Computational Models - Lecture 4[1]

Handout Mode

Nachum Dershowitz & Yishay Mansour

Tel Aviv University.

April 3–19 , 2017

[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)
- (begin) Algorithmic issues for CFL
- Chomsky Normal Form (CNF)
- Checking membership in a CNF grammar

- Sipser's book, 2.1

- **Midterm: May 26th, 2017 !!!**

Short overview

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:

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

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:

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

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:

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

A specific derivation in G_2:

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

More strings generated by G_2: a flower sees, the girl touches

English-like sentences, cont.

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

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 $(V \cap \Sigma = \emptyset)$
- R is a finite set of rules of the form $A \to x$, where $A \in V$ and $x \in (V \cup \Sigma)^*$.
- $S \in V$ is the start symbol.
- Let $u, v \in (V \cup \Sigma)^*$. If $A \to w \in R$, then uAv yields uwv, denoted $uAv \to uwv$.
- $u \stackrel{*}{\to} v$ if $u = v$, or $u \to u_1 \to \ldots \to u_k \to v$ for some sequence $u_1, u_2, \ldots, u_k$

Note that if $A \stackrel{*}{\to} xBy$ and $B \stackrel{*}{\to} z$, then $A \stackrel{*}{\to} xzy$.

Definition 2

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

where $\stackrel{*}{\to}$ 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?

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

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

What do we need to show?

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

Proving $ww^R \in \mathcal{L}(G_4)$. Proof by induction on the length of z.

- Base: $|z| = 0$. Since $S \to \varepsilon = z$, $z \in \mathcal{L}(G_4)$.
- Inductive step. Let $z = ww^R$ be a word of size $2k$.
  - Let $w = \sigma w'$ (hence $|w'| = k - 1$)
  - Hence, $z = \sigma w'(w')^R \sigma$
  - By i.h. $S \stackrel{*}{\to} w'(w')^R$
  - Hence, $S \to \sigma S \sigma \stackrel{*}{\to} \sigma w'(w')^R \sigma = z$

Proving $z \in \mathcal{L}(G_4) \implies z \in \{ww^R : w \in \{a,b\}^*\}$

Proof by induction on the number of derivations steps used to derive z from S.

- Single derivation. Only possible derivation is $S \to \varepsilon$, and thus $z = \varepsilon \in \mathcal{L}$.
- Inductive step. Assume $S \stackrel{*}{\to} z$ in k derivations steps.
  - First derivation is $S \to \sigma S \sigma$ for $\sigma \in \{a,b\}$
  - Hence $z = \sigma z' \sigma$, and z' is derived from S using $k-1$ steps
  - By i.h., $z' = w'(w')^R \in \mathcal{L}$
  - Hence, $z = \sigma w'(w')^R \sigma = w(w)^R$ for $w = \sigma w'$ is in $\mathcal{L}$

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

What do we need to show?

1. $w \in \mathcal{L}(G_5) \implies w \in \mathcal{L}$.
2. $w \in \mathcal{L} \implies w \in \mathcal{L}(G_5)$.

Claim 6

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

Proof: DIY

Claim 7

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

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

Are we done? How to prove the claim?

Proving Claim 7

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

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

Proof by induction on $|w|$:

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

  Assume for concreteness (other cases proved analogously)

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

  By i.h., $S \stackrel{*}{\to} w'B^{k'}$

  Hence, $S \stackrel{*}{\to} w'B^{k'} = w'BB^{k'-1} \stackrel{*}{\to} w'aBBB^{k'-1} = wB^{k'+1} = wB^k$

Section 1

Parse Trees

Parse trees

Definition 8 (parse tree)

A labeled tree T is a parse tree of CFG $G = (V, \Sigma, R, S)$, if

- Inner nodes labels by elements of V.
- Leaves are labeled by elements of $V \cup \Sigma \cup \{\varepsilon\}$.
- If $n_1, \ldots, n_k$ are the labels of the direct descendants (from left to right) of a node labeled A, then $(A \to n_1, \ldots, n_k) \in R$

The yield of T, are the labels of all its leaves ordered written from left to right.

Theorem 9

Let $G = (V, \Sigma, R, S)$ be a CFG, let $A \in V$ and let $x \in (\Sigma \cup V)^*$. Then $A \stackrel{*}{\to} x$ iff G has a parse tree root labeled by A, that yields x.

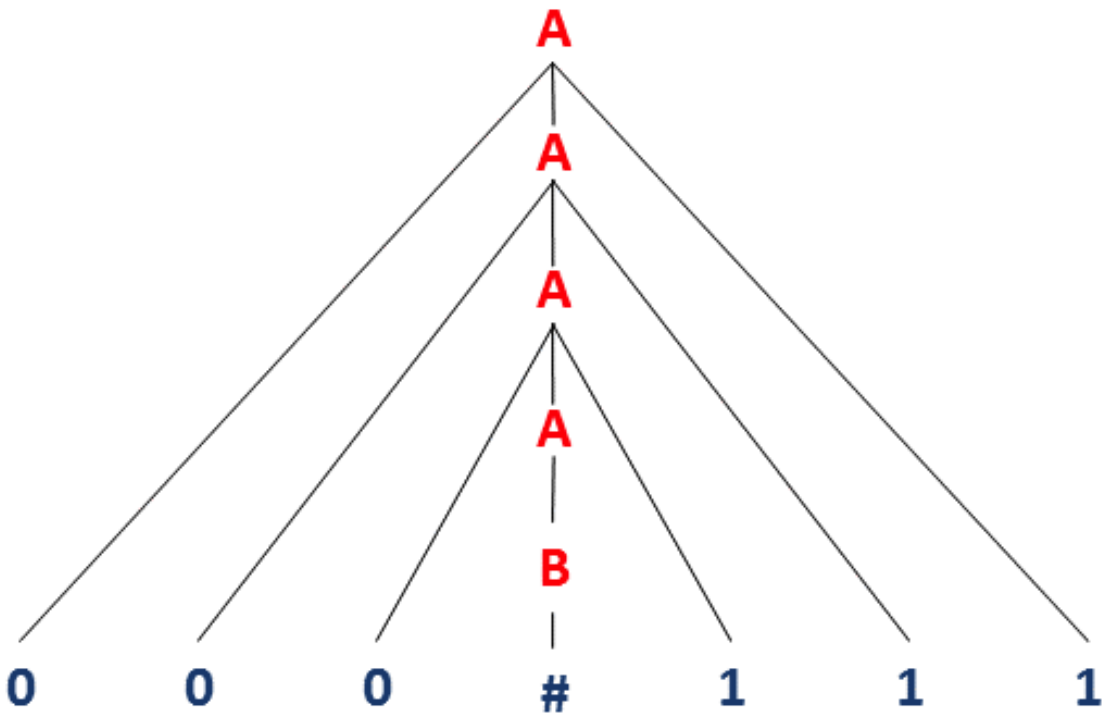
Proof? DIY

Might be that several derivation sequences (i.e., $S \to \ldots \to w$) show that $w \in \mathcal{L}(G)$, but only
A single tree rooted by S that yields w might have several derivation sequences (i.e., $S \to \ldots \to w$) that show that $w \in \mathcal{L}(G)$ (i.e., trees are

Example 1

$A \to 0A1|B$

$B \to \#$



The (unique) derivation sequence from A to $000\#111$:
$A \to 0A1 \to 00A11 \to 000A111 \to 000B111 \to 000\#111$

Example 2

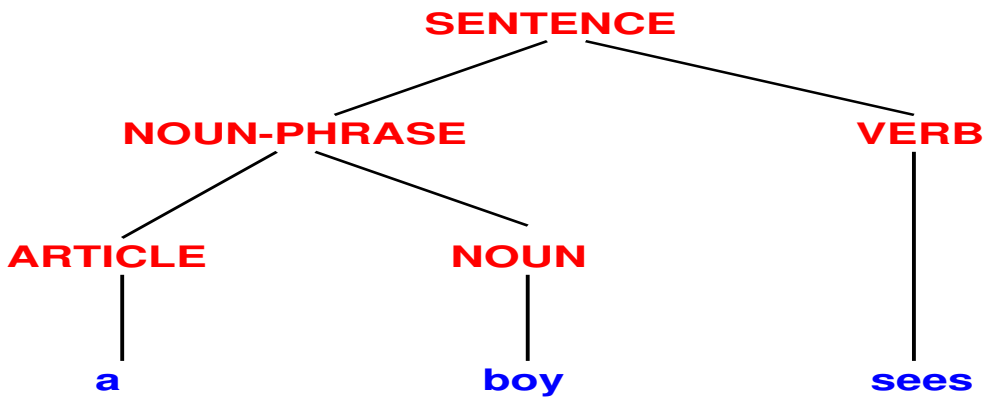
$< \text{SENTENCE} > \rightarrow < \text{NP} >< \text{VERB} >$

$< \text{NP} > \rightarrow < \text{ARTICLE} >< \text{NOUN} >$

$< \text{NOUN} > \rightarrow$boy | girl | flower

$< \text{ARTICLE} > \rightarrow$a | the

$< \text{VERB} > \rightarrow$touches | likes | sees



Two derivation sequences from $< \text{SENTENCE} >$ to *aboysees*:

- $< \text{SENTENCE} > \rightarrow < \text{NP} >< \text{VERB} > \rightarrow < \text{ARTICLE} >< \text{NOUN} >< \text{VERB} > \rightarrow$ *a* $< \text{NOUN} >< \text{VERB} > \rightarrow$ *aboy* $< \text{VERB} > \rightarrow$ *aboysees*
- $< \text{SENTENCE} > \rightarrow < \text{NP} >< \text{VERB} > \rightarrow < \text{ARTICLE} >< \text{NOUN} >< \text{VERB} > \rightarrow < \text{ARTICLE} >< \text{NOUN} >$ *sees* $\rightarrow < \text{ARTICLE} >$ *boysees* $\rightarrow$ *aboysees*

Section 2

Designing Context-Free Grammars

Designing CFGs

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 $S \to S_1 \mid S_2 \mid \ldots \mid S_i$.
- For languages (like $\{0^n\#1^n : n \geq 0\}$), with linked substrings, a rule of form $R \to uRv$ is helpful to force desired relation between substrings.
- For a regular language, grammar "follows" a DFA for the language (see next frame).

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 10

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

Proof?

Claim 11

$R_0 \stackrel{*}{\to} wR_j$ iff $\widehat{\delta}_M(q_0, w) = q_j$.

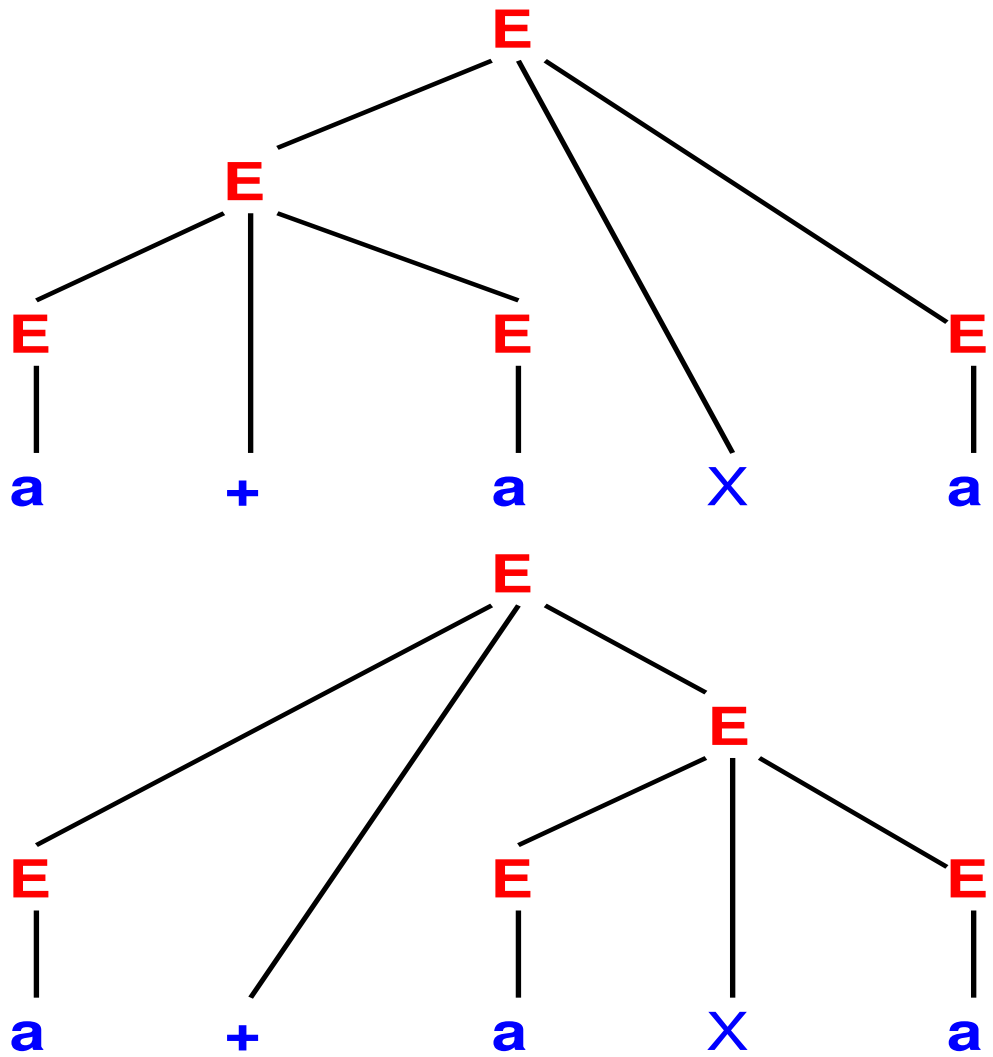
Proof? Next class we'll see alternative proof via "Push-Down Automata"

Section 3

Ambiguity in Context-Free Languages

Ambiguity in CFLs

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



Is this a problem?

Ambiguity in CFLs cont.

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

- $V = \{E, T, F\}$
- $\Sigma = \{a, +, \times, (,)\}$
- Rules:

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

Claim 12

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

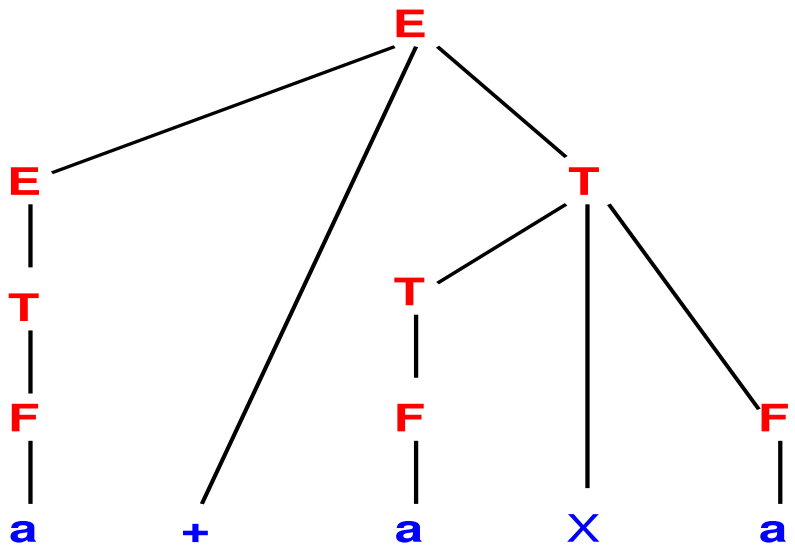
Proof Induction on derivation length.

But G' is not ambiguous. Proof (see Hopcroft, for a similar grammar)

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

G':

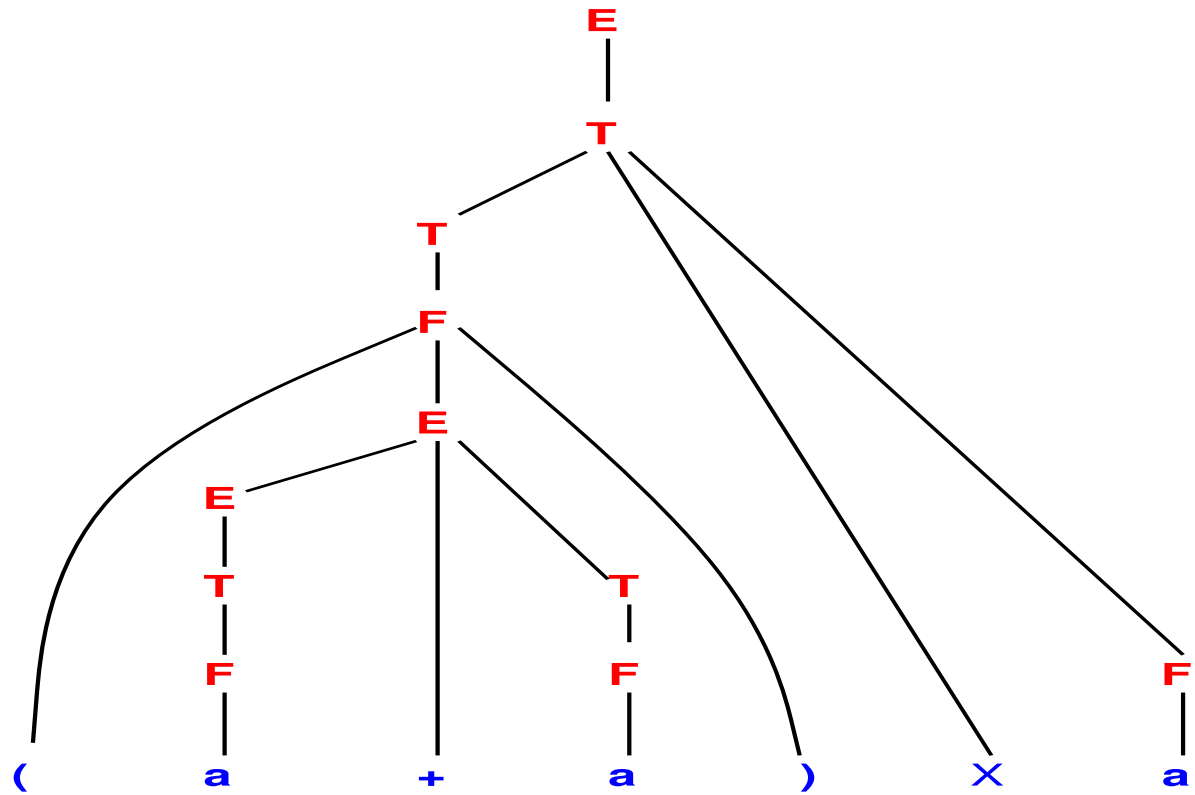
$$
\begin{aligned}
E &\rightarrow E + T \mid T \\
T &\rightarrow T \times F \mid F \\
F &\rightarrow (E) \mid a
\end{aligned}
$$



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 13

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: $\mathcal{L} = \{a^n b^n c^m d^m : n, m \geq 1\} \cup \{a^n b^m c^m d^n : n, m \geq 1\}$
- $\mathcal{L}$ is a CFL. Proof?
- $\mathcal{L}$ is inherently ambiguous. Proof? see Hopcroft

Checking membership in CFLs

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.

Part I

Chomsky Normal Form (CNF)

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:

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

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

What does parse tree look like?

Most internal nodes are degree 2 (except parents of leaves, which are degree 1)

CNF has bounded derivation length

Lemma 14

Let G be a CFG in CNF and let $w \in \mathcal{L}(G)$ be with $|w| = n \geq 1$. Then every derivation of w by G has a derivation of length $2n - 1$.

Proof? consider the parsing tree for w

Advantage: Easier to check whether $w \in \mathcal{L}(G)$,for example: check all derivations of length $2n - 1$, which is finite (yet, exponential).

Checking membership for CFG in CNF form

Let $G = (V, \Sigma, R, S)$ be CFG in CNF and let $A \in V$ and $w \in \Sigma^*$

Algorithm 15 (Derive(A, w))

- $w = \varepsilon$: if $A \to \varepsilon \in R$ (i.e., $A = S$) return `TRUE`, otherwise return `FALSE`.
- $|w| = 1$: if $A \to w \in R$ return `TRUE`, otherwise return `FALSE`.
- $|w| > 1$: for each $A \to BC$ and **each** non-trivial partition $w = w_1 w_2$:
  - Call Derive(B, w_1) and Derive(C, w_2).
  - Return `TRUE` if *both* return `TRUE`.
- Return `FALSE`.

Claim: $A \stackrel{*}{\to} w \iff$ Derive$(A, w) =$ `TRUE`. Proof?

$A \stackrel{*}{\to} w \implies$ Derive$(A, w) =$ `TRUE`, by induction on # of derivation steps.

Derive$(A, w) =$ `TRUE` $\implies A \stackrel{*}{\to} w$, by induction on $|w|$.

- Hence, Derive$(S, w) =$ `TRUE` $\iff w \in \mathcal{L}(G)$.
- Procedure Derive can also output a parse tree for w
- Where have we used the fact that G is in CNF?

Time complexity of Derive

What is the time complexity $T : \mathbb{N} \mapsto \mathbb{N}$ 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, w).
  - 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 $T(n) \leq R(n)I(n)$.

Generality of CNF

Theorem 16

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

Proof Idea:

- Add new start symbol S_0.
- Convert "long rules" to proper form (binary and a separate rule for each letter)
- Eliminate all ε rules of the form $A \to \varepsilon$.
- Eliminate all "unit" rules of the form $A \to B$.

Add new start symbol

Add new start symbol S_0 and rule $S_0 \to S$

(Guarantees that new start symbol is never on right hand side of a rule)
e.g.

$$\begin{aligned} S &\to A \mid ab \mid \varepsilon \\ A &\to baA \mid S \end{aligned}$$

becomes

$$\begin{aligned} S_0 &\to S \\ S &\to A \mid ab \mid \varepsilon \\ A &\to baA \mid S \end{aligned}$$

Convert "long rules": terminals

$$
\begin{aligned}
S &\rightarrow ccAbA \mid bc \mid b \\
A &\rightarrow a \mid bb
\end{aligned}
$$

becomes

$$
\begin{aligned}
S &\rightarrow CCABA \mid BC \mid b \\
A &\rightarrow a \mid BB \\
B &\rightarrow b \\
C &\rightarrow c
\end{aligned}
$$

Convert "long rules": multiple nonterminals

$$S \rightarrow AAAB$$

becomes

$$\begin{aligned} S &\rightarrow AN_1 \\ N_1 &\rightarrow AN_2 \\ N_2 &\rightarrow AB \end{aligned}$$

Eliminate "ε-rules"

Repeat until all $A \to \epsilon$ ($A \neq S_0$) rules are gone:

- remove $A \to \varepsilon$
- for any rule of form $C \to AB$ or $C \to BA$, add $C \to B$.
- for any rule of form $C \to AA$ add $C \to A$ and $C \to \varepsilon$ (unless $C \to \varepsilon$ has already been removed).
- for any rule of form $C \to A$ add $C \to \varepsilon$ (unless $C \to \varepsilon$ has already been removed.)

Eliminate "unit rules"

Repeat until all unit rules removed

- remove some $A \to B$
- for each $B \to CD$ (where $C, D \in V$), add $A \to CD$
- for each $B \to C$ (where $C \in V$), add $A \to C$ (unless $A \to C$ was a previously removed unit rule)

CNF: Example

$$
\begin{aligned}
S &\rightarrow ASA \mid aB \\
A &\rightarrow B \mid S \\
B &\rightarrow b \mid \varepsilon
\end{aligned}
$$

Is transformed into:

$$
\begin{aligned}
S_0 &\rightarrow AA_1 \mid UB \mid a \mid SA \mid AS \\
S &\rightarrow AA_1 \mid UB \mid a \mid SA \mid AS \\
A &\rightarrow b \mid AA_1 \mid UB \mid a \mid SA \mid AS \\
A_1 &\rightarrow SA \\
U &\rightarrow a \\
B &\rightarrow b
\end{aligned}
$$