Finite Automata Theory and Formal Languages

TMV027/DIT321– LP4 2013

Lecture 12
Ana Bove

May 7th 2013

Overview of today's lecture:

- Regular grammars and Chomsky hierarchy;
- Simplifications and Normal Forms for CFL;
- Pumping Lemma for CFL.

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:

- $p \rightarrow aq$ if $\delta(p, a) = q$
- $p \rightarrow \epsilon$ 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 $w \in \mathcal{L}(D)$.

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 $\hat{\delta}(p, av) = \hat{\delta}(\delta(p, a), v) \in F$.

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 $\alpha A\beta \rightarrow \alpha\gamma\beta$. 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 $A \rightarrow Ba$, $A \rightarrow aB$ or $A \rightarrow \epsilon$.

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 $S \Rightarrow^* \alpha X \beta$.

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.

Eliminating Useless Symbols

If we eliminate useless symbols we do not change the language generated by the grammar.

Note: It is important in which order we check these conditions.

Example: Consider the following grammar

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

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

$$S \rightarrow a$$

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

$$S \rightarrow a \qquad A \rightarrow b$$

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.

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 $S \to U$.
W is however 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.

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:

$$
\begin{array}{ll}
S \to aB \mid BC & C \to b \\
A \to aA \mid c \mid aDb & D \to B \\
B \to DB \mid C &
\end{array}
$$

S is reachable.
Hence a, B and C are reachable.
Then b and D are reachable.
However A and c are not reachable.

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

$$
\begin{array}{ll}
S \to aB \mid BC & C \to b \\
B \to DB \mid C & D \to B
\end{array}
$$

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. *Eliminate all non-generating symbols and all productions involving one or more of those symbols;*
2. *In the same way, eliminate now all symbols that are not reachable in the grammar.*

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:

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

The simplified grammar is:

$$
\begin{array}{lcl}
S & \rightarrow & gAe \\
A & \rightarrow & ooC \\
C & \rightarrow & gl
\end{array}
$$

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.

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:

$$S \rightarrow aSb \mid SS \mid \epsilon$$

By eliminating ϵ-productions we obtain

$$S \rightarrow ab \mid aSb \mid S \mid SS$$

Example: Consider the grammar given by the rules:

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

By eliminating ϵ-productions we obtain

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

Eliminating Unit Productions

Definition: A *unit production* is a production of the form $A \rightarrow B$.

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 $A \rightarrow \alpha$ whenever we have $A \rightarrow B$ and $B \rightarrow \alpha$;
2. Construct $G' = (V, T, \mathcal{R}', S)$ where 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:

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

By eliminating unit productions we obtain:

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

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. *Eliminating ϵ-productions;*
2. *Eliminating unit productions;*
3. *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 $A \rightarrow BC$ or $A \rightarrow a$.

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 $A \rightarrow a$ or $A \rightarrow X_1 X_2 \ldots X_k$ for $k > 1$.

If X_i is a terminal introduce a new variable A_i and a new rule $A_i \rightarrow X_i$ (if no such rule exists for X_i).

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

Now, all rules are of the form $B \rightarrow b$ or $B \rightarrow C_1 C_2 \ldots C_k$ with all C_j variables.

Introduce $k - 2$ new variables and break each rule $B \rightarrow C_1 C_2 \ldots C_k$ as

$$B \rightarrow C_1 D_1 \quad D_1 \rightarrow C_2 D_2 \quad \cdots \quad D_{k-2} \rightarrow C_{k-1} C_k$$

Example: Chomsky Normal Form

Consider the grammar given by the rules:

$$S \rightarrow aSb \mid SS \mid ab$$

We first obtain

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

Then we build a grammar in Chomsky Normal Form

$$\begin{array}{lcl} S & \rightarrow & AC \mid SS \mid AB \\ A & \rightarrow & a \\ B & \rightarrow & b \\ C & \rightarrow & SB \end{array}$$

Pumping Lemma for Left Regular Languages

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

If $a_1 a_2 \ldots a_m \in \mathcal{L}(G)$ and $m > n$, 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 such that for every $w \in \mathcal{L}$ with $|w| \geqslant n$, then we can write $w = xuyvz$ such that*

1. $|uyv| \leqslant n$;
2. $uv \neq \epsilon$, *that is, at least one of u and v is not empty;*
3. $\forall k \geqslant 0,\ xu^kyv^kz \in \mathcal{L}$.

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:

$$\begin{array}{lcl@{\qquad\qquad}lcl} S & \rightarrow & AC \mid AB & A & \rightarrow & a \\ B & \rightarrow & b & C & \rightarrow & SB \end{array}$$

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: Assume $\mathcal{L}$ is context-free.

Then we have n as stated in the Pumping lemma.

Consider $w = a^n b^n c^n$. We have that $|w| \geqslant n$.

So we know that $w = xuyvz$ such that

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

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

Since $|uv| > 0$ 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

Guest lecture by *Aarne Ranta*

Using Grammars in Compilers and Translation

and sections 7.3–7.4:

- Closure properties of CFL;
- Decision properties of CFL.