CS3200 - Parsing

Note Title 1/25/2016

Announcements

- HW due Saturday

Last time - Ch 2., Sec. 3

- Context free languages
- Automated Parsing tools
  - bison

Examples: Palindromes

Recursive dfn: First & last letters match, & interior is a palindrome.

alphabet $\{a, b\}$

$$S \rightarrow aSa \mid bSb \mid \varepsilon$$

(↑ or)

$$S \rightarrow a \mid b$$

Ex: Derive a CFG for all binary strings w/ an even # of 0's.

$$S \rightarrow S0S0S \mid \varepsilon \mid S1S$$

0011

An alternate for that last one:

$S \to 0T \mid T0$?

$\quad \to \varepsilon$

$\quad \to S1S$

$T \to S0S$

$S \to S1S \to 0T1S$

$\vdots$

00100: $S \to 0T \to 0S0S \to 00S$

$\to 00S1S \to 001S$

$\to 001\,0T \to 0010S0S$

$\varepsilon \quad \varepsilon$

Context-Free Languages

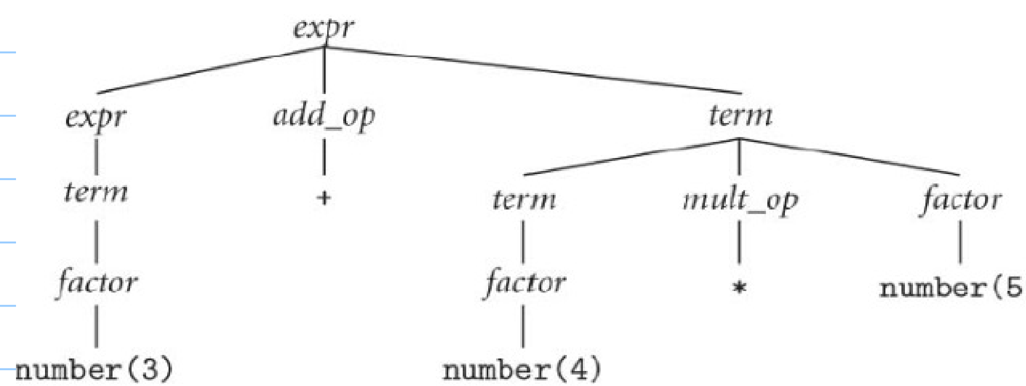
Recall that for any context free language, there / are an infinite # of grammars that can produce it.

We wish to somehow give a definition of a "good" set of productions.

Goal: Parsing (well) – given a language, detect if a string is in that language.

Why?

- Remember, goal is to compile a program.
- Need to be able to quickly check if it is valid and give a parse tree back!



So - a bad example: $S_0 \to S \mid X \mid Z$

$S \to A$

$A \to B$

$C \to Aa$

$X \to C$

$Y \to aY \mid a$

$Z \to \varepsilon$

Why is it bad?

- useless states
- unreachable non-terminals
- unit pair

Chomsky Normal Form (CNF)

Each rule in the grammar is either:

- $A \rightarrow BC$
  where neither B or C is the start variable, & both are nonterminals
- $A \rightarrow a$
  where a is a terminal
- $S \rightarrow \varepsilon$
  where S is the start symbol

__But:__ Clearly, not always in CNF!

__Luckily:__

__Thm:__ All grammars can be converted to CNF.

Procedure:

(1) Create a new start symbol S_0, & send $S_0 \to S$

(1a) Eliminate useless rules
(just delete ones that can't be reached)

② Remove nullable variables.
$A \to \varepsilon$

How?
Remove all ε productions.
Then fix.

Ex: $A \to CBC$ } $A \to CBC \mid CC$

⋮

~~$B \to \varepsilon$~~

$B \to b$

$D \to BACB$ } $D \to BACB \mid ACB \mid BAC \mid AC$

③ Remove unit rules:

$S \to A$

How? Must have:

$X \to Z_1,\ Z_1 \to Z_2,\ \dots\ Z \to Y$

(since we removed ε-transitions in ②)

Then:

$$\left[\begin{array}{l} S \to XAY \\ \underline{A \to B} \mid XY \\ B \to b \mid c \end{array}\right] \to \begin{array}{l} S \to XbY \mid XcY \\ A \to XY \end{array}$$

(Still ③)

For each unit pair (A, B)
and rule $B \to w$,

add $A \to w$ to a <u>new</u>
grammar.

(Note that (A, A) is a unit pair,
so all rules $A \to w$
will stick around.)

(4) Get rid of "long" righthand sides.

4a: Create $V_c \rightarrow c$ for every character.

Replace c with V_c everywhere.

Now either

$$A \rightarrow CDEF$$

or

$$V_c \rightarrow c.$$

4b: $A \rightarrow B_1 B_2 B_3 \dots B_k$

How to replace with only 2 nonterminals on the right?

$A \rightarrow B_1 X_1$

$X_1 \rightarrow B_2 X_2$

$X_2 \rightarrow B_3 X_3$

$\vdots$

$X_{k-2} \rightarrow B_{k-1} B_k$

$A \rightarrow WXYZ$

↳ $A \rightarrow WB$

$B \rightarrow XC$

$C \rightarrow YZ$

Ex:

$$S \rightarrow ASA \mid aB$$
$$A \rightarrow B \mid S$$
$$B \rightarrow b \mid \varepsilon$$

Now – why do we care??

Parsing: building those parse trees we saw

In general, there are an exponential number of parse trees for a given input.

So how to check quickly?

Even in CNF, might be 2^n possible parse trees.

<u>Cocke - Younger - Kasami (CYK) algorithm</u>

Uses a table & dynamic programming to give a parse tree in $O(n^3)$ time.

Grammar **must** be in CNF!

CYK Algorithm

Given a word $w = w_1 w_2 w_3 w_4 \cdots w_k$, we'll look at all possible substrings $w_i w_{i+1} \cdots w_{j-1} w_j$, and look at how they can be parsed.

We'll build a table from the bottom up.

Ex:

$S \to AB \mid BC$
$A \to BA \mid a$
$B \to CC \mid b$
$C \to AB \mid a$

Test if 'baaba' is in the language

yes — S is in top spot

5	{S, A				
4	{}	{S, C, A}			
length 3	{S}	{B}	{B}		
length 2	{S, A}	{B}	{S, C}	{S, A}	
length 1	{B}	{A, C}	{A, C}	{B}	{A, C}
	b	a	a	b	a

i →

Running times:

Say we have n rules.

Converting to CNF:

$O(n^2)$

Running CYK: n^3 time, & n^2 space

Other parsing algorithms

CYK is still pretty slow, especially for large programming languages.

After it was developed, a lot of work was put into figuring out what grammars could have faster algorithms.

Two big (& useful) classes have linear time parsers: LL & LR.