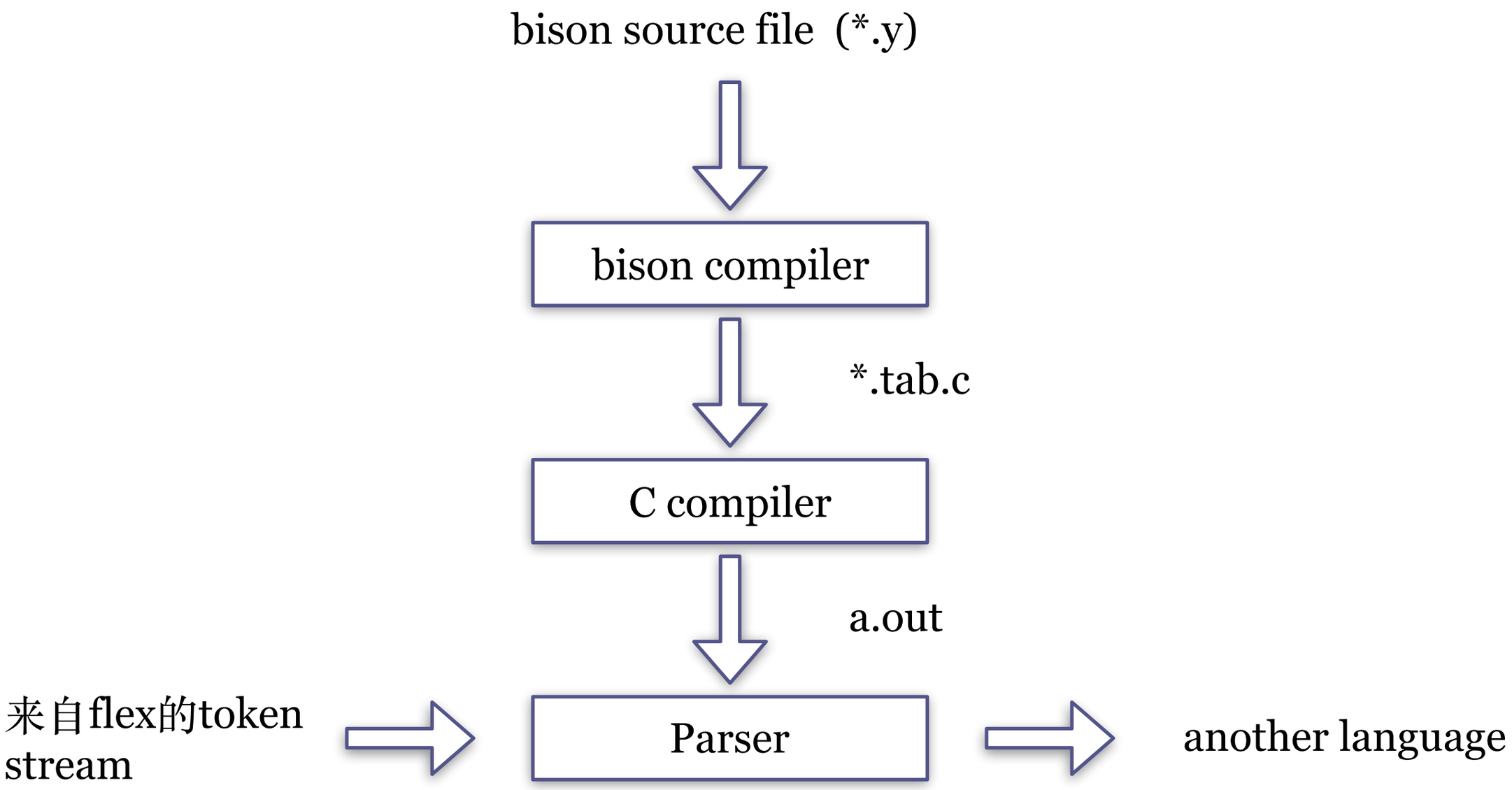
Bison 入门

编译原理

Bison简介

- Bison 是 yacc(Yet Another Compiler Compiler)的GNU版
- 生成语法分析器的工具
- 将语法描述转换为LALR(1)的context-free的C语言语法分析器

Bison工作原理


bison source file (*.y)
bison compiler
*.tab.c
C compiler
a.out
来自flex的token stream
Parser
another language

Bison 程序格式

Definitions：C和Bison的全局声明

%%

Grammar：语法规则

%%

UserCode：补充的C代码，一般包含main函数

Bison 程序格式

Definitions：C和Bison的全局声明

%%

Grammar：语法规则

%%

UserCode：补充的C代码，一般包含main函数

```
1   /* Infix notation calculator. */
2
3   %{
4       #include <stdio.h>
5       #include <math.h>
6       void yyerror (char const *);
7   %}
8
9   %token NUM
10  %left  '-' '+'
11  %left  '*' '/'
12  %right '^'        /* exponentiation */
13
14  %%
15  /* Grammar rules and actions follow.  */
16
17  input : /* empty */
18        | input line
19  ;
20
21  line  : '\n'
22        | exp '\n'              { printf("\t=> %d\n", $1); }
23  ;
24
25  exp   : NUM                   { $$ = $1;          }
26        | '(' exp ')'           { $$ = $2;          }
27        | exp '+' exp           { $$ = $1 + $3;     }
28        | exp '-' exp           { $$ = $1 - $3;     }
29        | exp '*' exp           { $$ = $1 * $3;     }
30        | exp '/' exp           { $$ = $1 / $3;     }
31        | exp '^' exp           { $$ = pow($1, $3); }
32  ;
33
34  %%
35
36  int main()
37  {
38      return yyparse();
39  }
40
41  /* Called by yyparse on error.  */
42  void yyerror (char const *s)
43  {
44      fprintf (stderr, "%s\n", s);
45  }
46
```

Bison 程序格式: Definitions

```
1  /* Infix notation calculator. */
2
3  %{
4        #include <stdio.h>
5        #include <math.h>
6        void yyerror (char const *);
7  %}
8
9  %token NUM
10 %left  '-' '+'
11 %left  '*' '/'
12 %right '^'         /* exponentiation */
13
14 %%
```

Bison 程序格式: Definitions

- C语言代码
  - 包含C代码的注释、头文件、宏定义
  - C代码必须由 %{ 与 %} 包围，会被原封不动的复制到生成的 *.tab.c 文件中

Bison 程序格式: Definitions

- Bison 全局声明与选项
  - %token　声明TOKEN，是终结符
    - Ex: %token　NUM
  - %left　　声明运算符为左结合性
    - Ex: %left ‘+’ ‘-’
  - %right　声明运算符为右结合性
    - Ex: %right ‘^’

Bison：YYSTYPE

- YYSTYPE 是Bison中终结符和非终结符值的类型
- 未定义时默认生成为 int 类型
- 有两种定义方式：
  - #define YYSTYPE double（在C语言声明中）
  - %union {
    int iType;
    double dType;
    char cType;
  - }（在Bison声明中）

Bison : YYSTYPE

```
%union {
    int iType;
    double dType;
    char cType;
}
```

在 Flex 中：（yylval的类型为YYSTYPE)

```
[0-9]+   { yylval.iType =  atoi(yytext); return NUM; }
[+-*/]   { yylval.cType = *yytext; return OP; }
```

Bison : YYSTYPE

```
%union {
    int iType;
    double dType;
    char cType;
}
```

在 Bison 中:

```
expr : NUM
     | expr '+' expr { $$ = $1 + $3 ;}
;
```

```
expr : NUM
     | expr '+' expr { $<iType>$ = $<iType>1 + $<iType>3 ;}
;
```

Bison : YYSTYPE

```
%union {
    int iType;
    double dType;
    char cType;
}
```

在 Bison 中:

```
%token  <iType>  NUM     (针对终止符)
%type   <iType>  exp     (针对非终止符)

%%
exp :  NUM
     | exp  '+'  exp  { $$ = $1 + $3 ;}
;
```

Bison 程序格式: Grammar Rules

```
14  %%
15  /* Grammar rules and actions follow.  */
16
17  input : /* empty */
18        | input line
19  ;
20
21  line  : '\n'
22        | exp '\n'            { printf("\t=> %d\n", $1); }
23  ;
24
25  exp   : NUM                 { $$ = $1;          }
26        | '(' exp ')'         { $$ = $2;          }
27        | exp '+' exp         { $$ = $1 + $3;     }
28        | exp '-' exp         { $$ = $1 - $3;     }
29        | exp '*' exp         { $$ = $1 * $3;     }
30        | exp '/' exp         { $$ = $1 / $3;     }
31        | exp '^' exp         { $$ = pow($1, $3); }
32  ;
33
34  %%
```

Bison 程序格式: Grammar Rules

- 类似于产生式
- 包括非终止符、终止符和动作
- 形式：
```
Non-terminal : Rule-Components1  {Action}
             |  Rule-Components2 {Action}
             |  Rule-Components3 {Action}
             ;
```

Bison 程序格式: Grammar Rules

```
25  exp    : NUM                { $$ = $1;          }
26         | '(' exp ')'        { $$ = $2;          }
27         | exp '+' exp        { $$ = $1 + $3;     }
28         | exp '-' exp        { $$ = $1 - $3;     }
29         | exp '*' exp        { $$ = $1 * $3;     }
30         | exp '/' exp        { $$ = $1 / $3;     }
31         | exp '^' exp        { $$ = pow($1, $3); }
32  ;
```

Bison 程序格式: User Code

```
34  %%
35
36  int main()
37  {
38      return yyparse();
39  }
40
41  /* Called by yyparse on error.  */
42  void yyerror (char const *s)
43  {
44      fprintf (stderr, "%s\n", s);
45  }
46
```

Bison 程序格式: User Code

- Bison 对此部分不做任何处理
- 直接拷贝到 *.tab.c 的末尾
- 包括:
  - 用户自定义函数
  - main函数

与flex的配合使用

- flex生成词法分析器，bison生成语法分析器
- bison的yyparse()每次调用flex的yylex()会获得一个token

与flex的配合使用

```
%{
    #include <stdlib.h>
    #include "cal.tab.h"
%}

%%

[0-9]+          { yylval = atoi(yytext); return NUM; }
"+"             return '+';
"-"             return '-';
"*"             return '*';
"/"             return '/';
"^"             return '^';
"("             return '(';
")"             return ')';
"\n"            return '\n';
[ \t]+          ;       /* ignore whitespace */

.               yyerror("Unknown character");

%%

int yywrap(void) {
    return 1;
}
```

```
/* Infix notation calculator. */

%{
    #include <stdio.h>
    #include <math.h>
    void yyerror (char const *);
%}

%token NUM
%left  '-' '+'
%left  '*' '/'
%right '^'       /* exponentiation */

%%
/* Grammar rules and actions follow.  */

input : /* empty */
      | input line
;

line  : '\n'
      | exp '\n'            { printf("\t=> %d\n", $1); }
;

exp   : NUM                 { $$ = $1;          }
      | '(' exp ')'         { $$ = $2;          }
      | exp '+' exp         { $$ = $1 + $3;     }
      | exp '-' exp         { $$ = $1 - $3;     }
      | exp '*' exp         { $$ = $1 * $3;     }
      | exp '/' exp         { $$ = $1 / $3;     }
      | exp '^' exp         { $$ = pow($1, $3); }
;

%%

int main()
{
    return yyparse();
}

/* Called by yyparse on error.  */
void yyerror (char const *s)
{
    fprintf (stderr, "%s\n", s);
}
```

与flex的配合使用

```
%{
    #include <stdlib.h>
    #include "cal.tab.h"
%}

%%

[0-9]+          { yylval = atoi(yytext); return NUM; }
"+"             return '+';
"-"             return '-';
"*"             return '*';
"/"             return '/';
"^"             return '^';
"("             return '(';
")"             return ')';
"\n"            return '\n';
[ \t]+          ;       /* ignore whitespace */

.               yyerror("Unknown character");

%%

int yywrap(void) {
    return 1;
}
```

注意

yylval 在bison编译时加参数-d时，内部会自动生成声明。

作用：存储token的值, 使共享给bison。

Bison Conflicts

- 解决方法:
  - 移入优先 （规约和移入冲突）
  - 先定义的规则优先 （规约与规约冲突）

移入/规约冲突

```
if_stmt:
   "if" expr "then" stmt
| "if" expr "then" stmt "else" stmt
;
```

- 移入优先
- 优先选择 第二条， 因为第二条是准备移入 “else”.

移入/规约冲突

- 1-2*5

- 未规定运算优先级时，会先算减法再算乘法

移入/规约冲突

- 1 - 2 * 5
- %left ‘-’
- %left ‘*’
- 这里，先写的优先级低于后写的，即减法低于乘法

规约/规约冲突

- E -> id
- T - > id

- 这里的两个产生式，都表达了id可以规约为他们。
- 在Bison中，先定义的语法规则优先。
- E： id {Action1};
- T： id {Action2};
- 这里id优先规约为E

安装部署

- Linux：（已内置）
- Mac：（已内置）

编译执行

- 生成语法分析器sample.tab.c：
- > bison -d sample.y （-d 带上会生成sample.tab.h)
- 生成词法分析器lex.yy.c：
- > lex sample.flex
- 编译c代码：
- > gcc lex.yy.c sample.tab.c -o sample.out -ly -ll
- 执行可执行文件：
- > ./sample.out

演示

Thanks