CS310

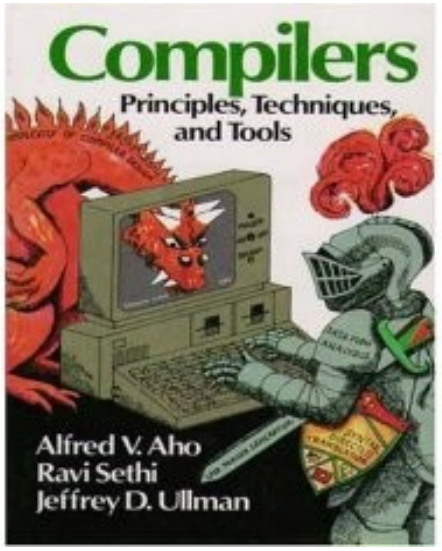
Parsing with Context Free Grammars

Today's reference:

Compilers: Principles, Techniques, and Tools

by: Aho, Sethi, Ullman

aka: The Dragon Book



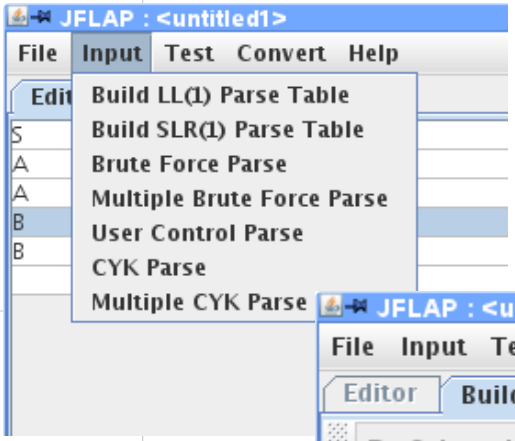
Section 4.4 – 4.8

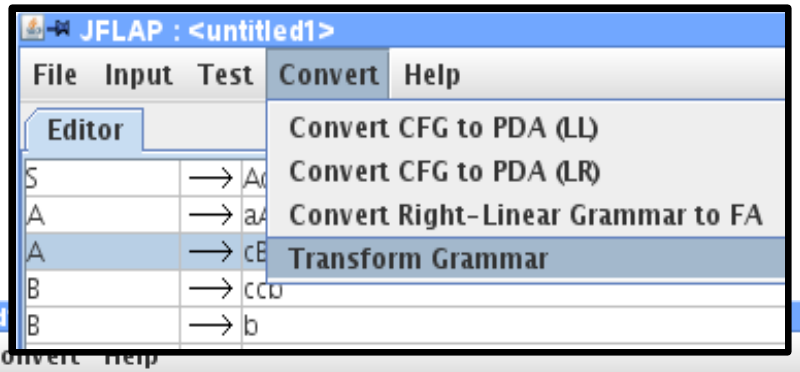
October 29, 2008

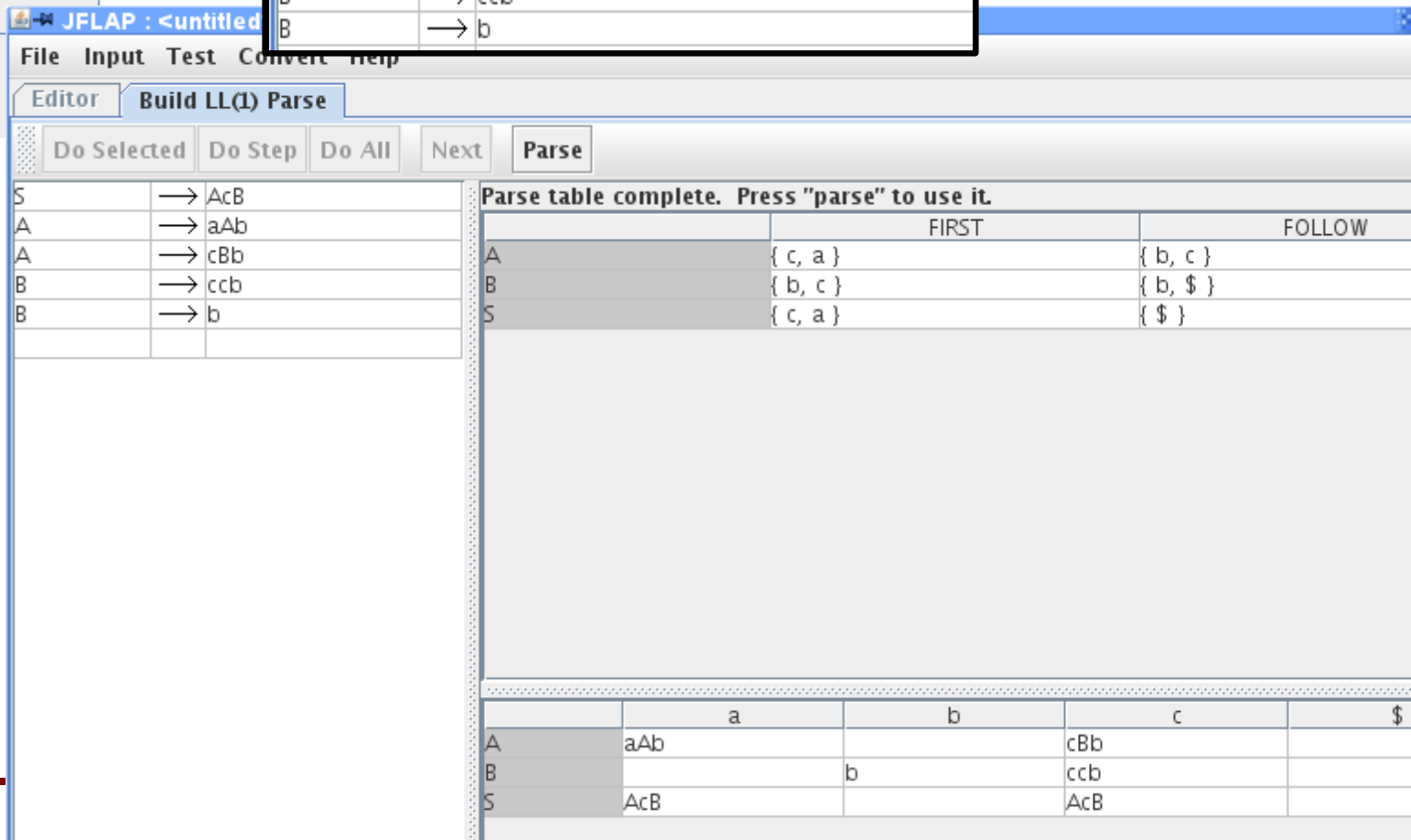
Parsing with JFLAP

- FIRST? FOLLOW? Parse Table?

(1) S -> AcB
(2) A -> aAb
(3) A -> cBb
(4) B -> ccb
(5) B -> b







LL(2) Parse Table

(1) S -> AcB
(2) A -> aAb
(3) A -> aBb
(4) B -> ccb
(5) B -> b

- Two lookahead symbols
  Is two enough for this grammar?
  Is one enough?

	aa	ab	ac	bb	ba	bc
S						
A						
B						

Bottom Up Parsing

- Shift-reduce parsing
  - used in many automatic parser generators,
- *Reduce* the string to the start symbol
- *Shift* a symbol from the string on to a stack

(1) E -> E + E
(2) E -> E * E
(3) E -> (E)
(4) E -> x

Stack	Input	Action
$	x + x * x $	shift
$ x	+ x * x $	reduce
$ E		

Shift/Reduce Conflict

stmt → IF expr THEN stmt
| IF expr THEN stmt ELSE stmt
| somethingelse

Stack	Input	Action
... $ IF expr THEN stmt	ELSE .... $	???

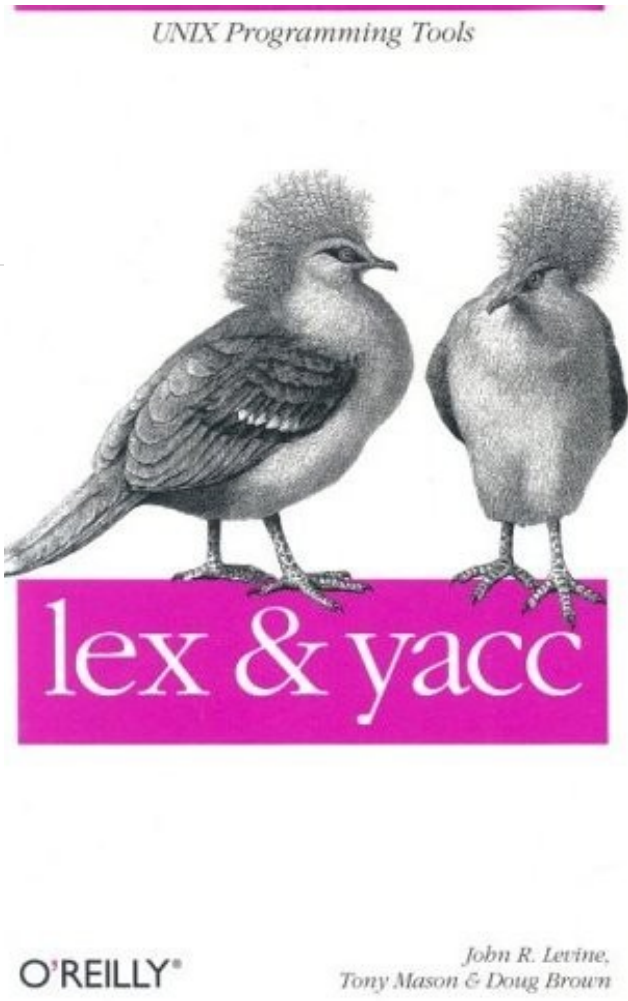
CS310

Lex & Yacc

Today's reference:
UNIX Programming Tools:
lex & yacc

by: Levine, Mason, Brown

Chapter 1, 2, 3



Automatic Parser Generators

- You supply the tokenizer
- You supply the grammar

- Output source code that will produce a parser
  - decorate the grammar rules with source code to perform various tasks

Lex (Flex*)

- Lex:
  - specify the terminals or regular expressions that represent the terminals
  - produces a C source file as output that will divide input into tokens

```
$ flex file.l
$ gcc -lfl -o file lex.yy.c
$ ./file
```

* Flex and Bison are the GNU equivalents of Lex and Yacc

```
%{
     /* identify is and are as verbs */
%}

%%

[\t ]+     /* ignore whitespace */
is |
are        { printf("%s: is a verb", yytext); }
[a-zA-Z]+ { printf("%s: not a verb", yytext);}

%%

main()
{
     yylex();
}
```

```
%{
      /* count verbs and nonverbs */
      int verbCount=0, nonVerbCount=0;
%}
%%

[\t ]+        /* ignore whitespace */


is |
are           { printf("%d verbs\n", ++verbCount);  }
[a-zA-Z]+     { printf("%d non-verbs\n", ++nonVerbCount);}

%%

main()
{
  yylex();
  printf("%d verbs| %d nonverbs\n", verbCount,
      nonVerbCount);
}
```

Yacc (Bison)

- Yacc:
  - take the tokens (terminals) from lex and apply a grammar
  - check syntax

  - file.tab.c contains the C code to apply the grammar
  - file.tab.h contains the data structures to be used by lex to pass data to yacc

```
$ bison file.y
```

```
%{
#include “yaccfile.tab.h”
int lineno=1;
%}

%%

[\t ]+              /* ignore whitespace */
a                   { return(LITTLEA); }
b                   { return(LITTLEB); }
c                   { return(LITTLEC); }
\n                  { lineno++; return(END);}
[a-zA-Z0-9]+        { return(ERROR); }


%%
```

```
%{
#include <stdio.h>
extern FILE *yyin; extern int lineno; extern char* yytext;
%}
%token LITTLEA LITTLEB LITTLEC END ERROR
%%
start:      BIGA LITTLEC BIGB END
      { printf("\n\tThat string is accepted!", yytext); }

BIGA: LITTLEA BIGA LITTLEB
    | LITTLEC BIGB LITTLEB
BIGB: LITTLEC LITTLEC LITTLEB
    | LITTLEB
%%
int yyerror(char *msg){
      printf("ERROR: (%d:%s) %s\n",lineno, yytext, msg);
}
main(){
      do{
            yyparse();
      }while(!feof(yyin));
}
```

```
%{
#include “yaccfile.tab.h”
int lineno=1;
%}

%%

[\t ]+     /* ignore whitespace */
is |
are              { return(VERB); }
computer |
bob |
alice            { return(NOUN);}

\n               { lineno++; return(END);}
[a-zA-Z0-9]+     { return(ERROR); }

%%
```

```
%{
#include <stdio.h>
extern FILE *yyin; extern int lineno; extern char* yytext;
%}
%token VERB NOUN END ERROR

%%

sentence: nounphrase verbphrase END
nounphrase: NOUN { printf("noun %s\n",yytext); }
verbphrase: VERB { printf("verb %s\n",yytext); }

%%
main(){
      do{
            yyparse();
      }while(!feof(yyin));
}
int yyerror(char *msg){
      printf("ERROR: (%d:%s) %s\n",lineno, yytext, msg);
}
```

Build the executable

```
bison –dv yaccfile.y
flex lexfile.l
gcc –o parser lex.yy.c yaccfile.tab.c -lfl
```

```
./parser
```

control-C to exit the parser

Shift/Reduce Conflicts

```
stmt:   IF expr THEN stmt
      | IF expr THEN stmt ELSE stmt
```

Solutions:

- Rewrite the grammar. (*.y file)
- Use **precedence** to tell the parser how to handle this

```
%nonassoc LOWER_THAN_ELSE
%nonassoc ELSE

stmt: IF expr THEN stmt %prec LOWER_THAN_ELSE
      | IF expr THEN stmt ELSE stmt
```