Understanding Context Free Grammars

Give some legal strings accepted by this grammar?

E -> E + T | E – T | T
T -> T * F | T / F | F
F -> i

Give some legal strings accepted by this grammar?

S -> procedure id P ; | ε
P -> (L) | ε
L -> R : T | R : T ; L
R -> V D
V -> var | ε
D -> D , id | id
T -> int | real

Writing Context Free Grammars

Write a grammar for predicates within if conditions,
Where the condition is a predicate or logical expression

Example legal conditions:

x < y
(w == 8) and (j<10)
w > d or m == 9
not w
not (w < y)

From Regular Expression to Context Free Grammar

Regular Exprs:

b

RS

R|S

R*

CFG Production Rules:

Example Regular Expression:

I -> L (L | D) *

What is the equivalent context free grammar?

Derivations and Parse Trees

Consider the grammar:

S -> (L) | a

L -> L,S | S

Input string: (a, (a,a))

Question – Is this string in this language?

Important Terminology

Derivation –
Leftmost versus Rightmost derivation -
Sentential Form –
Sentence –
Parse Tree –
Concrete syntax Tree –
Abstract syntax Tree –
Ambiguous grammar -

The Chomsky Hierarchy
By Norman Chomsky 1959

Four Main Types of Grammars based on their form:

Type 0 – unrestricted grammars

Type 1 – context-sensitive grammars

Type 2 – context-free grammars

Type 3 – regular grammars

Format of a Bison Spec File

```
%{
Prologue  - macros
%}
Bison declarations
%%
Grammar rules and actions
%%
Epilogue – copied directly
```

Example Prologue

```
%{
#include <stdio.h>
#include "ptypes.h"
%}
%union {
long int n;
tree t; /* tree is defined in ‘ptypes.h’. */
}
%{
static void print_token_value (FILE *,
int, YYSTYPE);
#define YYPRINT(F, N, L)
print_token_value (F, N, L)
%}
```

Example Rules Section without actions

```
expr: primary
| primary '+' primary
;
primary: constant
| '(' expr ')'
;
```

Example Epilogue

```
int yyerror(char *msg)
{  fprintf(stderr,"Error: %s\n",msg);
   return 0;
}

int main(void)
{   yyparse();
    return 0;
}
```

A complete Example

A simple thermostat controller

Let's say we have a thermostat that we want to control using a simple language.

A session with the thermostat may look like this:

```
heat on
        Heater on!
heat off
        Heater off!
target temperature 22
        New temperature set!
```

The Lex Spec

```
%{
#include <stdio.h>
#include "y.tab.h"
%}
%%
[0-9]+ return NUMBER;
heat    return TOKHEAT;
on|off return STATE;
target return TOKTARGET;
temperature return TOKTEMPERATURE;
\n /* ignore end of line */;
[ \t]+ /* ignore whitespace */;
%%
```

The (incomplete) Bison Spec

```
commands: /* empty */
    | commands command ;

command: heat_switch
    | target_set ;

heat_switch: TOKHEAT STATE  { printf("\tHeat turned on or off\n"); } ;

target_set: TOKTARGET TOKTEMPERATURE NUMBER
                                        { printf("\tTemperature set\n"); } ;
```

The Makefile with Lex/Flex and Yacc/Bison

```
all: turtle eins.ps

eins.ps: eins.tlt turtle
	turtle < eins.tlt > eins.ps

lex.yy.o:  lex.yy.c turtle.tab.h symtab.h
	gcc -c lex.yy.c

lex.yy.c: turtle.l
	flex turtle.l

turtle.tab.h: turtle.y
	bison -d turtle.y

turtle.tab.c: turtle.y
	bison -d turtle.y

turtle.tab.o: turtle.tab.c symtab.h
	gcc -c turtle.tab.c

turtle: turtle.tab.o lex.yy.o symtab.c
	gcc lex.yy.o turtle.tab.o symtab.c -lfl -o turtle
```

Example Bison Spec with Actions

```
input:                          /* empty */
| input line
;
line: '\n'
| exp '\n'          { printf ("\t%.10g\n", $1); }
;
exp: NUM            { $$ = $1; }
| exp exp '+'       { $$ = $1 + $2; }
| exp exp '-'       { $$ = $1 - $2; }
| exp exp '*'       { $$ = $1 * $2; }
| exp exp '/'       { $$ = $1 / $2; }
/* Exponentiation */
| exp exp '^'       { $$ = pow ($1, $2); }
/* Unary minus */
| exp 'n'           { $$ = -$1; }
;
%%
```