MA/CSSE 474
Theory of Computation

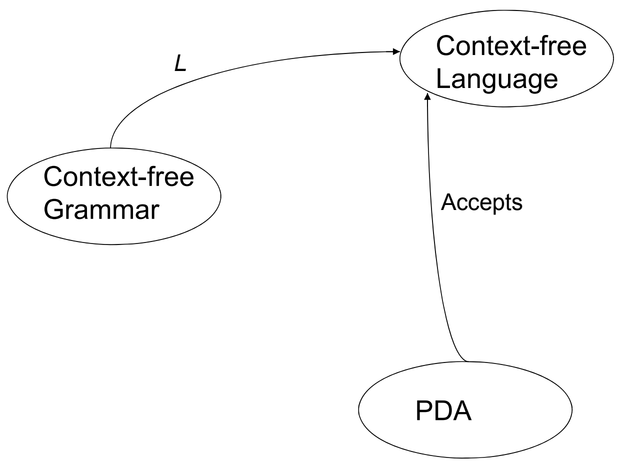
Intro to Context-free Grammars

Your Questions?

- Previous class days' material (and exercises)
- Reading Assignments
- HW 9 problems
- Exam 2
- Anything else

Your parents will probably concur!...

Context-free Grammars, Languages, and PDAs



Shorthand notation

$S \to \varepsilon$
$S \to aT$
$S \to bT$
$T \to a$
$T \to b$
$T \to aS$
$T \to bS$

Can be abbreviated by

$S \to \varepsilon \mid aT \mid bT$
$T \to a \mid b \mid aS \mid bS$

Context-free Grammar Formal Definition

A CFG G=(V, Σ, R, S) (Each part is finite)

Σ is the ***terminal alphabet***; it contains the set of symbols that make up the strings in *L*(*G*), and

N *(our textbook does not use this name, but I will) is the* ***nonterminal alphabet***: a set of working symbols that G uses to structure the language. These symbols disappear by the time the grammar finishes its job and generates a string. **Note:** $\Sigma \cap N = \varnothing$.

Rule alphabet (vocabulary): **V** = $\Sigma \cup N$

- **R**: A finite set of productions of the form A → β, where A ∈ N and β ∈ V* | Rules are also known as **productions**.

G has a unique **start symbol**, **S** ∈ N

Formal Definitions: Derivations, Context-free Languages

$x \Rightarrow_G y$ iff $x = \alpha A \beta$

and $A \rightarrow \gamma$ is in R

$y = \alpha \gamma \beta$

$w_0 \Rightarrow_G w_1 \Rightarrow_G w_2 \Rightarrow_G \ldots \Rightarrow_G w_n$ is a **derivation** in *G*.

Let $\Rightarrow_G{}^*$ be the reflexive, transitive closure of $\Rightarrow_G$.

Then the **language generated by G**, denoted *L*(*G*), is:

$\{w \in \Sigma^* : S \Rightarrow_G{}^* w\}$.

A language *L* is ***context-free*** if there is some context-free grammar *G* such that $L = L(G)$.

Regular Grammars

A brief side-trip into Chapter 7

Regular Grammars

In a regular grammar, every rule (production) in R must have a right-hand side that is:

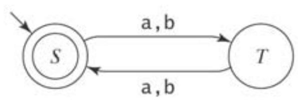
- ε, or
- a single terminal, or
- a single terminal followed by a single nonterminal.

Regular: $S \to \mathrm{a}$, $S \to \varepsilon$, and $T \to \mathrm{a}S$

Not regular: $S \to \mathrm{a}S\mathrm{a}$ and $S \to T$

Regular Grammar Example

$L = \{w \in \{a, b\}^* : |w| \text{ is even}\}$ $((aa) \cup (ab) \cup (ba) \cup (bb))^*$



$S \rightarrow \varepsilon$
$S \rightarrow aT$
$S \rightarrow bT$
$T \rightarrow a$
$T \rightarrow b$
$T \rightarrow aS$
$T \rightarrow bS$

Derive *abbb* from this grammar

Regular Languages and Regular Grammars

Theorem: A language is regular iff it can be defined by a regular grammar.

Proof: By two constructions.

Regular Languages and Regular Grammars

Regular grammar $\rightarrow$ ***FSM:***

grammartofsm($G = (V, \Sigma, R, S)$) =

1. Create in M a separate state for each nonterminal in V.
2. Start state is the state corresponding to S .
3. If there are any rules in R of the form $X \rightarrow a$, for some $a \in \Sigma$, create a new state labeled #.
4. For each rule of the form $X \rightarrow a\ Y$, add a transition from X to Y labeled a.
5. For each rule of the form $X \rightarrow a$, add a transition from X to # labeled a.
6. For each rule of the form $X \rightarrow \varepsilon$, mark state X as accepting.
7. Mark state # as accepting.

$S \rightarrow bS,\ S \rightarrow aT$
$T \rightarrow aS,\ T \rightarrow b,\ T \rightarrow \varepsilon$

FSM $\rightarrow$ ***Regular grammar:*** Similar.
Essentially reverses this procedure.

Recursive Grammar Rules

- A rule is ***recursive*** iff it is $X \rightarrow w_1 Y w_2$, where:
  $Y \Rightarrow^* w_3 X w_4$ for some w_1, w_2, w_3, and w_4 in V^*.

- A grammar G is ***recursive*** iff G contains at least one recursive rule.

- **Examples:** $S \rightarrow (S)$ $\qquad$ $S \rightarrow (T)$
  $\qquad\qquad\qquad\qquad\qquad\qquad\qquad T \rightarrow (S)$

In general, non-recursive grammars are boring!

Self-Embedding Grammar Rules

- A rule in a grammar G is ***self-embedding*** iff it is :
  $X \rightarrow w_1 Y w_2$, where $Y \Rightarrow^* w_3 X w_4$ and both $w_1 w_3$ and $w_2 w_4$ are in Σ^+.

What is the difference between *self-embedding* and *recursive*?

- A grammar is ***self-embedding*** iff it contains at least one self-embedding rule.
- Examples: $S \rightarrow \text{a}S\text{a}$ self-embedding

  $S \rightarrow \text{a}S$ recursive but not self-embedding

  $S \rightarrow \text{a}T$
  $T \rightarrow S\text{b}$ self-embedding

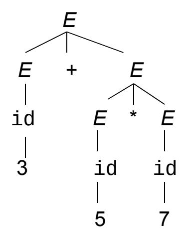
Where Context-Free Grammars Get Their Power

- If a CFG G is not self-embedding then $L(G)$ is regular.
- If a language L has the property that every grammar that defines it is self-embedding, then L is not regular.

Structure

Context free languages:

We care about structure.



Derivation Tree

- Consider our grammar for **Bal:**
  S → (S) | ε | SS
- Draw a derivation tree (a.k.a. Parse tree) for the string (())((()())

Hints for designing context-free grammars

- Generate concatenated regions:
  A → BC
- Generate outside in:
  A → aAb
- Union of two sets:
  A → B | C

$L = \{a^n b^n c^m : n, m \geq 0\}$

$L = \{ a^{n_1} b^{n_1} a^{n_2} b^{n_2} \ldots a^{n_k} b^{n_k} : k \geq 0 \wedge \forall i \leq k\ (n_i \geq 0)\}$

$L = \{a^n b^m : n \neq m\}$

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

CFG for Simple Arithmetic Expressions

$G = (V, \Sigma, R, E)$, where
$V = \{+, *, (,), \texttt{id}, E\}$,
$\Sigma = \{+, *, (,), \texttt{id}\}$,
$R = \{$
$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow (E)$
$E \rightarrow \texttt{id}$
$\}$

Derive id + id * id

BNF

A notation for writing practical context-free grammars

- The symbol | should be read as “or”.

  Example: $S \rightarrow \mathrm{a}S\mathrm{b} \mid \mathrm{b}S\mathrm{a} \mid SS \mid \varepsilon$

- Allow a nonterminal symbol to be any sequence of characters surrounded by angle brackets.

  Examples of nonterminals:

  <program>
  <variable>

BNF for a Java Fragment

```
<block>     ::= {<stmt-list>}  |
                {}

<stmt-list> ::= <stmt>  |
                <stmt-list> <stmt>

<stmt>      ::= <block>  |
                while (<cond>) <stmt>  |
                if (<cond>) <stmt>  |
                do <stmt> while (<cond>);
|
               <assignment-stmt>;  |
               return  |
               return <expression>  |
               <method-invocation>;
```

Spam Generation

```
<spc> → space | . | - | _ | = | : | * | / | ¦ | empty

<Word> → <V> <spc> <I> <spc> <A> <spc> <G> <spc> <R> <spc> <A>
  <V> → V | v | \/
  <I> → I | i | ! | ¡ | : | ì | í | ï | î | ì | Í | Ï | Î | ¦ | l | 1
  <A> → A | a | /\ | @ | ^ | Á | Â | À | Å | Ã | á | â | ä | à | å | ã
  <G> → G | g | & | 6 | 9
  <R> → R | r | ®
```

Example production:

```
<spc> → -
  <V> → v   <I> ::= !   <A> ::= ä   <G> ::= G   <R> ::= ®   <A> ::= ^
<Word> → v-!-ä-G-®-^
```

These production rules yield 1,843,200 possible spellings.
How Many Ways Can You Spell V1@gra? By Brian Hayes
American Scientist, July-August 2007
http://www.americanscientist.org/template/AssetDetail/assetid/55592

HTML

```
<ul>
    <li>Item 1, which will include a sublist</li>
        <ul>
            <li>First item in sublist</li>
            <li>Second item in sublist</li>
        </ul>
    <li>Item 2</li>
</ul>
```

A grammar:

/* Text is a sequence of elements.
`HTMLtext` → `Element HTMLtext` | ε

`Element` → `UL` | `LI` | … (and other kinds of elements that are allowed in the body of an HTML document)

/* The `<ul>` and `</ul>` tags must match.
`UL` → `<ul>` `HTMLtext` `</ul>`

/* The `<li>` and `</li>` tags must match.
`LI` → `<li>` `HTMLtext` `</li>`

English

S → *NP VP*
NP → `the` *Nominal* | `a` *Nominal* | *Nominal* | *ProperNoun* | *NP PP*
Nominal → *N* | *Adjs N*
N → `cat` | `dogs` | `bear` | `girl` | `chocolate` | `rifle`
ProperNoun → `Chris` | `Fluffy`
Adjs → *Adj Adjs* | *Adj*
Adj → `young` | `older` | `smart`
VP → *V* | *V NP* | *VP PP*
V → `like` | `likes` | `thinks` | `shoots` | `smells`
PP → *Prep NP*
Prep → `with`

Prove the Correctness of a Grammar

$A^nB^n = \{a^nb^n : n \geq 0\}$

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

$R = \{ \; S \rightarrow a\, S\, b$
$\quad S \rightarrow \varepsilon$
$\}$

- Prove that *G* generates only strings in *L.*
- Prove that *G* generates all the strings in *L*.