CYK Algorithm

Lecture 15

October 16, 2018

Parsing

We saw regular languages and context free languages.

Most programming languages are specified via context-free grammars. Why?

- CFLs are sufficiently expressive to support what is needed.
- At the same time one can "efficiently" solve the parsing problem: given a string/program w, is it a valid program according to the CFG specification of the programming language?

CFG specification for C

```
<relational-expression> ::= <shift-expression>
                          | <relational-expression> < <shift-expression>
                          | <relational-expression> > <shift-expression>
                          | <relational-expression> <= <shift-expression>
                          | <relational-expression> >= <shift-expression>

<shift-expression> ::= <additive-expression>
                     | <shift-expression> << <additive-expression>
                     | <shift-expression> >> <additive-expression>

<additive-expression> ::= <multiplicative-expression>
                        | <additive-expression> + <multiplicative-expression>
                        | <additive-expression> - <multiplicative-expression>

<multiplicative-expression> ::= <cast-expression>
                              | <multiplicative-expression> * <cast-expression>
                              | <multiplicative-expression> / <cast-expression>
                              | <multiplicative-expression> % <cast-expression>

<cast-expression> ::= <unary-expression>
                    | ( <type-name> ) <cast-expression>

<unary-expression> ::= <postfix-expression>
                     | ++ <unary-expression>
                     | -- <unary-expression>
                     | <unary-operator> <cast-expression>
                     | sizeof <unary-expression>
                     | sizeof <type-name>
```

Algorithmic Problem

Given a CFG $G = (V, T, P, S)$ and a string $w \in T^*$, is $w \in L(G)$?

- That is, does S derive w?
- Equivalently, is there a parse tree for w?

Algorithmic Problem

Given a CFG $G = (V, T, P, S)$ and a string $w \in T^*$, is $w \in L(G)$?

- That is, does S derive w?
- Equivalently, is there a parse tree for w?

Simplifying assumption: G is in Chomsky Normal Form (CNF)

- L does not contain ϵ. Productions are all of the form $A \rightarrow BC$ or $A \rightarrow a$ where $a \in T$. Thus no non-terminal can derive ϵ.
- Every CFG G can be converted into CNF form via an efficient algorithm
- Advantage: parse tree is a binary tree.

Example

$S \rightarrow AB \mid XB$
$Y \rightarrow AB \mid XB$
$X \rightarrow AY$
$A \rightarrow \mathbf{0}$
$B \rightarrow \mathbf{1}$

Question:

- Is $\mathbf{000111}$ in $L(G)$?
- Is $\mathbf{00011}$ in $L(G)$?

Towards Recursive Algorithm

Assume G is a CNF grammar.
S derives w iff one of the following holds:

- $|w| = 1$ and $S \rightarrow w$ is a rule in P
- $|w| > 1$ and there is a rule $S \rightarrow AB$ and a split $w = uv$ with $|u|, |v| \geq 1$ such that A derives u and B derives v

Towards Recursive Algorithm

Assume G is a CNF grammar.
S derives w iff one of the following holds:

- $|w| = 1$ and $S \rightarrow w$ is a rule in P
- $|w| > 1$ and there is a rule $S \rightarrow AB$ and a split $w = uv$ with $|u|, |v| \geq 1$ such that A derives u and B derives v

Observation: Subproblems generated require us to know if some non-terminal A will derive a substring of w.

Recursive solution

$w = w_1 w_2 \ldots w_n$
Assume r non-terminals in V

Deriv(A, i, j): 1 if non-terminal A derives substring $w_i w_{i+1} \ldots w_j$, otherwise 0

Recursive formula: Deriv(A, i, j) is 1 iff

- $j = i$ and $A \rightarrow w_i$ is a rule or
- $j > i$ and there is rule $A \rightarrow BC$ and there is $i \leq h < j$ such that Deriv$(B, i, h) = 1$ and Deriv$(C, h+1, j) = 1$

Output: $w \in L(G)$ iff Deriv$(S, 1, n) = 1$.

Analysis

Assume $V = \{A_1, A_2, \ldots, A_r\}$ with $S = A_1$

- Number of subproblems: $O(rn^2)$
- Space: $O(rn^2)$
- Time to evalue a subproblem from previous ones: $O(|P|n)$ where P is set of rules
- Total time: $O(|P|rn^3)$ which is polynomial in both $|w|$ and $|G|$. For fixed G the run time is cubic in input string length.
- Not practical for most programming languages. Most languages assume restricted forms of CFGs that enable more efficient parsing algorithms.

Example

$S \to AB \mid XB$
$Y \to AB \mid XB$
$X \to AY$
$A \to \mathbf{0}$
$B \to \mathbf{1}$

Question:

- Is $\mathbf{000111}$ in $L(G)$?
- Is $\mathbf{00011}$ in $L(G)$?

Order of evaluation for iterative algorithm: increasing order of substring length.

Example

$$S \to AB \mid XB$$
$$Y \to AB \mid XB$$
$$X \to AY$$
$$A \to \mathbf{0}$$
$$B \to \mathbf{1}$$