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 = "phony"
- use make to run commands:

Example

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

test:
    java Example inFile.txt outFile.txt
    java Example inErrFile.txt outErrFile.txt
```

Programming Assignment 2

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

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
```

From running make:

```
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

$\Rightarrow^+$ 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

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

$L(G) = \varepsilon, (,), ((())), (((()))), \ldots$

Example: nested paren

$N = \varepsilon, 3$

$\Sigma = \varepsilon (, ) 3$

$P = q \Rightarrow (q)$

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

Sequence is: $()$
Parse tree example

Productions
1) prog → BEGIN stmts END
2) stmts → stmts SEMICOLON stmt
3)   | stmt
4) stmt → ID ASSIGN expr
5) expr → ID
6)   | expr PLUS ID

Derivation order

Productions
1) prog → BEGIN stmts END
2) stmts → stmts SEMICOLON stmt
3)   | stmt
4) stmt → ID ASSIGN expr
5) expr → ID
6)   | expr PLUS ID

Leftmost derivation:

Rightmost derivation:
Expression Grammar Example

1) expr → INTLIT
2)   | expr PLUS expr
3)   | expr TIMES expr
4)   | LPAREN expr RPAREN

Derive: 4 + 7 * 3

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

> 1 leftmost derivation of w

OR

> 1 rightmost derivation of w

OR

> 1 parse tree for w
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 → INTLIT
2) | expr PLUS expr
3) | expr TIMES expr
4) | LPAREN expr RPAREN

Try to make * eval’d last

IntLIT (3)
Grammars for expressions (cont.)

What about associativity? Consider $1 + 2 + 3 \equiv (1+2) + 3$

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

\[
\begin{align*}
\text{expr} & \rightarrow \text{expr} + \text{expr} \\
\text{term} & \rightarrow \text{term} \times \text{term} \\
\text{factor} & \rightarrow \text{INTLIT} \\
\text{INTLIT} & \rightarrow ( \text{expr} )
\end{align*}
\]
List grammars

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

\[
\begin{align*}
\text{alist} & \rightarrow \text{ITEM} \\
& \quad \vert \text{alist \ltimes \text{alist}} \\
\end{align*}
\]

Derive ABC

\[
\begin{align*}
\text{alist} & \rightarrow \text{ITEM} \\
& \quad \vert \text{alist} \rightarrow \text{ITEM} \land \text{alist} \\
\end{align*}
\]

Another ambiguous example

\[
\begin{align*}
\text{stmt} & \rightarrow \quad \text{IF cond THEN stmt} \\
& \quad \vert \quad \text{IF cond THEN stmt ELSE stmt} \\
& \quad \vert \quad \ldots
\end{align*}
\]

Given this sequence in this grammar: \text{if a then if b then s1 else s2}

How would you derive it?