CS 4300: Compiler Theory

Chapter 4
Syntax Analysis

Dr. Xuejun Liang

Outlines (Sections)

1. Introduction
2. Context-Free Grammars
3. Writing a Grammar
4. Top-Down Parsing
5. Bottom-Up Parsing
6. Introduction to LR Parsing: Simple LR
7. More Powerful LR Parsers
8. Using Ambiguous Grammars
9. Parser Generators

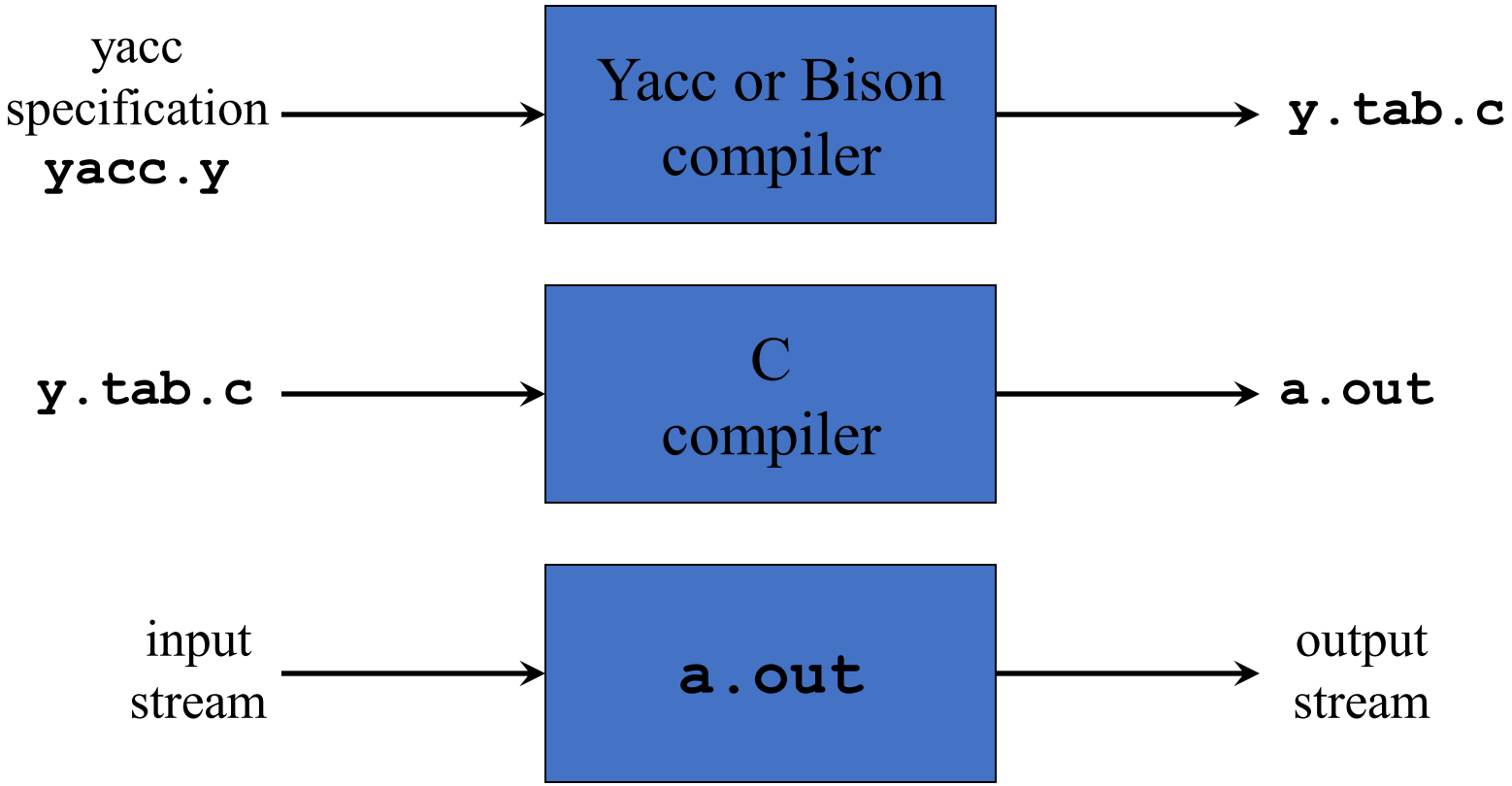
Quick Review of Last Lecture

- LALR Parsing (Look-Ahead LR)
  - Constructing LALR Parsing Tables
- LL, SLR, LR, LALR Summary
- Dealing with Ambiguous Grammars
  - Using Associativity and Precedence to Resolve Conflicts
- Error Detection and Recovery in LR Parsing

9. Parser Generator Yacc and Bison

- *Yacc* (Yet Another Compiler Compiler)
  - Generates LALR parsers
- *Bison*
  - Improved version of Yacc

Creating an LALR(1) Parser with Yacc/Bison



Yacc Specification

- A *yacc specification* consists of three parts:

  yacc declarations, and C declarations within **%{ %}**

  %%

  translation rules

  %%

  user-defined auxiliary procedures

- The *translation rules* are productions with actions:

  $production_1$ { $semantic\ action_1$ }
  $production_2$ { $semantic\ action_2$ }
  …
  $production_n$ { $semantic\ action_n$ }

Writing a Grammar in Yacc

- Productions in Yacc are of the form

  Nonterminal **:** tokens/nonterminals { *action* }
  | tokens/nonterminals { *action* }
  …
  ;

- Tokens that are single characters can be used directly within productions, e.g. **'+'**
- Named tokens must be declared first in the declaration part using

  %token *TokenName*

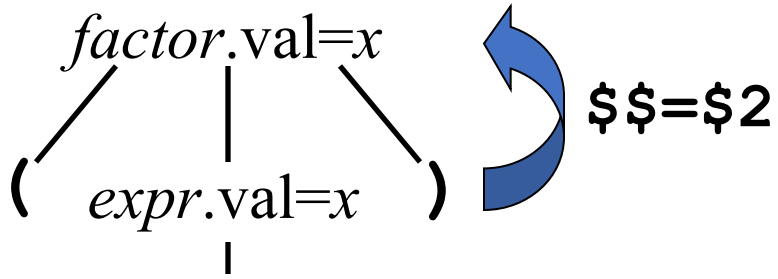
Synthesized Attributes

- Semantic actions may refer to values of the *synthesized attributes* of terminals and nonterminals in a production:

  $X : Y_1\ Y_2\ Y_3 \ldots Y_n \qquad \{\ action\ \}$

  - **`$$`** refers to the value of the attribute of X
  - **`$`***i* refers to the value of the attribute of Y_i
- For example

  `factor : '(' expr ')' { $$=$2; }`



Example 1

```
%{ #include <ctype.h> %}
%token DIGIT
%%
line    : expr '\n'              { printf("= %d\n", $1); }
        ;
expr    : expr '+' term          { $$ = $1 + $3; }
        | term                   { $$ = $1; }
        ;
term    : term '*' factor        { $$ = $1 * $3; }
        | factor                 { $$ = $1; }
        ;
factor  : '(' expr ')'           { $$ = $2; }
        | DIGIT                  { $$ = $1; }
        ;
%%
int yylex()
{ int c = getchar();
  if (isdigit(c))
  { yylval = c-'0';
    return DIGIT;
  }
  return c;
}
```

Also results in definition of **`#define DIGIT xxx`**

Attribute of **`factor`** (child)

Attribute of **`term`** (parent)

Attribute of token (stored in **`yylval`**)

Example of a very crude lexical analyzer invoked by the parser

Dealing With Ambiguous Grammars

- By defining operator precedence levels and left/right associativity of the operators, we can specify ambiguous grammars in Yacc, such as
  $E \rightarrow E$**+**E | E**-**E | E*****E | E**/**E | **(**E**)** | **-**E | **num**
- To define precedence levels and associativity in Yacc's declaration part:

```
%left '+' '-'
%left '*' '/'
%right UMINUS
```

Example 2

```
%{
#include <ctype.h>
#include <stdio.h>
#define YYSTYPE double
%}
%token NUMBER
%left '+' '-'
%left '*' '/'
%right UMINUS
%%
lines   : lines expr '\n'        { printf("= %g\n", $2); }
        | lines '\n'
        | /* empty */
        ;
expr    : expr '+' expr          { $$ = $1 + $3; }
        | expr '-' expr          { $$ = $1 - $3; }
        | expr '*' expr          { $$ = $1 * $3; }
        | expr '/' expr          { $$ = $1 / $3; }
        | '(' expr ')'           { $$ = $2; }
        | '-' expr %prec UMINUS  { $$ = -$2; }
        | NUMBER
        ;
%%
```

Double type for attributes and **`yylval`**

Example 2 (cont'd)

```
%%
int yylex()
{ int c;
  while ((c = getchar()) == ' ')
    ;
  if ((c == '.') || isdigit(c))
  { ungetc(c, stdin);
    scanf("%lf", &yylval);
    return NUMBER;
  }
  return c;
}
int main()
{ if (yyparse() != 0)
    fprintf(stderr, "Abnormal exit\n");
  return 0;
}
int yyerror(char *s)
{ fprintf(stderr, "Error: %s\n", s);
}
```

Crude lexical analyzer for fp doubles and arithmetic operators

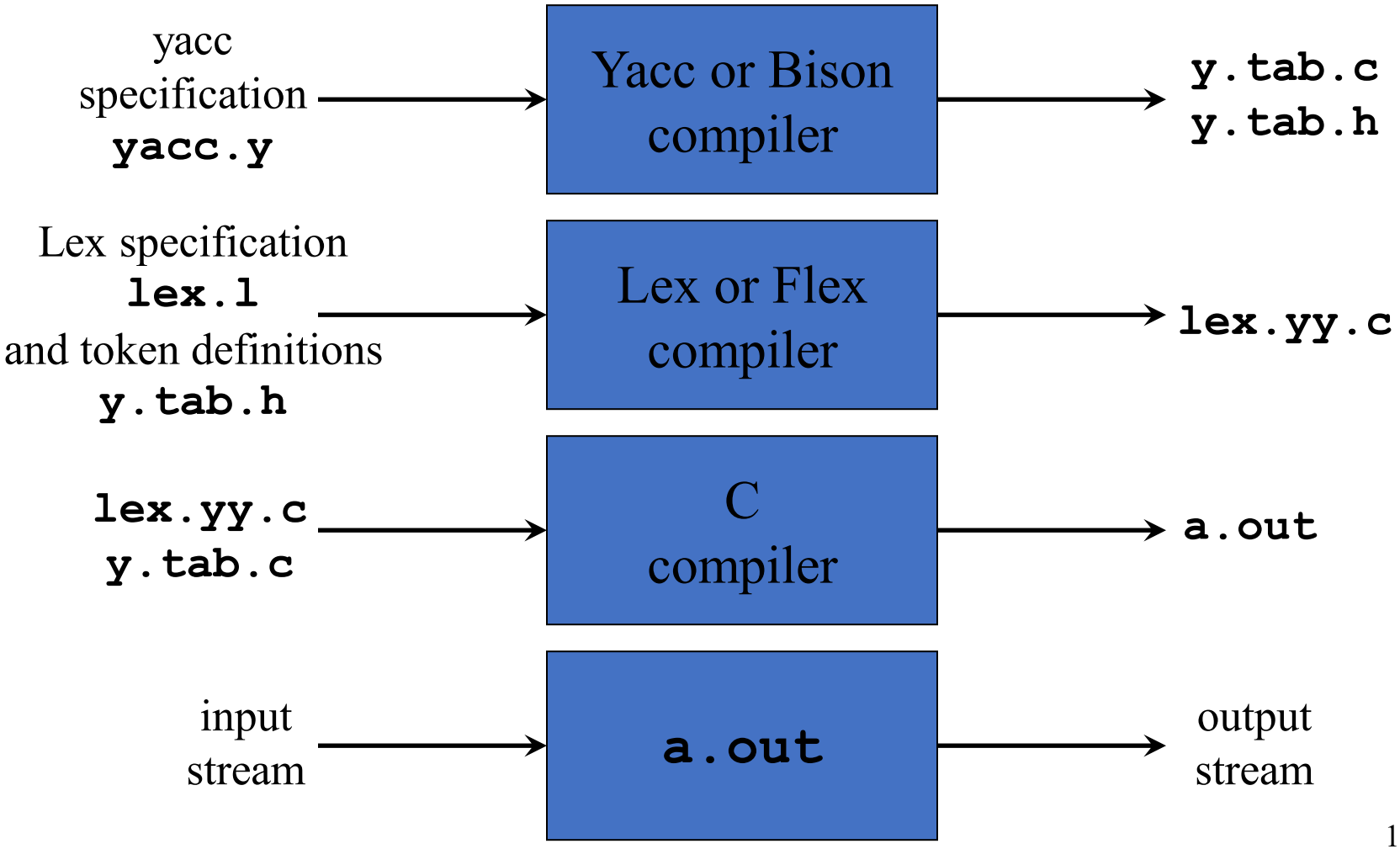
Run the parser

Invoked by parser to report parse errors

Resolve Parsing Action Conflicts

- Two default rules
  - A reduce/reduce conflict is resolved by choosing the conflicting production listed first
  - A shift/reduce conflict is resolved in favor of shift.
- Using precedence and associativity to resolve a shift/ reduce conflict between shifting input symbol **a** and reducing by production A $\rightarrow\alpha$
  - Reduces if the precedence of the production is greater than that of a, or if the precedences are the same and the associativity of the production is left
  - Otherwise, shift

Combining Lex/Flex with Yacc/Bison



Lex Specification for Example 2

```
%option noyywrap
%{
#define YYSTYPE double
#include "y.tab.h"

extern double yylval;
%}
number  [0-9]+\.?|[0-9]*\.[0-9]+
%%
[ ]             { /* skip blanks */ }
{number}        { sscanf(yytext, "%lf", &yylval);
                  return NUMBER;
                }
\n|.            { return yytext[0]; }
```

Generated by Yacc, contains **`#define NUMBER xxx`**

Defined in **`y.tab.c`**

```
yacc -d example2.y
lex example2.l
gcc y.tab.c lex.yy.c
./a.out
```

```
bison -d -y example2.y
flex example2.l
gcc y.tab.c lex.yy.c
./a.out
```

Error Recovery in Yacc

```
%{
…
%}
…
%%
lines   : lines expr '\n'       { printf("%g\n", $2; }
        | lines '\n'
        | /* empty */
        | error '\n'            { yyerror("reenter last line: ");
                                  yyerrok;
                                }
        ;
…
```

Error production:
set error mode and
skip input until newline

Reset parser to normal mode

Yacc Programming Assignment

- P297 Exercises for Section 4.9
  - Exercise 4.9.1
- Instructions for using our project server and submitting your assignments.
  1. Make a new directory called PAYL
  2. Navigate into the directory PAYL and type

     make -f /home/CS4300/assignments/PAYL/Makefile

     It will copy a README file in the directory and setup for your assignment submission. Please read the README file.
  3. Your Yacc source file name must be called bexpr.y and your Lex source file name must be called bexpr.l. They should be inside the directory PAYL.
  4. To submit your files, under the directory PAYL, type

     make submit