Computational Models — Lecture 4[1]

Handout Mode

Ronitt Rubinfeld and Iftach Haitner.

Tel Aviv University.

March 21/23, 2016

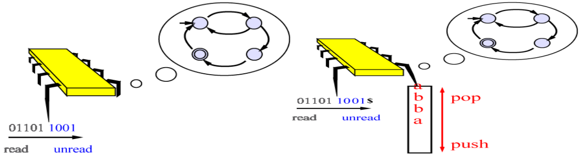
[1] Based on frames by Benny Chor, Tel Aviv University, modifying frames by Maurice Herlihy, Brown University.

Talk Outline

- Context Free Grammars/Languages (CFG/CFL)
- Algorithmic issues for CFL

Next two weeks:

- Pumping Lemma for context free languages
- Push Down Automata (PDA)
- Equivalence of CFGs and PDAs



- Sipser's book, 2.1, 2.2 & 2.3

Short Overview of the Course

So far we saw

- finite automata,
- regular languages,
- regular expressions,
- Myhill-Nerode theorem
- pumping lemma for regular languages.

We now introduce stronger machines and languages with more expressive power:

- pushdown automata,
- context-free languages,
- context-free grammars,
- pumping lemma for context-free languages.

Context Free Grammars (CFG)

An example of a context free grammar, G_1:

- $A \to 0A1$
- $A \to B$
- $B \to \#$

Terminology:

- Each line is a substitution rule or production.
- Each rule has the form: symbol $\to$ string.
  The left-hand symbol is a variable (usually upper-case).
- A string consists of variables and terminals.
- One variable is the start variable (lhs of top rule). In this case, it is A.

Rules for Generating Strings

- Write down the start variable.
- Pick a variable written down in current string and a derivation that starts with that variable.
- Replace that variable with right-hand side of that derivation.
- Repeat until no variables remain.
- Return final string (concatenation of terminals).

Process is inherently non deterministic.

Example

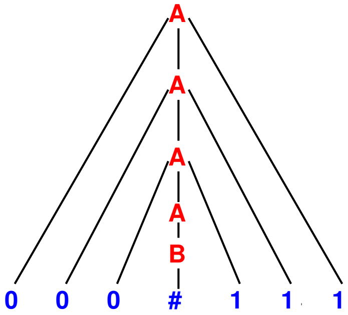
Grammar G_1:

- $A \to 0A1$
- $A \to B$
- $B \to \#$

Derivation with G_1:

$$\begin{aligned} \mathbf{A} &\to 0A1 \\ &\to 00A11 \\ &\to 000A111 \\ &\to 000B111 \\ &\to \mathbf{000\#111} \end{aligned}$$

Parsing Trees



Indifferent to derivation order.

Question 1

What strings can be generated in this way from the grammar G_1?

Answer: Exactly those of the form $0^n\#1^n$ ($n \geq 0$).

Context-Free Languages (CFL)

The language generated in this way is called the language of the grammar.

For example, $\mathcal{L}(G_1) = \{0^n\#1^n : n \geq 0\}$.

Any language generated by a context-free grammar is called a context-free language.

A Useful Abbreviation

Rules with same variable on left hand side

$$
\begin{aligned}
A &\rightarrow 0A1 \\
A &\rightarrow B
\end{aligned}
$$

are written as:

$$A \rightarrow 0A1 \mid B$$

English-like Sentences

A grammar G_2 to describe a few English sentences:

$$
\begin{aligned}
< \text{SENTENCE} > &\rightarrow \mathcal{NP} < \text{VERB} > \\
\mathcal{NP} &\rightarrow < \text{ARTICLE} >< \text{NOUN} > \\
< \text{NOUN} > &\rightarrow \mathsf{boy} \mid \mathsf{girl} \mid \mathsf{flower} \\
< \text{ARTICLE} > &\rightarrow \mathsf{a} \mid \mathsf{the} \\
< \text{VERB} > &\rightarrow \mathsf{touches} \mid \mathsf{likes} \mid \mathsf{sees}
\end{aligned}
$$

Deriving English-like Sentences

A specific derivation in G_2:

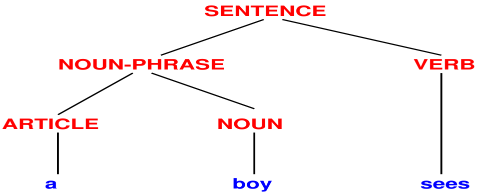
$$\begin{aligned}
< \text{SENTENCE} > \; &\rightarrow \; \mathcal{NP} < \text{VERB} > \\
&\rightarrow \; < \text{ARTICLE} >< \text{NOUN} >< \text{VERB} > \\
&\rightarrow \; \mathsf{a} \;\; < \text{NOUN} >< \text{VERB} > \\
&\rightarrow \; \mathsf{a\ boy} \;\; < \text{VERB} > \\
&\rightarrow \; \mathsf{a\ boy\ sees}
\end{aligned}$$

More strings generated by G_2:

a flower sees
the girl touches

Derivation and Parse Tree

$$
\begin{aligned}
<\textbf{SENTENCE}> \quad &\rightarrow \quad \mathcal{NP} <\textbf{VERB}> \\
&\rightarrow \quad <\textbf{ARTICLE}><\textbf{NOUN}><\textbf{VERB}> \\
&\rightarrow \quad \textsf{\textbf{a}} \ <\textbf{NOUN}><\textbf{VERB}> \\
&\rightarrow \quad \textsf{\textbf{a boy}} \ <\textbf{VERB}> \\
&\rightarrow \quad \textsf{\textbf{a boy sees}}
\end{aligned}
$$



Formal Definition

A context-free grammar is a 4-tuple (V, Σ, R, S), where

- V is a finite set of variables
- Σ is a finite set of terminals
- R is a finite set of rules: each rule is a variable and a finite string of variables and terminals.
- S is the start symbol.
- If u and v are strings of variables and terminals, and $A \to w$ is a rule of the grammar, then uAv yields uwv, written $uAv \to uwv$.
- $u \overset{*}{\to} v$ if $u = v$, or $u \to u_1 \to \ldots \to u_k \to v$ for some sequence $u_1, u_2, \ldots, u_k$

Definition 2

The language of the grammar G, denoted $\mathcal{L}(G)$, is $\{w \in \Sigma^* : S \overset{*}{\to} w\}$

where $\overset{*}{\to}$ is determined by G.

Example 1

$G_3 = (\{S\}, \{a, b\}, R, S)$.

R (Rules): $S \to aSb \mid SS \mid \varepsilon$

Some words in the language: *aabb*, *aababb*.

Question 3

What is this language?

Hint: Think of parentheses: i.e., a is "(" and b is ")". (()), (()())

Using larger alphabet (i.e., more terminals), ([]()), represent well formed programs with many kinds of nested loops, "if then/else" statements.

Example 2

$G_4 = (\{S\}, \{a, b\}, R, S)$.

R (Rules): $S \to aSa \mid bSb \mid \varepsilon$

Some words in the language: *abba*,*aabaabaa*.

Question 4

What is this language?

$\mathcal{L}(G_4) = \{ww^R \colon w \in \{a, b\}^*\}$ (almost but not quite the set of palindromes)

Proving $\mathcal{L}(G_4) = \{ww^R \colon w \in \{a,b\}^*\}$

What do we need to show?

1. If $z = ww^R$ then z is generated by G_4.
2. If z generated by G_4 then $z = ww^R$.

Part 1:
Proof by induction on the length of z.
Base: $|z| = 0$, then $S \to \varepsilon$. Hence $\varepsilon \in \mathcal{L}(G_4)$.
Inductive claim:
Let $|z| = 2k$.
Let $z = ww^R$ and $w = \sigma w'$. Then $z = \sigma w'(w')^R\sigma$.
Consider the derivation $S \to \sigma S\sigma$.
By the induction hypothesis $S \xrightarrow{*} w'(w')^R$, since $|w'(w')^R| = 2k - 2 < |z|$.
Therefore, G_5 generates $S \to \sigma S\sigma \xrightarrow{*} \sigma w'(w')^R\sigma = z$.
Hence $z \in \mathcal{L}(G_4)$.

Proving $\mathcal{L}(G_4) = \{ww^R \colon w \in \{a,b\}^*\}$

Part 2:

Assume that $S \xrightarrow{*} z$.

We prove by induction on the number of derivations that $z = ww^R$.

Base: If we have a single derivation, then the only possible derivation is $S \to \varepsilon$, and we are done.

Inductive claim:

Assume $S \xrightarrow{*} z$ in k derivations.

Assume we have $S \to \sigma S \sigma$ as the first derivation.

Then by the inductive hypothesis, $S \to w'(w')^R$.

Therefore, $z = \sigma w'(w')^R \sigma$.

Example 3

$G_5 = (\{S, A, B\}, \{a, b\}, R, S)$

R (Rules):
$S \to aB \mid bA \mid \varepsilon$
$A \to a \mid aS \mid bAA$
$B \to b \mid bS \mid aBB$

Some words in the language: *aababb*, *baabba*.

Question 5

What is this language?

$\mathcal{L}(G_5) = \{w \in \{a, b\}^* : \#_a(w) = \#_b(w)\}$

$\#_x(w)$ — number of occurrences of x in w

Proving $\mathcal{L}(G_5) = \{w \in \{a,b\}^* : \#_a(w) = \#_b(w)\}$

What do we need to show?

1. If w generated by G_5 then $\#_a(w) = \#_b(w)$.
2. If $\#_a(w) = \#_b(w)$ then w is generated by G_5.

Claim 6

If $S \xrightarrow{*} w$, then $\#_a(w) + \#_A(w) = \#_b(w) + \#_B(w)$.

Claim 7

For $w \in \{a,b\}^*$ let $k = k(w) = \#_a(w) - \#_b(w)$. Then:

1. If $k = 0$, then $S \xrightarrow{*} wS$
2. If $k > 0$, then $S \xrightarrow{*} wB^{|k|}$
3. If $k < 0$, then $S \xrightarrow{*} wA^{|k|}$

Are we done?! How we prove these claims?

Proving Claim 7

Proof by induction on $|w|$:

- Basis: $w = \epsilon$ then $k(w) = 0$. Since $S \xrightarrow{*} S$, then $S \xrightarrow{*} \epsilon S$
- Induction step: for $w \in \{a, b\}^n$ write $w = w'\sigma$ with $|w'| = n - 1$

  Suppose:

  1. $k' = k(w') = (\#_a(w') - \#_b(w')) > 0$
  2. $\sigma = a$

  Note that if both are the case, we have $k' = k - 1$

  By the induction hypothesis $S \xrightarrow{*} w' B^{k'}$

  Since $B \rightarrow aBB$, then
  $S \xrightarrow{*} w' B^{k'} = w' BB^{k'-1} \xrightarrow{*} w'(aBB)B^{k'-1} = wB^{k'+1} = wB^{k(w)}$

To complete the proof, need to show for $\sigma = b$ and for $k' \leq 0$, $k' = 0$ (6 cases in all).

Designing Context-Free Grammars

No recipe in general, but few rules-of-thumb

- If CFG is the union of several CFGs, rename variables (not terminals) so they are disjoint, and add new rule $S \rightarrow S_1 \mid S_2 \mid \ldots \mid S_i$.
- For languages (like $\{0^n\#1^n : n \geq 0\}$), with linked substrings, a rule of form $R \rightarrow uRv$ is helpful to force desired relation between substrings.
- For a regular language, grammar "follows" a DFA for the language (see next frame).

How expressive are CFG's

Are they more expressive or less expressive than regular languages?

CFG for Regular Languages

Given a DFA : $M = (Q, \Sigma, \delta, q_0, F)$

CFG G for $\mathcal{L}(M)$:

1. Let R_0 be the starting variable
2. Add rule $R_i \to aR_j$, for any $q_i, q_j \in Q$ and $a \in \Sigma$ with $\delta(q_i, a) = q_j$
3. Add rule $R_i \to \varepsilon$, for any $q_i \in F$

Claim 8

$\mathcal{L}(G) = \mathcal{L}(M)$

Proof?

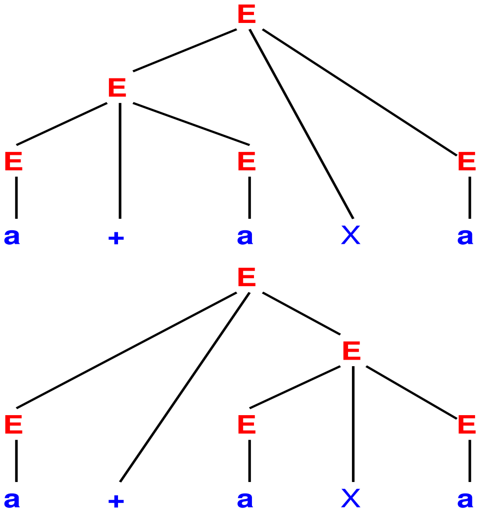
Claim 9

$R_0 \xrightarrow{*} wR_j$ iff $M(w) = q_j$.

Proof? Next class we'll see alternative proof via “Push-Down Automata"

Ambiguity in CFLs

Grammar G: $E \to E + E \mid E \times E \mid (E) \mid a$



Arithmetic Example

Consider the grammar $G' = (V, \Sigma, R, E)$, where

- $V = \{E, T, F\}$
- $\Sigma = \{a, +, \times, (,)\}$
- Rules: $\begin{array}{lcl} E & \to & E + T \mid T \\ T & \to & T \times F \mid F \\ F & \to & (E) \mid a \end{array}$

Claim 10

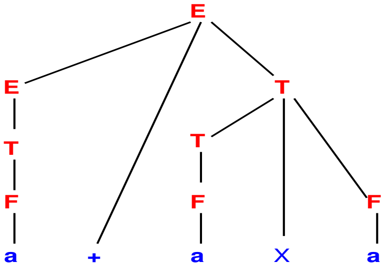
$\mathcal{L}(G') = \mathcal{L}(G)$

Proof? but G' is not ambiguous.

Parsing Tree of G' for $a + a \times a$

G':

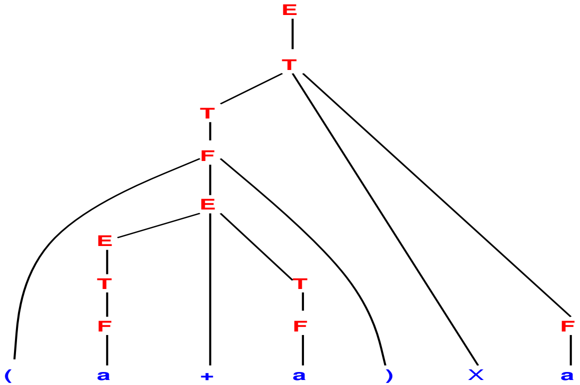
$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T \times F \mid F \\ F &\rightarrow (E) \mid a \end{aligned}$$



Parsing Tree of G' for $(a + a) \times a$

G':

$$
\begin{aligned}
E &\rightarrow E + T \mid T \\
T &\rightarrow T \times F \mid F \\
F &\rightarrow (E) \mid a
\end{aligned}
$$



Ambiguity

Definition 11

A string w is derived ambiguously from grammar G, if w has two or more different parse trees that generate it from G. A CFG is ambiguous, if it ambiguously derives a string.

- Ambiguity is usually not only a syntactic notion but also semantic, implying multiple meanings for the same string. Think of $a + a \times a$ from last grammar.
- It is sometime possible to eliminate ambiguity by finding a different context free grammar generating the same language. This is true for the arithmetic expressions grammar.

  Some languages are inherently ambiguous.

  Example: $\{1^i 2^j 3^k : \ i = j \vee j = k\}$

  Proof? Book!

Part I

Checking Membership

Checking Membership in a CFL

Challenge

Given a CFG G and a string w, decide whether $w \in \mathcal{L}(G)$?

Initial Idea: Design an algorithm that tries all derivations.

Problem: If G does not generate w, we'll never stop.

Possible solution: Use special grammars that are:

- just as expressive!
- better for checking membership.

Chomsky Normal Form (CNF)

A simplified, canonical form of context free grammars.
$G = (V, \Sigma, R, S)$ is in a CNF, if every rule in in R has one of the following forms:

$$\begin{array}{ll} A \to a, & A \in V \;\; \wedge \;\; a \in \Sigma \\ A \to BC, & A \in V \;\; \wedge \;\; B, C \in V \setminus \{S\} \\ S \to \varepsilon. & \end{array}$$

Simpler to analyze: each derivation adds (at most) a single terminal, S only appears once, ε appears only at the empty word

CNF: Theorem

Theorem 12

Any context-free language is generated by a context-free grammar in Chomsky Normal Form.

Proof Idea: [Next time] Give algorithms which does the following:

- Add new start symbol S_0.
- Eliminate all ε rules of the form $A \to \varepsilon$.
- Eliminate all "unit" rules of the form $A \to B$.
- Patch up rules so that grammar generates the same language.
- Convert remaining "long rules" to proper form.

CNF: Bounded Derivation Length

Lemma 13

For a CNF grammar G and $w \in \mathcal{L}(G)$ with $|w| = n \geq 1$, it holds that w has a derivation of length $2n - 1$.

Proof? consider the parsing tree for w – degree at most 2 and n leaves.

Advantage: Easier to check whether $w \in \mathcal{L}(G)$ – only need to try all derivations of length $2n - 1$. This is finite, but there are a lot of these!

Checking Membership for Grammars in CNF Form

Given a CNF grammar $G = (V, \Sigma, R, S)$, we build a function Derive(A, x) that returns TRUE iff $A \xrightarrow{*} x$.

Algorithm 14 (Derive(A, x))

- If $x = \varepsilon$: if $A \to \varepsilon \in R$ (i.e., $A = S$) return TRUE, otherwise return FALSE.
- If $|x| = 1$: if $A \to x \in R$ return TRUE, otherwise return FALSE.
- For each $A \to BC$ and each partition $x = x_1 x_2$:
  - Call Derive(B, x_1) and Derive(C, x_2).
  - Return TRUE if *both* return TRUE.
- Return FALSE.

Test whether $w \in \mathcal{L}(G)$ by calling Derive(S, w)

Correctness?

- Procedure Derive can also output a parse tree for w
- Have we critically used that G is in CNF?

Time Complexity of Derive

What is the time complexity $T\colon \mathbb{N} \mapsto \mathbb{N}$ of Derive?

- Each recursive call tests $|R|$ rules and n partitions.
- $T(n) \leq |R| \cdot n \cdot 2T(n-1)$
- $T(n) \in O((|R| \cdot n)^n)$.

Still exponential...

Efficient Algorithm

- Keep in memory the results of Derive(A, x).
  - Number of different inputs: $|V| \cdot n^2$.
  - Only $|V| \cdot n^2$ calls, each takes $O(|R| \cdot n)$.
  - $T(n) \in O(|R| \cdot n^3 \cdot |V|)$.
- Polynomial time!
- This approach is called Dynamic Programming

  Basic idea:

  - If number of different inputs is limited, say $I(n)$.
  - Each run (excluding recursive calls) takes at most $R(n)$ time
  - Total running time is bounded by $T(n) \leq R(n)I(n)$.