Finite Automata and Formal Languages

TMV026/DIT321– LP4 2012

Lecture 14
Ana Bove

May 8th 2012

Overview of today's lecture:

- Closure Properties for Context-Free Languages
- Decision Properties for Context-Free Languages

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

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

or

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

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

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

$$S \Rightarrow S_1 S_2 \Rightarrow^* u_1 u_2$$

with

$$S_1 \Rightarrow^* u_1 \text{ and } S_2 \Rightarrow^* u_2$$

for $u_1 \in \mathcal{L}(G_1)$ and $u_2 \in \mathcal{L}(G_2)$.

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 $G*$.
It is also easy to see that $S' \Rightarrow^* S \Rightarrow^* u$ if $u \in \mathcal{L}(G)$ is a valid derivation both in $G+$ and in $G*$.

In addition, if $u_1, \ldots, u_k \in \mathcal{L}(G)$, it is easy to see that the derivation

$$\begin{aligned} S' &\Rightarrow SS' \Rightarrow^* u_1 S' \Rightarrow u_1 SS' \Rightarrow^* u_1 u_2 S' \Rightarrow^* \ldots \\ &\Rightarrow^* u_1 u_2 \ldots u_{k-1} S' \Rightarrow^* u_1 u_2 \ldots u_{k-1} S \Rightarrow^* u_1 u_2 \ldots u_{k-1} u_k \end{aligned}$$

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 22 lecture 13).

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: As an application consider the following language over $\Sigma = \{0, 1\}$:

$$\mathcal{L} = \{uu \mid u \in \Sigma^*\}$$

Consider now $\mathcal{L}' = \mathcal{L} \cap \mathcal{L}(0^*1^*0^*1^*) = \{0^n1^m0^n1^m \mid n, m \geqslant 0\}$.
$\mathcal{L}'$ is not a CFL (see exercise 6 on exercises for week 7).

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 $\mathcal{L}^r = \{\text{rev}(w) \mid w \in \mathcal{L}\}$.*

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

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

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

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

Closure under Homomorphisms and Inverse Homomorphisms

Theorem: *CFL are closed under homomorphism and inverse homomorphisms.*

Proof: For the closure under homomorphisms see Theorem 7.24 point 4 in the book.
(It uses the notion of *substitution* which we have not seen.)

For the closure under inverse homomorphisms see Theorem 7.30 in the book.
(It uses *push-down automata* which we have not seen.)

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: we have already seen an algorithm to test this.

  (See the algorithm that tests for generating symbols in slide 3 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 belong to the language.

Testing Membership in a Context-Free Language

To check 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 of 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.

The CYK Algorithm

Let $G = (V, T, \mathcal{R}, S)$ be a CFG in CNF and $w = a_1 a_2 \dots a_n \in T^*$ the word we want to check for membership in $\mathcal{L}(G)$.

In the CYK algorithm we fill a table

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

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

We work row by row upwards and compute the V_{ij}'s.

Remember that V_{ij} is the set of variables generating $a_i a_{i+1} \dots a_j$ of length $j - i + 1$.

Then, 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.

Hence, V_{ij} is in row $j - i + 1$.

In the bottom row we have $i = j$, that is, ways of generating the string a_i.

When $i < j$, in the row 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

Remember we work with a CFG in CNF.
We compute V_{ij} as follows:

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

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

The length of the string is at least 2, so $A \Rightarrow^* a_i a_{i+1} \dots a_j$ starts with $A \Rightarrow BC$ such that $B \Rightarrow^* a_i a_{i+1} \dots a_k$ and $C \Rightarrow^* a_{k+1} \dots a_j$ 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
$(V_{ii}, V_{(i+1)j}), (V_{i(i+1)}, V_{(i+2)j}), \dots, (V_{i(j-1)}, V_{jj})$.

CYK Algorithm: Example

Consider the grammar given by the rules

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

Does *abba* belong to the grammar?

We fill the corresponding table:

$\{S\}$			
$\emptyset$	$\{B\}$		
$\{S\}$	$\emptyset$	$\{S\}$	
$\{A\}$	$\{B\}$	$\{B\}$	$\{A\}$
a	b	b	a

$S \in V_{14}$ then $S \Rightarrow^* abba$.

CYK Algorithm: Example

Consider the grammar given by the rules

$$\begin{array}{ll} S \to XY & X \to XA \mid a \mid b \\ Y \to AY \mid a & A \to a \end{array}$$

Does *babaa* belong to the grammar?

We fill the corresponding table:

$\emptyset$				
$\emptyset$	$\emptyset$			
$\emptyset$	$\emptyset$	$\{S, X\}$		
$\{S, X\}$	$\emptyset$	$\{S, X\}$	$\{S, X, Y\}$	
$\{X\}$	$\{A, X, Y\}$	$\{X\}$	$\{A, X, Y\}$	$\{A, X, Y\}$
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 $\mathcal{L}_1$ and $\mathcal{L}_2$ are CFL, is $\mathcal{L}_1 \cap \mathcal{L}_2 = \emptyset$?
- If $\mathcal{L}_1$ and $\mathcal{L}_2$ are CFL, is $\mathcal{L}_1 = \mathcal{L}_2$? is $\mathcal{L}_1 \subseteq \mathcal{L}_2$?
- If $\mathcal{L}$ is a CFL and $\mathcal{P}$ a RL, is $\mathcal{P} = \mathcal{L}$? is $\mathcal{P} \subseteq \mathcal{L}$?
- If $\mathcal{L}$ is a CFL over Σ, is $\mathcal{L} = \Sigma^*$?

LL(k) Parsers and Grammars

Definition: An *LL parser* is a top-down parser for a subset of the context-free grammars. It parses the input from left to right, and constructs a leftmost derivation of the sentence.

The class of grammars which are parsable in this way is known as the *LL grammars*.

An LL parser is called an *LL(k) parser* if it uses k tokens of look-ahead when parsing a sentence. If such a parser exists for a certain grammar then it is called an *LL(k) grammar*.

LL(1) Grammars

Then a grammar is LL(1) if to construct the leftmost derivation we can decide what is the production to use next just by looking only at the first symbol of the string to be parsed.

Example: $S \rightarrow +SS \mid a \mid b$ is LL(1).

Example: $S \rightarrow F \mid (S + F)$ $F \rightarrow a$ is also LL(1).

Any LL(1) grammar is unambiguous: by definition there is at most one leftmost derivation for any string.

Any regular grammar is LL(1) iff it corresponds to a DFA.

There are algorithms to decide if a grammar is LL(1).

Undecidable Problems

To prove that a certain problem P is undecidable one usually *reduces* an already known undecidable problem U to the problem P: instances of U become instances of P.

(Can be seen like one "transforms" U so it "becomes" P).

That is, $w \in U$ iff $w' \in P$ for certain w and w'.

Then, a solution to P would serve as a solution to U.

However, we know there are no solutions to U since U is known to be undecidable.

Then we have a contradiction.

Post Correspondence Problem

It is an undecidable decision problem introduced by Emil Post in 1946. (See Section 9.4 in the book.)

> *Given words $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ in $\{0,1\}^*$, is it possible to find $i_1, \ldots, i_k$ such that $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$?*

Example: Given $u_1 = 1, u_2 = 10, u_3 = 001, v_1 = 011, v_2 = 11, v_3 = 00$ we have that $u_3u_2u_3u_1 = v_3v_2v_3v_1 = 001100011$.

Post Correspondence Problem and Context-Free Languages

Let $\Sigma = \{0, 1, a_1, \ldots, a_n\}$.

To the sequence $u_1, \ldots, u_n$ we associate the grammar G_u with rules

$$A \to u_1 a_1 \mid \ldots \mid u_n a_n \mid u_1 A a_1 \mid \ldots \mid u_n A a_n$$

To the sequence $v_1, \ldots, v_n$ we associate the grammar G_v with rules

$$B \to v_1 a_1 \mid \ldots \mid v_n a_n \mid v_1 B a_1 \mid \ldots \mid v_n B a_n$$

The grammars are not ambiguous. (Can you see why?)

Let G be the grammar with all the productions of G_u and G_v plus $S \to A \mid B$.

Post Correspondence Problem and Context-Free Languages

Theorem: *The grammar G is ambiguous iff the Post correspondence problem for $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ has a solution.*

Theorem: *$\mathcal{L}(G_u) \cap \mathcal{L}(G_v) \neq \emptyset$ iff the Post correspondence problem for $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ has a solution.*

See Section 9.5 in the book for proofs that most of the statements in slide 17 are undecidable using the Post correspondence problem.