CS 536 Announcements for Thursday, March 10, 2022

Last Time
- bottom-up parsing
- CYK algorithm
- Chomsky normal form
- removing useless nonterminals

Today
- wrap up CYK
- classes of grammars
- top-down parsing

Next Time
- building a predictive parser
- FIRST and FOLLOW sets

Parsing (big picture)

Context-free grammars (CFGs)
- language generation:
- language recognition:

Translation
- given \( w \in L(G) \), create
- given \( w \in L(G) \), create
**CYK algorithm**

**Step 1:** get grammar in Chomsky Normal Form

**Step 2:** build all possible parse trees bottom-up
- start with runs of 1 terminal
- connect 1-terminal runs into 2-terminal runs
- connect 1- and 2-terminal runs into 3-terminal runs
- connect 1- and 3- or 2- and 2-terminal runs into 4-runs
- ...
- if we can connect entire tree, rooted at start symbol, we’ve found a valid parse

**Pros:** able to parse an arbitrary CFG

**Cons:** $O(n^3)$ time complexity

For special classes of grammars, we can parse in $O(n)$ time

---

**Classes of grammars**

- **LL(1)**

- **LALR(1)**

Both are accepted by parser generators

**LALR(1)**
- parsed by bottom-up parsers
- harder to understand

**LL(1)**
- parsed by top-down parsers
Top-down parsers

- Start at start symbol
- Repeatedly "predict" what production to use

Predictive parser overview
Example

CFG: \[ S \rightarrow (S) | \{S\} | \epsilon \]

Parse table:

<table>
<thead>
<tr>
<th></th>
<th>(    )</th>
<th>{    }</th>
<th></th>
<th>EOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Input: (    {    }    )    EOF
Predictive parser algorithm

stack.push(EOF)
stack.push(start nonterm)
t = scanner.getToken()

repeat

if stack.top is terminal y
match y with t
pop y from stack
t = scanner.getToken()

if stack.top is nonterminal X
get table[X, current token t]
pop X from stack
push production's RHS (each symbol from R to L)

until one of the following:
stack is empty
stack.top is a terminal that does not match t
stack.top is a nonterm and parse-table entry is empty

Example

CFG:  

```
S \rightarrow ( S ) | \{ S \} | \epsilon
```

Parse table:

```
   (   )   \{   \}   EOF

S
```

Input: ( ( } EOF
Consider

**CFG:**

\[ S \rightarrow (S) | \{S\} | () | \{\} | \epsilon \]

**Parse table:**

<table>
<thead>
<tr>
<th></th>
<th>( )</th>
<th>{ }</th>
<th>EOF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Two issues**

1) How do we know if the language is LL(1)?

2) How do we build the selector table?
Converting non-LL(1) grammars to LL(1) grammars

Necessary (but no sufficient conditions) for LL(1) parsing

- free of left recursion – no left-recursive rules
- left-factored – no rules with a common prefix, for any nonterminal

Left recursion

- A grammar $G$ is recursive in nonterm $X$ if $X \Rightarrow^+ \alpha X \beta$
- A grammar $G$ is left recursive in nonterm $X$ if $X \Rightarrow^+ X \beta$
- A grammar $G$ is immediately left recursive in $X$ if $X \Rightarrow X \beta$

Why left-recursion is a problem
Removing left-recursion
Grammars that are not left-factored

If a nonterminal has two productions whose right-hand sides have a common prefix, the grammar is not left-factored.

Example:
Combined example

\[ \text{exp} \rightarrow (\text{exp}) \]
\[ \mid \text{exp exp} \]
\[ \mid (\ ) \]