Lec 22 Context Free Grammar

	Human	
ID	↳ list of char	fact (3)
lexing	↳ list of tokens	(ID "fact") :: LPAREN :: INT 3 :: RPAREN
parsing	↳ expression	App (Var "fact" , Num 3) ← abstract syntax tree

Chomsky hierarchy

level		representation		application
General grammar	—	turing machine	—	computing
Context-sencitive grammar	—	whatever		whatever
Context-free grammar	—	pushdown automata	—	parsing
Regular grammar	—	finite automata	—	lexing / search

Context free grammar

programme

$P \to \varepsilon \mid E ; P$

empty

$E \to E + E \mid E * E \mid E \text{ andalso } E \mid \text{case } E \text{ of } M \mid \dots$

expression

$M \to Q \Rightarrow E \mid Q \Rightarrow E \text{ "|" } M$

$Q \to () \mid \cdots$

Def A context free grammar (CFG) is a tuple (Σ, V, S, R) s.t.

- Σ is set of terminals viz. chars we see in program,
- V is set of non-terminals ($\Sigma \cap V = \emptyset$)
- $S \in V$ is the start symbol
- R is set of rules of form $N \to w$ where $N \in V$, $w \in (\Sigma \cup V)^*$ ← star from regex

Suppose $\alpha, \beta \in (\Sigma \cup V)^*$

- β is derivable from α in one step ($\alpha \overset{1}{\Rightarrow} \beta$) if:
  $\exists \gamma, \delta$ s.t. $\alpha = \gamma N \delta$ and $\beta = \gamma \omega \delta$ where $N \to \omega \in R$
- β is derivable from α if $\alpha = \beta$ or $\alpha \overset{1}{\Rightarrow} \sigma_1 \overset{1}{\Rightarrow} \sigma_2 \Rightarrow \cdots \overset{1}{\Rightarrow} \beta$
- $L(G) = \{ \omega \in \Sigma^* \mid S \Rightarrow \omega \}$

Ex. Let $G = (\Sigma, V, S, R)$

$\Sigma = \{a, b\}$

$S = \{S, A\}$

$$R = \left\{ \begin{array}{l} S \to AbA \\ A \to \varepsilon \\ A \to a \\ A \to aA \end{array} \right\}$$

$A \to a$ ← actually redundant

Derive

$S \overset{1}{\Rightarrow} AbA \overset{1}{\Rightarrow} abA \overset{1}{\Rightarrow} abaA \overset{1}{\Rightarrow} aba$ ← left most derivation: always expand left most

$S \overset{1}{\Rightarrow} AbA \overset{1}{\Rightarrow} AbaA \overset{1}{\Rightarrow} Aba \overset{1}{\Rightarrow} aba$

using rule 3. Multiple ways. Then we get multiple parse trees :C may cause inconsistency ← Ambiguous

<u>Def</u> A grammar is ambiguous if it has multiple left-most derivation

→ Figuring out if a CFG is ambiguous is undecidable

→ There are languages s.t. all CFG to describe it is ambiguous

<u>Thm</u> All regex can be expressed as CFG

0 – $(\Sigma, \{S\}, S, \emptyset)$

1 – $(\Sigma, \{S\}, S, \{S \to \varepsilon\})$

$a - (\Sigma, \{S\}, S, \{S \to a\})$

$\vdots$

Ex. $\{a^n \mid n \equiv 0 \pmod 3, n \geq 0\}$ ← regular. finite storage to accept.
$\{a^n b^n \mid n \geq 0\}$ ← not regular, but context free. finite storage + stack to accept
$\{a^n b^n c^n \mid n \geq 0\}$ ← not context free. Need finite storage + at least two stacks
$\{a^n b^m c^m d^n \mid m, n \geq 0\} \cup \{a^n b^n c^m d^m \mid m, n \geq 0\}$ ← context free, but always ambiguous

<u>Pumping Lemma</u> — for proving some lang is not in a Chomsky class

If L is infinite regular regular language, $\exists\, \alpha, \omega, \beta$ s.t.
$\omega \neq \varepsilon$ and $\alpha \omega^k \beta \in L$ for every k viz. there's some loop

Consider $\{a^n b^n \mid n \geq 0\}$. Suppose it's regular.
Then $\exists\, w \in \varepsilon$, $\alpha w^k \beta \in L$ for all k.
idea is that we can't repeat any ab, aabb, ..

Consider $S \to \varepsilon \mid aSa \mid bSb \mid a \mid b$

Simple Parsor ← (usually we can use parser generator these days)

probably better
→ Bottom up : try rebuild to S from string
→ Top down : start from S and try make target string
for now

$G_1 = \{S \to x \mid \lambda x . S \mid (S\ S)\}$

```
datatype token = LAMBDA | LPAREN | RPAREN | ID of string | DOT
datatype exp = Var of string
             | Lam of string * exp
             | App of exp * exp
```

```
(* parseExp : token list -> (exp * token list -> 'a) -> 'a
fun parseExp (ID x :: ts) k = Var (x, ts)
  | parseExp (LPAREN :: ts) k =
      parseExp ts (fn (e1, ts') =>
          parseExp ts' (fn (e2, RPAREN :: ts'') =>
            k (App (e1, e2), ts'')
      )
  | parseExp (LAMBDA :: ID x :: DOT :: ts) k =
     parseExp ts (fn (e, ts') => k (Lam (x, e), ts'))
  | ...         _   _          = raise SyntaxError
```

Expect

(λx.x y)