Programming Assignment 2
- due Friday, February 25

Homework 1 is available

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

Today
- Makefiles
- list grammars
- resolving ambiguity

Next Time
- wrap up CFGs
- syntax-directed translation

CFG review

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

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

\( \Rightarrow^+ \) means "derives in 1 or more steps"

language defined by a CFG \( G \)

\( L(G) = \{ w \mid s \Rightarrow^+ w \} \) where
- \( s \) = start is the start non-terminal of \( G \), an
- \( w \) = sequence consisting of (only) terminal symbols or \( \varepsilon \)
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
• used make to run commands:

Example

clean:
rm -f *.class
Programming Assignment 2

Modify:
- minim.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 minim.jlex.java
cleantest:
    rm -f allTokens.out

Running the tester

royal-30(51)% 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
Derivation order

1) prog $\rightarrow$ BEGIN stmts END

2) stmts $\rightarrow$ stmts SEMICOLON stmt
3) \quad | \quad stmt

4) stmt $\rightarrow$ ID ASSIGN expr

5) expr $\rightarrow$ ID
6) \quad | \quad expr PLUS ID

Leftmost derivation :

Rightmost derivation :
Expression Grammar Example

1) \( \text{expr} \rightarrow \text{INTLIT} \)
2) \( \mid \text{expr} \, \text{PLUS} \, \text{expr} \)
3) \( \mid \text{expr} \, \text{TIMES} \, \text{expr} \)
4) \( \mid \text{LPAREN} \, \text{expr} \, \text{RPAREN} \)

Derive: \( 4 + 7 \times 3 \)

For grammar \( G \) and string \( w \), \( G \) is \textbf{ambiguous} if there is
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) expr \rightarrow \text{INTLIT}
2) \mid \text{expr PLUS expr}
3) \mid \text{expr TIMES expr}
4) \mid \text{LPAREN expr 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
You try: add exponentiation (POW)

Add exponentiation (POW) to this grammar, with the correct precedence and associativity.

\[
\begin{align*}
\text{expr} & \rightarrow \text{expr} \text{ PLUS} \text{ term} \\
& \quad | \quad \text{term} \\
\text{term} & \rightarrow \text{term} \text{ TIMES} \text{ factor} \\
& \quad | \quad \text{factor} \\
\text{factor} & \rightarrow \text{INTLIT} \\
& \quad | \quad \text{LPAREN} \text{ expr} \text{ RPAREN}
\end{align*}
\]