϵ-NFA.)

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

The following procedure eliminates the unit production of G:

1. Build $\mathcal{P}$ with all the productions of $\mathcal{R}$ plus a rule $A \to \alpha$ whenever we have $A \to B$ and $B \to \alpha$;
2. Construct $G' = (V, T, \mathcal{R}', S)$ where $\mathcal{R}'$ contains all the productions in $\mathcal{P}$ except for the unit production.

Theorem: *The grammar G' constructed from the grammar G as above is such that $\mathcal{L}(G') = \mathcal{L}(G)$.*

Proof: See Theorem 7.13 in the book.

Example: Eliminating Unit Productions

Consider the grammar given by the rules:

$$\begin{array}{lll} S & \rightarrow & CBh \mid D \\ B & \rightarrow & Sf \mid ggg \\ D & \rightarrow & E \mid SABC \end{array} \qquad \begin{array}{lll} A & \rightarrow & aaC \\ C & \rightarrow & cA \mid d \mid C \\ E & \rightarrow & be \end{array}$$

By eliminating unit productions we obtain:

$$\begin{array}{lll} S & \rightarrow & CBh \mid be \mid SABC \\ B & \rightarrow & Sf \mid ggg \\ D & \rightarrow & be \mid SABC \end{array} \qquad \begin{array}{lll} A & \rightarrow & aaC \\ C & \rightarrow & cA \mid d \\ E & \rightarrow & be \end{array}$$

Simplification of a Grammar

Theorem: *Let $G = (V, T, \mathcal{R}, S)$ be a CFG whose language contains at least one string other than ϵ. If we construct G' by*

1. *First, eliminating ϵ-productions;*
2. *Then, eliminating unit productions;*
3. *Finally, eliminating useless symbols;*

using the procedures shown before then $\mathcal{L}(G') = \mathcal{L}(G) - \{\epsilon\}$.

In addition, G' contains no ϵ-productions, no unit productions and no useless symbols.

Proof: See Theorem 7.14 in the book.

Note: It is important to apply the steps in this order!

Chomsky Normal Form

Definition: A CFG is in *Chomsky Normal Form* (CNF) if G has no useless symbols and all the productions are of the form $A \to BC$ or $A \to a$.

Note: Observe that a CFG that is in CNF has no unit or ϵ-productions!

Theorem: *For any CFG G whose language contains at least one string other than ϵ, there is a CFG G' that is in Chomsky Normal Form and such that $\mathcal{L}(G') = \mathcal{L}(G) - \{\epsilon\}$.*

Proof: See Theorem 7.16 in the book.

Constructing a Chomsky Normal Form

Let us assume G has no ϵ- or unit productions and no useless symbols.

Then every production is of the form $A \to a$ or $A \to X_1X_2 \ldots X_k$ for $k > 1$.

If X_i is a terminal introduce a new variable A_i and a new rule $A_i \to X_i$ (if no such rule exists for X_i with a variable that has no other rules).

Use A_i in place of X_i in any rule whose body has length > 1.

Now, all rules are of the form $B \to b$ or $B \to C_1C_2 \ldots C_k$ with all C_j variables.

Introduce $k - 2$ new variables and break each rule $B \to C_1C_2 \ldots C_k$ as

$$B \to C_1D_1 \quad D_1 \to C_2D_2 \quad \cdots \quad D_{k-2} \to C_{k-1}C_k$$

Example: Chomsky Normal Form

Example: Consider the grammar given by the rules:

$$S \to aSb \mid SS \mid ab$$

We first obtain

$$S \to ASB \mid SS \mid AB \qquad A \to a \qquad B \to b$$

Then we build a grammar in Chomsky Normal Form

$$\begin{array}{lll} S & \to & AC \mid SS \mid AB \\ C & \to & SB \end{array} \qquad \begin{array}{lll} A & \to & a \\ B & \to & b \end{array}$$

Example: Observe however that

$$S \to aa \mid a$$

is NOT equivalent to

$$S \to SS \mid a$$

Instead we need to build

$$S \to AA \mid a \qquad A \to a$$

Pumping Lemma for Left Regular Languages

Let $G = (V, T, \mathcal{R}, S)$ be a left regular grammar and let $n = |V|$.

If $a_1 a_2 \ldots a_m \in \mathcal{L}(G)$ for $m > n$, then any derivation

$$S \Rightarrow a_1 A_1 \Rightarrow a_1 a_2 A_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_i A \Rightarrow \ldots \Rightarrow a_1 \ldots a_j A \Rightarrow \ldots \Rightarrow a_1 \ldots a_m$$

has length m and there is at least one variable A which is used twice.

(Pigeon-hole principle)

If $x = a_1 \ldots a_i$, $y = a_{i+1} \ldots a_j$ and $z = a_{j+1} \ldots a_m$, we have $|xy| \leqslant n$ and $xy^k z \in \mathcal{L}(G)$ for all k.

Pumping Lemma for Context-Free Languages

Theorem: *Let $\mathcal{L}$ be a context-free language.*
Then, there exists a constant n—which depends on $\mathcal{L}$—such that for every $w \in \mathcal{L}$ with $|w| \geqslant n$, it is possible to break w into 5 strings x, u, y, v and z such that $w = xuyvz$ and

1. $|uyv| \leqslant n$;
2. $uv \neq \epsilon$, *that is, either u or v is not empty;*
3. $\forall k \geqslant 0.\ xu^k yv^k z \in \mathcal{L}$.

Proof: (Sketch)

We can assume that the language is presented by a grammar in Chomsky Normal Form, working with $\mathcal{L} - \{\epsilon\}$.

Observe that parse trees for grammars in CNF have at most 2 children.

Note: If $m + 1$ is the height of a parse tree for w, then $|w| \leqslant 2^m$.
(Prove this as an exercise!)

Proof Sketch: Pumping Lemma for Context-Free Languages

Let $|V| = m > 0$. Take $n = 2^m$ and w such that $|w| \geqslant 2^m$.

Any parse tree for w has a path from root to leave of length at least $m + 1$.

Let $A_0, A_1 \ldots, A_k$ be the variables in the path. We have $k \geqslant m$.

Then at least 2 of the last $m + 1$ variables should be the same,
say A_i and A_j.

Observe figures 7.6 and 7.7 in pages 282–283.

See Theorem 7.18 in the book for the complete proof.

Example: Pumping Lemma for Context-Free Languages

Consider the following grammar:

$$\begin{array}{lllllll} S & \to & AC \mid AB & \quad & A & \to & a \\ B & \to & b & \quad & C & \to & SB \end{array}$$

Consider the derivation for the string *aaaabbbb*

$$S \Rightarrow AC \Rightarrow aC \Rightarrow aSB \Rightarrow aACB \Rightarrow aaCB \Rightarrow aaSBB \Rightarrow aaABBB$$
$$\Rightarrow aaaBBB \Rightarrow aaabBB \Rightarrow aaabbB \Rightarrow aaabbb$$

Consider the parse tree and the last 2 occurrences of the symbol S.

Then we have $x = a$, $u = a$, $y = ab$, $v = b$, $z = b$.

Example: Pumping Lemma for Context-Free Languages

Lemma: *The language* $\mathcal{L} = \{a^m b^m c^m \mid m > 0\}$ *is not context-free.*

Proof: Let us assume $\mathcal{L}$ is context-free.

Let n be the constant stated by the Pumping lemma.

Let $w = a^n b^n c^n$; we have that $|w| \geqslant n$.

By the PL we know that $w = xuyvz$ such that

$$|uyv| \leqslant n \qquad uv \neq \epsilon \qquad \forall k \geqslant 0.\ xu^k yv^k z \in \mathcal{L}$$

Since $|uyv| \leqslant n$ there is one letter $d \in \{a, b, c\}$ that *does not* occur in uyv.

Since $uv \neq \epsilon$ there is another letter $e \in \{a, b, c\}, e \neq d$ that *does* occur in uv.

Then e has more occurrences than d in xu^2yv^2z and this contradicts the fact that $xu^2yv^2z \in \mathcal{L}$.

Overview of Next Lecture

Sections 7.3–7.4:

- Closure properties of CFL;
- Decision properties of CFL;
- Guest lecture by Andreas Abel:
  Programming Language Technology: Putting Formal Languages to Work.