CS 536 Announcements for Monday, February 12, 2024

Programming Assignment 2 – due Tuesday, February 20

Last Time
- why regular expressions aren't enough
- CFGs
  - formal definition
  - examples
  - language defined by a CFG

Today
- Makefiles
- parse trees
- resolving ambiguity
- expression grammars
- list grammars

Next Time
- syntax-directed translation

Makefiles

Basic structure

```<target>: <dependency list>
  <command to satisfy target>```

Example

```Example.class: Example.java IO.class
  javac Example.java

IO.class: IO.java
  javac IO.java```

Make creates an internal **dependency graph**
- a file is rebuilt if one of its dependencies changes

Variables – for common configuration values to use throughout your makefile

Example

```JC = /s/std/bin/javac
JFLAGS = -g

Example.class: Example.java IO.class
  $(JC) $(JFLAGS) Example.java

IO.class: IO.java
  $(JC) $(JFLAGS) IO.java```
Phony targets
- target with no dependencies
- use make to run commands:

Example
```
clean:
    rm -f *.class
```

Programming Assignment 2

Modify:
- base.jlex
- P2.java
- Makefile

Makefile
```
###
# testing - add more here to run your tester and compare
# its results to expected results
###
test:
    java -cp $(CP) P2
diff allTokens.in allTokens.out

###
# clean up
###
clean:
    rm -f *~ *.class base.jlex.java
cleantest:
    rm -f allTokens.out
```

Running the tester
```
royal-12(53)% make test
java -cp ./deps:. P2
3:1 ****ERROR**** ignoring illegal character: a
diff allTokens.in allTokens.out
3d2
< a
make: *** [Makefile:40: test] Error 1
```
CFG review

formal definition: CFG $G = (N, \Sigma, P, S)$

CFG generates a string by applying productions until no non-terminals remain

$\Rightarrow^+ \text{ means "derives in 1 or more steps"}$

language defined by a CFG $G$

$L(G) = \{ w | s \Rightarrow^+ w \}$ where

$s = \text{start is the start non-terminal of } G,$ an

$w = \text{sequence consisting of (only) terminal symbols or } \varepsilon$

Parse trees

= way to visualize a derivation

To derive a string (of terminal symbols):

- set root of parse tree to start symbol
- repeat
  - find a leaf non-terminal $x$
  - find production of the form $x \rightarrow \alpha$
  - "apply" production: symbols in $\alpha$ become the children of $x$
- until there are no more leaf non-terminals

Derived sequence determined from leaves, from left to right
Parse tree example

Productions
1) prog \( \rightarrow \) BEGIN stmts END
2) stmts \( \rightarrow \) stmts SEMICOLON stmt
3) \( \mid \) stmt
4) stmt \( \rightarrow \) ID ASSIGN expr
5) expr \( \rightarrow \) ID
6) \( \mid \) expr PLUS ID

Derivation order

Productions
1) prog \( \rightarrow \) BEGIN stmts END
2) stmts \( \rightarrow \) stmts SEMICOLON stmt
3) \( \mid \) stmt
4) stmt \( \rightarrow \) ID ASSIGN expr
5) expr \( \rightarrow \) ID
6) \( \mid \) expr PLUS ID

Leftmost derivation :

Rightmost derivation :
Expression Grammar Example

1) $expr \rightarrow \text{INTLIT}$
2) $| \: expr \: \text{PLUS} \: expr$
3) $| \: expr \: \text{TIMES} \: expr$
4) $| \: \text{LPAREN} \: expr \: \text{RPAREN}$

Derive: $4 + 7 * 3$

For grammar G and string w, G is **ambiguous** if there is

OR

OR

OR
Grammars for expressions

**Goal:** write a grammar that correctly reflects precedences and associativities

**Precedence**
- use different non-terminal for each precedence level
- start by re-writing production for lowest precedence operator first

**Example**
1) \( \text{expr} \rightarrow \text{INTLIT} \)
2) \( \mid \text{expr} \text{PLUS} \text{expr} \)
3) \( \mid \text{expr} \text{TIMES} \text{expr} \)
4) \( \mid \text{LPAREN} \text{expr} \text{RPAREN} \)
Grammars for expressions (cont.)

What about associativity? Consider $1 + 2 + 3$

**Definition: recursion in grammars**

A grammar is *recursive in non-terminal* $x$ if 
$x \Rightarrow + \alpha x \gamma$ for non-empty strings of symbols $\alpha$ and $\gamma$

A grammar is *left-recursive in non-terminal* $x$ if 
$x \Rightarrow + x \gamma$ for non-empty string of symbols $\gamma$

A grammar is *right-recursive in non-terminal* $x$ if 
$x \Rightarrow + \alpha x$ for non-empty string of symbols $\alpha$

**In expression grammars**

- for left associativity, use left recursion
- for right associativity, use right recursion

**Example**
List grammars

Example a list with no separators, e.g., A B C D E F G

Another ambiguous example

stmt $\rightarrow$ IF cond THEN stmt
| IF cond THEN stmt ELSE stmt
| . . .

Given this sequence in this grammar: if a then if b then s1 else s2
How would you derive it?