Finite Automata Theory and Formal Languages

TMV027/DIT321– LP4 2018

Lecture 14

Ana Bove

May 14th 2018

Recap: Context-free Grammars

- Simplification of grammars:
  - Elimination of ϵ-productions;
  - Elimination of unit productions;
  - Elimination of useless symbols:
    - Elimination of non-generating symbols;
    - Elimination of non-reachable symbols;
- Chomsky normal forms: rules of the form $A \to a$ or $A \to BC$.

Overview of Today's Lecture

- Regular grammars;
- Chomsky hierarchy;
- Pumping lemma for CFL;
- Closure properties of CFL;
- Decision properties of CFL;

Contributes to the following learning outcome:

- Explain and manipulate the diff. concepts in automata theory and formal lang;
- Understand the power and the limitations of regular lang and context-free lang;
- 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.

Regular Grammars

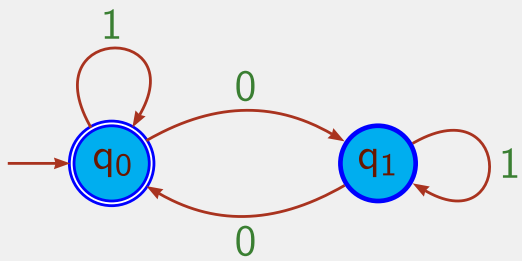
Definition: A grammar where all rules are of the form $A \to aB$ or $A \to \epsilon$ is called *left regular*.

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:



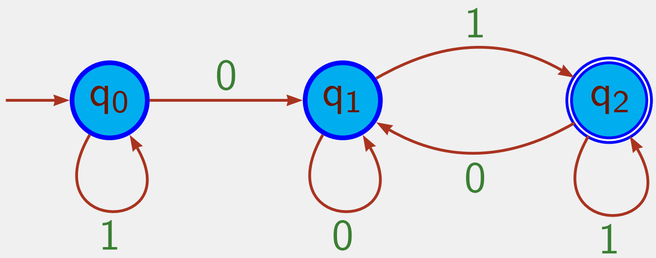
Exercise: What could the left regular grammar be for this language?

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.

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

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

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

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

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

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 mathematical induction on $|w|$ that

- $\forall\, p, q.\; p \Rightarrow^* wq$ iff $\hat{\delta}(p, w) = q$ and
- $\forall\, p.\; p \Rightarrow^* w$ 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.

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

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. Then the Pumping lemma must apply.

Let n be the constant stated by the Pumping lemma.

Let $w = a^n b^n c^n \in \mathcal{L}$; we have that $|w| \geqslant n$.

By the lemma 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 $xu^2yv^2z \in \mathcal{L}$.

Hence $\mathcal{L}$ cannot be a context-free language.

Closure under Union

Theorem: *Let $G_1 = (V_1, T, \mathcal{R}_1, S_1)$ and $G_2 = (V_2, T, \mathcal{R}_2, S_2)$ be CFG. Then $\mathcal{L}(G_1) \cup \mathcal{L}(G_2)$ is a context-free language.*

Proof: Let us assume $V_1 \cap V_2 = \emptyset$ (easy to get via renaming).

Let S be a fresh variable.

We construct $G = (V_1 \cup V_2 \cup \{S\}, T, \mathcal{R}_1 \cup \mathcal{R}_2 \cup \{S \to S_1 \mid S_2\}, S)$.

It is now easy to see that $\mathcal{L}(G) = \mathcal{L}(G_1) \cup \mathcal{L}(G_2)$ since a derivation will have the form

$$S \Rightarrow S_1 \Rightarrow^* w \text{ if } w \in \mathcal{L}(G_1)$$

or

$$S \Rightarrow S_2 \Rightarrow^* w \text{ if } w \in \mathcal{L}(G_2)$$

Closure under Concatenation

Theorem: *Let $G_1 = (V_1, T, \mathcal{R}_1, S_1)$ and $G_2 = (V_2, T, \mathcal{R}_2, S_2)$ be CFG. Then $\mathcal{L}(G_1)\mathcal{L}(G_2)$ is a context-free language.*

Proof: Again, let us assume $V_1 \cap V_2 = \emptyset$.

Let S be a fresh variable.

We construct $G = (V_1 \cup V_2 \cup \{S\}, T, \mathcal{R}_1 \cup \mathcal{R}_2 \cup \{S \to S_1S_2\}, S)$.

It is now easy to see that $\mathcal{L}(G) = \mathcal{L}(G_1)\mathcal{L}(G_2)$ since a derivation will have the form

$$S \Rightarrow S_1S_2 \Rightarrow^* uv$$

with

$$S_1 \Rightarrow^* u \text{ and } S_2 \Rightarrow^* v$$

for $u \in \mathcal{L}(G_1)$ and $v \in \mathcal{L}(G_2)$.

Closure under Closure

Theorem: *Let $G = (V, T, \mathcal{R}, S)$ be a CFG. Then $\mathcal{L}(G)^+$ and $\mathcal{L}(G)^*$ are context-free languages.*

Proof: Let S' be a fresh variable.

We construct $G+ = (V \cup \{S'\}, T, \mathcal{R} \cup \{S' \to S \mid SS'\}, S')$ and
$G* = (V \cup \{S'\}, T, \mathcal{R} \cup \{S' \to \epsilon \mid SS'\}, S')$.

It is easy to see that $S' \Rightarrow \epsilon$ in $G*$.

Also that $S' \Rightarrow^* S \Rightarrow^* w$ if $w \in \mathcal{L}(G)$ is a valid derivation both in $G+$ and in $G*$.

In addition, if $w_1, \ldots, w_k \in \mathcal{L}(G)$, it is easy to see that the derivation

$$\begin{aligned} S' &\Rightarrow SS' \Rightarrow^* w_1S' \Rightarrow w_1SS' \Rightarrow^* w_1w_2S' \Rightarrow^* \ldots \\ &\Rightarrow^* w_1w_2\ldots w_{k-1}S' \Rightarrow^* w_1w_2\ldots w_{k-1}S \Rightarrow^* w_1w_2\ldots w_{k-1}w_k \end{aligned}$$

is a valid derivation both in $G+$ and in $G*$.

Non Closure under Intersection

Example: Consider the following languages over $\{a, b, c\}$:

$$\mathcal{L}_1 = \{a^k b^k c^m \mid k, m > 0\}$$

$$\mathcal{L}_2 = \{a^m b^k c^k \mid k, m > 0\}$$

It is easy to give CFG generating both $\mathcal{L}_1$ and $\mathcal{L}_2$, hence $\mathcal{L}_1$ and $\mathcal{L}_2$ are CFL.

However $\mathcal{L}_1 \cap \mathcal{L}_2 = \{a^k b^k c^k \mid k > 0\}$ is not a CFL (see slide 12).

Closure under Intersection with Regular Language

Theorem: *If $\mathcal{L}$ is a CFL and $\mathcal{P}$ is a RL then $\mathcal{L} \cap \mathcal{P}$ is a CFL.*

Proof: See Theorem 7.27 in the book.
(It uses *push-down automata* which we have not seen.)

Example: Consider the following language over $\Sigma = \{0, 1\}$:

$$\mathcal{L} = \{ww \mid w \in \Sigma^*\}$$

Is $\mathcal{L}$ a regular language?

Consider $\mathcal{L}' = \mathcal{L} \cap \mathcal{L}(0^*1^*0^*1^*) = \{0^n1^m0^n1^m \mid n, m \geqslant 0\}$.

$\mathcal{L}'$ is not a CFL (see additional exercise 4 in exercises for CFL).

Hence $\mathcal{L}$ cannot be a CFL since $\mathcal{L}(0^*1^*0^*1^*)$ is a RL.

Non Closure under Complement

Theorem: *CFL are not closed under complement.*

Proof: Notice that
$$\mathcal{L}_1 \cap \mathcal{L}_2 = \overline{\overline{\mathcal{L}_1} \cup \overline{\mathcal{L}_2}}$$

If CFL are closed under complement then they should be closed under intersection (since they are closed under union).

Then CFL are in general not closed under complement.

Closure under Difference?

Theorem: *CFL are not closed under difference.*

Proof: Let $\mathcal{L}$ be a CFL over Σ.

It is easy to give a CFG that generates Σ^*.

Observe that $\overline{\mathcal{L}} = \Sigma^* - \mathcal{L}$.

Then if CFL are closed under difference they would also be closed under complement.

Theorem: *If $\mathcal{L}$ is a CFL and $\mathcal{P}$ is a RL then $\mathcal{L} - \mathcal{P}$ is a CFL.*

Proof: Observe that $\overline{\mathcal{P}}$ is a RL and $\mathcal{L} - \mathcal{P} = \mathcal{L} \cap \overline{\mathcal{P}}$.

Closure under Reversal and Prefix

Theorem: *If $\mathcal{L}$ is a CFL then so is $\mathcal{L}^{\mathrm{r}} = \{\mathrm{rev}(w) \mid w \in \mathcal{L}\}$.*

Proof: Given a CFG $G = (V, T, \mathcal{R}, S)$ for $\mathcal{L}$ we construct the grammar $G^{\mathrm{r}} = (V, T, \mathcal{R}^{\mathrm{r}}, S)$ where $\mathcal{R}^{\mathrm{r}}$ is such that, for each rule $A \to \alpha$ in $\mathcal{R}$, then $A \to \mathrm{rev}(\alpha)$ is in $\mathcal{R}^{\mathrm{r}}$.

One should show by induction on the length of the derivations in G and G^{r} that $\mathcal{L}(G^{\mathrm{r}}) = \mathcal{L}^{\mathrm{r}}$.

Theorem: *If $\mathcal{L}$ is a CFL then so is* $\mathrm{Prefix}(\mathcal{L})$.

Proof: For closure under prefix see exercise 7.3.1 part a) in the book.

Decision Properties of Context-Free Languages

Very little can be answered when it comes to CFL.

The major tests we can answer are whether:

- The language is empty;

  (See the algorithm that tests for generating symbols in slide 4 lecture 13: if $\mathcal{L}$ is a CFL given by a grammar with start variable S, then $\mathcal{L}$ is empty if S is not generating.)

- A certain string belongs to the language.

Testing Membership in a Context-Free Language

Checking if $w \in \mathcal{L}(G)$, where $|w| = n$, by trying all productions may be exponential on n.

An efficient way to check for membership in a CFL is based on the idea of *dynamic programming*.

(Method for solving complex problems by breaking them down into simpler problems, applicable mainly to problems where many of their subproblems are really the same; not to be confused with the *divide and conquer* strategy.)

The algorithm is called the *CYK algorithm* after the 3 people who independently discovered the idea: Cock, Younger and Kasami.

It is a $O(n^3)$ algorithm.

Example: CYK Algorithm

Consider the following grammar in CNF given by the rules

$$S \to AB \mid BA \qquad A \to AS \mid a \qquad B \to BS \mid b$$

and starting symbol S.

Does *abba* belong to the language generated by the grammar?

We fill the corresponding table:

$\{S\}_{abba}$			
$\emptyset_{abb}$	$\{B\}_{bba}$		
$\{S\}_{ab}$	$\emptyset_{bb}$	$\{S\}_{ba}$	
$\{A\}_{a}$	$\{B\}_{b}$	$\{B\}_{b}$	$\{A\}_{a}$
a	b	b	a

Then $S \Rightarrow^* abba$.

The CYK Algorithm

Let $G = (V, T, \mathcal{R}, S)$ be a CFG in CNF and $w = a_1 a_2 \dots a_n \in T^*$.

Does $w \in \mathcal{L}(G)$?

In the CYK algorithm we fill a table

$$
\begin{array}{|cccccc}
V_{1n} & & & & & \\
V_{1(n-1)} & V_{2n} & & & & \\
\vdots & \vdots & & & & \\
V_{12} & V_{23} & V_{34} & \dots & V_{(n-1)n} & \\
V_{11} & V_{22} & V_{33} & \dots & V_{(n-1)(n-1)} & V_{nn} \\
\hline
a_1 & a_2 & a_3 & \dots & a_{n-1} & a_n
\end{array}
$$

where $V_{ij} \subseteq V$ is the set of A's such that $A \Rightarrow^* a_i a_{i+1} \dots a_j$.

We want to know if $S \in V_{1n}$, hence $S \Rightarrow^* a_1 a_2 \dots a_n$.

CYK Algorithm: Observations

- Each row corresponds to the substrings of a certain length:
  - bottom row is length 1,
  - second from bottom is length 2,
  - ...
  - top row is length n;
- We work row by row upwards and compute the V_{ij}'s;
- In the bottom row we have $i = j$, that is, ways of generating a_i;
- V_{ij} is the set of variables generating $a_i a_{i+1} \dots a_j$ of length $j - i + 1$ (hence, V_{ij} is in row $j - i + 1$);
- In the rows below that of V_{ij} we have all ways to generate shorter strings, including all prefixes and suffixes of $a_i a_{i+1} \dots a_j$.

CYK Algorithm: Table Filling

We compute V_{ij} as follows (remember we work with a CFG in CNF):

Base case: First row in the table. Here $i = j$.
Then $V_{ii} = \{A \mid A \to a_i \in \mathcal{R}\}$.

Recursive step: To compute V_{ij} for $i < j$ we have all V_{pq}'s in rows below.

The length of the string is at least 2, so $A \Rightarrow^* a_i a_{i+1} \ldots a_j$ starts with $A \Rightarrow BC$ such that

$$B \Rightarrow^* a_i a_{i+1} \ldots a_k \text{ and}$$
$$C \Rightarrow^* a_{k+1} \ldots a_j \text{ for some } k.$$

So $A \in V_{ij}$ if $\exists k, i \leqslant k < j$ such that

- $B \in V_{ik}$ and $C \in V_{(k+1)j}$;
- $A \to BC \in \mathcal{R}$.

We need to look at
$(V_{ii}, V_{(i+1)j}), (V_{i(i+1)}, V_{(i+2)j}), \ldots, (V_{i(j-1)}, V_{jj})$.

Example: CYK Algorithm

Consider the grammar given by the rules

$$S \to XY \qquad X \to XA \mid a \mid b$$
$$Y \to AY \mid a \qquad A \to a$$

and starting symbol S.

Does *babaa* belong to the language generated by the grammar?

We fill the corresponding table:

$\emptyset_{babaa}$				
$\emptyset_{baba}$	$\emptyset_{abaa}$			
$\emptyset_{bab}$	$\emptyset_{aba}$	$\{S,X\}_{baa}$		
$\{S,X\}_{ba}$	$\emptyset_{ab}$	$\{S,X\}_{ba}$	$\{S,X,Y\}_{aa}$	
$\{X\}_{b}$	$\{A,X,Y\}_{a}$	$\{X\}_{b}$	$\{A,X,Y\}_{a}$	$\{A,X,Y\}_{a}$
b	a	b	a	a

$S \notin V_{15}$ then $S \not\Rightarrow^* babaa$.

Undecidable Problems for Context-Free Grammars/Languages

Definition: An *undecidable problem* is a decision problem for which it is impossible to construct a single algorithm that always leads to a correct yes-or-no answer.

Example: Halting problem: does this program terminate?

The following problems are undecidable:

- Is the CFG G ambiguous?
- Is the CFL $\mathcal{L}$ inherently ambiguous?
- If $\mathcal{L}(G_1)$ and $\mathcal{L}(G_2)$ are CFL, is $\mathcal{L}(G_1) \cap \mathcal{L}(G_2) = \emptyset$?
- If $\mathcal{L}(G_1)$ and $\mathcal{L}(G_2)$ are CFL, is $\mathcal{L}(G_1) = \mathcal{L}(G_2)$? is $\mathcal{L}(G_1) \subseteq \mathcal{L}(G_2)$?
- If $\mathcal{L}(G)$ is a CFL and $\mathcal{P}$ a RL, is $\mathcal{P} = \mathcal{L}(G)$? is $\mathcal{P} \subseteq \mathcal{L}(G)$?
- If $\mathcal{L}(G)$ is a CFL over Σ, is $\mathcal{L}(G) = \Sigma^*$?

Overview of Next Lecture

Sections 6, 8 (just a bit of both):

- Push-down automata;
- Turing machines.