Lecture 4.
How to Write a Netlist Parser?

Guoyong Shi, PhD

shiguoyong@ic.sjtu.edu.cn

School of Microelectronics

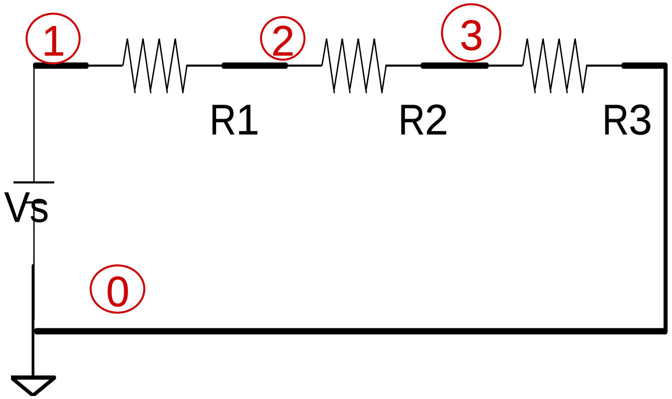
Shanghai Jiao Tong University

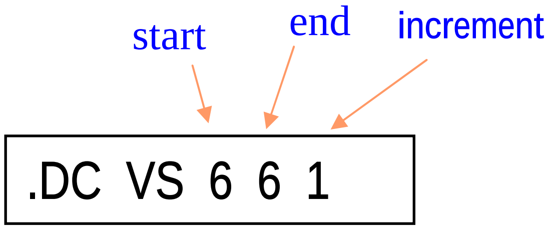
Fall 2010

Outline

- Spice Netlist
- Netlist Parsing
- Parser Principle
- Flex and Bison
- Spice Netlist Grammar
- PCCTS
- Assignment 2 (parser)

A Netlist Example




start

end

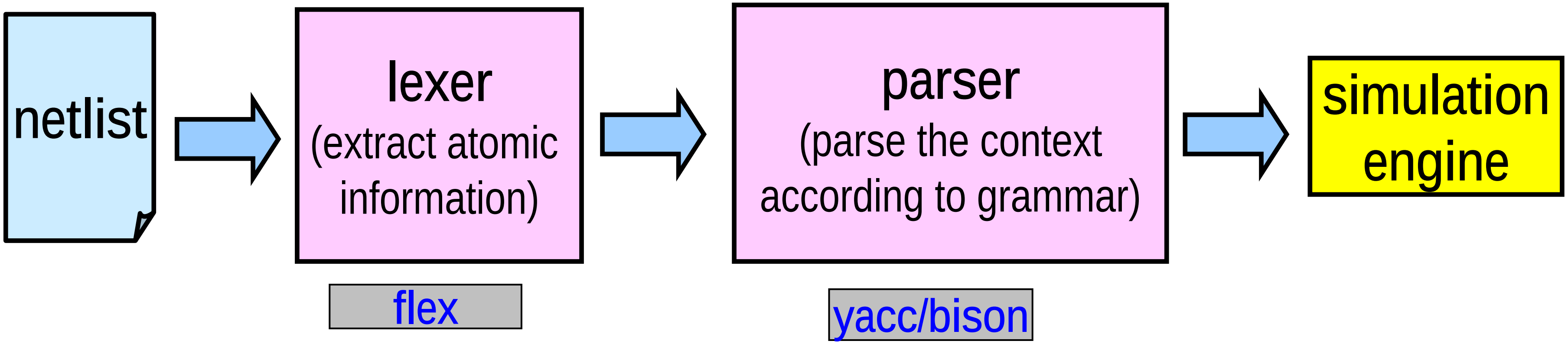
increment

.DC VS 6 6 1

```
* SERIES CIRCUIT (comment line)
VS  1  0  5
R1  1  2  1K
R2  2  3  2K
R3  3  0  3K
.DC   VS   6   6   1
.PRINT DC V(2,3)  V(2)  I(R2)
.END
```

Parser Principle

- A parser is used to extract structural information from a text file.
- A netlist has a simple grammar that defines the meaning of the circuit components.



netlist

lexer
(extract atomic information)

parser
(parse the context according to grammar)

simulation engine

flex

yacc/bison

Lexical Analysis

- Regular Expressions (RE)
  - An expression is a string of characters
  - RE is a set of chars or meta-chars
  - REs are used for text searching or string matching

LF	[\n]
DELIM	[\t]
WS	{DELIM}+
ALPH	[A-Za-z_]
DIGIT	[0-9]
ALPH_NUM	{ALPH}|{DIGIT}
INTEGER	{DIGIT}+
FLOAT	… …
SIGN	“+”|”-”
… …	

Lexical Analysis

- **The first line of a Spice netlist is always treated as a comment line**
- **([^\n])* -- any number of chars ("^") excluding <newline>**
- **"v"{ALPH_NUM}* -- a name string starting with "v"**
  - **defines a V_ELEMENT**
- **"r"{ALPH_NUM}* -- a name string starting with "r"**
  - **defines an R_ELEMENT**

Flex

- **A fast lexical analyzer generator**
  - **http://www.gnu.org/software/flex/manual**
- **Compile**

```
% flex filename.lex
% flex –i filename.lex   (case-insensitive scanner)
```

- **Flex is still under development, see**
  - **The Flex Project: http://flex.sourceforge.net/**
  - **for the latest source code and documentation**

Flex Input File

- **Input file format**

  Definitions – defining string pattern names

  %%

  Rules – in pairs of [<matching pattern> <action>]

  %%

  User Code – copied verbatim to “lex.yy.c”.

  -- containing routines called by the action part

Grammar 1

- Suppose we'd like to process the expression
  - x1 = (1+2)*3;
- This is an arithmetic expression, and can be evaluated.
- Suppose our expressions are allowed to have:
  - +, -, (,), =
  - NUM (integer numbers)
  - ; (each expr ended by semicolon)
- Such expressions can be described by the following grammar:
  - (next page)

Grammar 2

```
L → ID = E; L | empty
E → E add_op T | T
T → T mul_op F | F
F → NUM | ( E )
```

- **ID is an identifier (variable) for storing the expression value.**
- **add_op & mul_op are operators "+" & "-".**
- **The symbol "|" reads like "OR".**
- **The symbol "→" reads like "substitution": LHS is substituted by RHS.**
- **The 4 rules define a grammar structure.**

Grammar 3

- L → ID = E; L | empty
- This means we can have multiple expressions in the same line,separated by ";". For example,
- x1 = 1 + 2; x2 = 2 * (3 + 4);

- A grammar looks like recursion. The "L" on the RHS of "→" can be substituted recursively by the mapping, until the point "L = empty" is reached.

Grammar 4

E → E add_op T | T

T → T mul_op F | F

F → NUM | (E)

higher priority

- These 3 lines define the expression structure.
- The line order is important; it specifies the computation priority.
  - Multiplication has the higher priority than Addition.

- The line at the bottom usually specifies the atomic expression; i.e., cannot be decomposed further.

Bison

- A general-purpose parser generator
- Converts an annotated context-free grammar in an LALR(1) parser or GLR parser
- Can be used to develop language parsers
  - from simple desk calculators
  - to complex programming languages
- http://www.gnu.org/software/bison/
- Upward compatible with Yacc

Flex & Bison

- **Bison normally is used together with flex**
  - **flex as a lexical analyzer**
  - **bison as a grammar analyzer**
- **bison and flex are available in** *cygwin*

- **Create a flex file, say,** *example.lex*
  - **%flex example.lex (➔ lex.yy.c)**
- **Create a grammar-action file, say,** *example.y*
- **Compile**
  - ***% bison –d example.y***
  - **[-d] forces to generate** *example.tab.h* **&** *example.tab.c*

Bison Input File

- The input file for bison (“.y” file) is a grammar file.
- It mainly has three sections:

%{

C declarations -- copied verbatim

%}

Bison declarations

%%

Grammar rules -- netlist grammar is parsed here

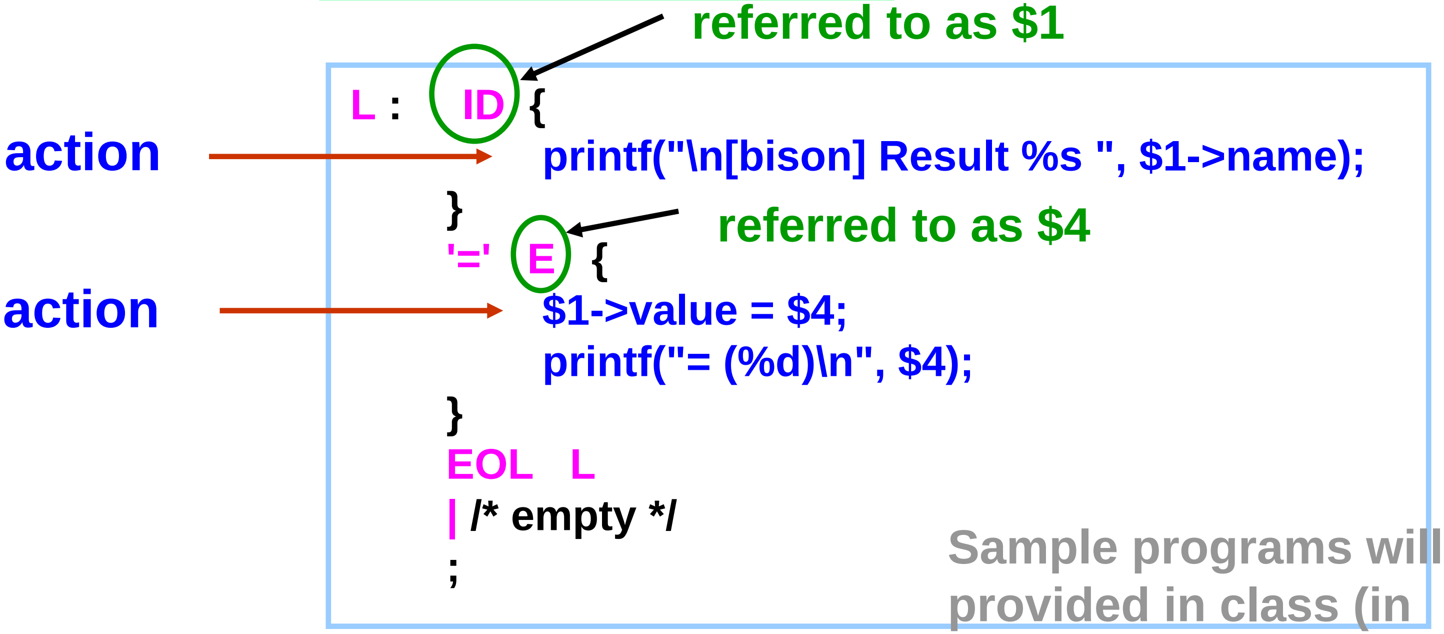
%%

Additional C code

Grammar Parsing (Bison)

- **The "Grammar Rules" section is the place where the actions are taken for the structural elements that match the grammar.**

L → ID = E; L | empty



referred to as $1

referred to as $4

action

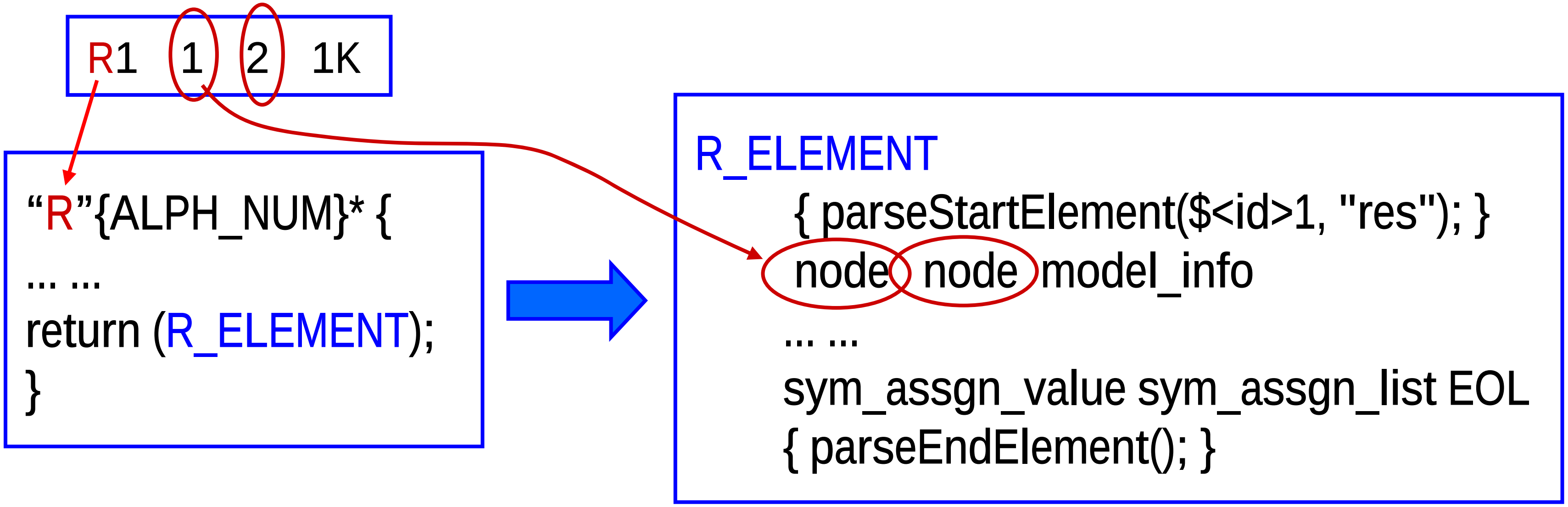
action

```
L :   ID {
           printf("\n[bison] Result %s ", $1->name);
      }
      '=' E {
           $1->value = $4;
           printf("= (%d)\n", $4);
      }
      EOL  L
      | /* empty */
      ;
```

Sample programs will be provided in class (in parser_drill/...)

Flex talks to Bison

Communication between *Flex* and *Bison*



R1 1 2 1K

```
"R"{ALPH_NUM}* {
... ...
return (R_ELEMENT);
}
```

```
R_ELEMENT
     { parseStartElement($<id>1, "res"); }
     node node model_info
     ... ...
     sym_assgn_value sym_assgn_list EOL
     { parseEndElement(); }
```

in flex file
"parse.lex"

in bison file
"parse.y"

Linking

- **When linking with object files lex.yy.o, xxx.tab.o, use**

.... `-lfl` ... **(in cygwin/Linux ...)**

flex library

Otherwise, you'll see error:

... undefined reference to `_yywrap'

PCCTS

- **PCCTS**
  - **Purdue Compiler Construction Tool Set**
  - **by Terence John Parr (PhD Purdue, 1993)**
  - **A C++ parser generator**
  - **Open source, well documented**
  - **Find it by going to Google**
- **ANTLR**
  - **ANother Tool for Language Recognition**
  - **A parser generator in PCCTS**
  - **First released 1992**
- **Terence John Parr, *Language Translation using PCCTS and C++ (A Reference Guide)*, Automata Publishing Company, San Jose, CA 95129.**

Assignment 2 (parser)

This assignment is for on-line learning.

- Go to Internet, find some learning materials about flex & bison.
- Do some flex/bison exercises on CYGWIN or your Linux installation.
- Write a report on what you have done, including some programs you have tried.
- You can attempt to write a netlist parser by printing out what is parsed.
- Turn in your report to Moodle (or to TA) within a week.

References

1. T. J. Parr, Language Translation using PCCTS and C++, A Reference Guide, 1993.
2. Online materials on compiler tools.

Acknowledgement

- **Contributors to the open source software tools used in this lecture are greatly acknowledged.**