西安交通大学
XI'AN JIAOTONG UNIVERSITY

2022-2023 学年第二学期

《编译器设计专题实验》

实验报告 6

学　　院：电信学部

班　　级：[redacted]

学　　号：[redacted]

姓　　名：[redacted]

二〇二三年 五月

目录

《实验 6-语法分析(二),实现 COOL 语法分析》

一、实验内容（必做）

完成 PA3 的 COOL 语法分析。

二、实验结果

(1) 测试只有 program 和 class 的程序

文法如下所示：

$$
\begin{aligned}
\mathit{program} &::= [\![\mathit{class};]\!]^{+} \\
\mathit{class} &::= \textbf{class TYPE [inherits TYPE] \{ } [\![\mathit{feature};]\!]^{*} \textbf{\}}
\end{aligned}
$$

yacc 代码：

```
program : class_list
                { ast_root = program($1); }
        ;

class_list
        : class                 /* single class */
                { $$ = single_Classes($1);
                  parse_results = $$; }
        | class_list class      /* several classes */
                { $$ = append_Classes($1,single_Classes($2));
                  parse_results = $$; }
        ;

/* If no parent is specified, the class inherits from the Object class. */
class   : CLASS TYPEID '{' dummy_feature_list '}' ';'
                { $$ = class_($2,idtable.add_string("Object"),$4,
                              stringtable.add_string(curr_filename)); }
        | CLASS TYPEID INHERITS TYPEID '{' dummy_feature_list '}' ';'
                { $$ = class_($2,$4,$6,stringtable.add_string(curr_filename)); }
        ;

/* Feature list may be empty, but no empty features in list. */
dummy_feature_list:             /* empty */
                {  $$ = nil_Features(); }
        ;
```

测试结果：

```
GuoSongjian(Fri May 19 15:17:58):~/compiler_exp/cool/cool/assignments/PA3$
../../bin/reference-lexer good.cl | ./parser
#5
_program
  #2
  _class
    A
    Object
    "good.cl"
    (
    )
  #5
  _class
    BB__
    A
    "good.cl"
    (
    )
```

(2) 实现 feature 的相关文法

文法如下所示：

$$\begin{aligned} feature &::= \text{ID}(\ [\ formal\ [\![, formal]\!]^*\ ]\ ) : \text{TYPE}\ \{\ expr\ \} \\ &\quad |\quad \text{ID} : \text{TYPE}\ [\ \texttt{<-}\ expr\ ] \end{aligned}$$

yacc 代码：

```
feature_list
        :       { $$ = nil_Features(); }
        | feature_list feature
                { $$ = append_Features($1, single_Features($2)); }
        ;

feature : OBJECTID '(' formal_list ')' ':' TYPEID '{' expression '}' ';'
                { $$ = method($1, $3, $6, $8); }
        | OBJECTID ':' TYPEID assignment ';'
                { $$ = attr($1, $3, $4); }
        ;

assignment
        :       { $$ = no_expr(); }
        | ASSIGN expression
                { $$ = $2; }
;

formal_list
        :       { $$ = nil_Formals(); }
;

expression
        : INT_CONST
                { $$ = int_const($1); }
;
```

测试结果：

```
GuoSongjian(Fri May 19 16:59:22):~/compiler_exp/cool/cool/assignments/PA3$
../../bin/reference-lexer good.cl | ./parser
#7
_program
  #4
  _class
    A
    Object
    "good.cl"
    (
    #2
    _attr
      num
      INT
      #2
      _int
        3
      : _no_type
    #3
    _method
      main
      SELF_TYPE
      #3
      _int
        5
      : _no_type
    )
  #7
  _class
    BB__
    A
    "good.cl"
    (
    )
```

(3) 实现 formal 的相关文法

文法如下所示：

$$formal \ ::= \ \text{ID} : \text{TYPE}$$

yacc 代码：

```
formal_list
        :       { $$ = nil_Formals(); }
        | formal
                { $$ = single_Formals($1); }
        | formal_list ',' formal
                { $$ = append_Formals($1, single_Formals($3)); }
;

formal  : OBJECTID ':' TYPEID
                { $$ = formal($1, $3); }
;
```

测试结果：

```
GuoSongjian(Fri May 19 17:06:45):~/compiler_exp/cool/cool/assignments/PA3$
../../bin/reference-lexer good.cl | ./parser
#7
_program
  #4
  _class
    A
    Object
    "good.cl"
    (
    #2
    _attr
      num
      INT
      #2
      _int
        3
      : _no_type
    #3
    _method
      main
      #3
      _formal
        a
        INT
      #3
      _formal
        b
        STRING
      SELF_TYPE
      #3
      _int
        5
      : _no_type
    )
  #7
  _class
    BB__
    A
    "good.cl"
    (
    )
```

（4）实现代码块的相关文法

文法如下所示：

$$
\begin{aligned}
&\mid \{\ [\![expr;]\!]^{+}\ \} \\
&\mid (expr) \\
&\mid \text{ID} \\
&\mid \text{integer} \\
&\mid \text{string} \\
&\mid \textbf{true} \\
&\mid \textbf{false}
\end{aligned}
$$

yacc 代码：

```
expression_list1
        :       { $$ = nil_Expressions(); }
        | expression
                { $$ = single_Expressions($1); }
        | expression_list1 ',' expression
                { $$ = append_Expressions($1, single_Expressions($3)); }
;
expression_list2
        : expression ';'
                { $$ = single_Expressions($1); }
        | expression_list2 expression ';'
                { $$ = append_Expressions($1, single_Expressions($2)); }
        | error ';'
                { yyerrok; }
;
expression
        : '{' expression_list2 '}'
                { $$ = block($2); }
        | '(' expression ')'
                { $$ = $2; }
        | OBJECTID
                { $$ = object($1); }
        | INT_CONST
                { $$ = int_const($1); }
        | STR_CONST
                { $$ = string_const($1); }
        | BOOL_CONST
                { $$ = bool_const($1); }
;
```

测试结果：

```
GuoSongjian(Fri May 19 20:15:31):~/compiler_exp/cool/cool/assignments/PA3$
../../bin/reference-lexer good.cl | ./parser
#5
_program
  #5
  _class
    A
    Object
    "good.cl"
    (
    #4
    _method
      main
      #2
      _formal
        a
        INT
      #2
      _formal
        b
        STRING
      SELF_TYPE
```

```
    #3
    _block
      #3
      _int
        5
      : _no_type
      #3
      _string
        "hello world"
      : _no_type
      #3
      _bool
        1
      : _no_type
    : _no_type
)
```

（5）实现 let、case 语句的相关文法

文法如下所示：

$$
\begin{aligned}
&\mid \quad \text{let ID : TYPE [<- } expr \text{] } [\![\text{, ID : TYPE [<- } expr \text{]}]\!]^{*} \text{ in } expr \\
&\mid \quad \text{case } expr \text{ of } [\![\text{ID : TYPE => } expr\text{;}]\!]^{+} \text{esac}
\end{aligned}
$$

yacc 代码：

```
expression
        : '{' expression_list2 '}'
                { $$ = block($2); }
        | '(' expression ')'
                { $$ = $2; }
        | OBJECTID
                { $$ = object($1); }
        | INT_CONST
                { $$ = int_const($1); }
        | STR_CONST
                { $$ = string_const($1); }
        | BOOL_CONST
                { $$ = bool_const($1); }
        | LET let
                { $$ = $2; }
        | CASE expression OF case_list ESAC
                { $$ = typcase($2, $4); }
;
let     : OBJECTID ':' TYPEID assignment IN expression
                { $$ = let($1, $3, $4, $6); }
        | OBJECTID ':' TYPEID assignment ',' let
                { $$ = let($1, $3, $4, $6); }
        | error IN expression
                { yyerrok; }
        | error ',' let
                { yyerrok; }
;
case_list
        : case
                { $$ = single_Cases($1); }
        | case_list case
                { $$ = append_Cases($1, single_Cases($2)); }
;
case    : OBJECTID ':' TYPEID DARROW expression ';'
                { $$ = branch($1, $3, $5); }
;
```

测试结果：

```
GuoSongjian(Fri May 19 21:00:59):~/compiler_exp/cool/cool/assignments/PA3$
../../bin/reference-lexer good.cl | ./parser
#9
_program
  #9
  _class
    Foo
    Object
    "good.cl"
    (
    #8
    _method
      doh
      Int
      #8
      _block
        #3
        _let
          i
          Int
          #3
          _no_expr
          : _no_type
          #3
          _int
            3
          : _no_type
        : _no_type
```

```
        #7
        _typcase
          #4
          _object
            self
          : _no_type
          #5
          _branch
            n
            Razz
            #5
            _object
              bar
            : _no_type
          #6
          _branch
            n
            Bar
            #6
            _object
              n
            : _no_type
        : _no_type
      : _no_type
    )
GuoSongjian(Fri May 19 21:02:17):~/compiler_exp/cool/cool/assignments/PA3$
```

（6）实现其它表达式（运算、if、while 等）的相关文法

文法如下所示：

$$
\begin{aligned}
expr \ ::= \ & \text{ID} \ \texttt{<-} \ expr \\
| \ & expr[@\text{TYPE}].\text{ID}(\ [\ expr\ [\![, expr]\!]^*\ ]\ ) \\
| \ & \text{ID}(\ [\ expr\ [\![, expr]\!]^*\ ]\ ) \\
| \ & \textbf{if}\ expr\ \textbf{then}\ expr\ \textbf{else}\ expr\ \textbf{fi} \\
| \ & \textbf{while}\ expr\ \textbf{loop}\ expr\ \textbf{pool} \\
| \ & \{\ [\![expr;]\!]^+\} \\
| \ & \textbf{let}\ \text{ID} : \text{TYPE}\ [\ \texttt{<-}\ expr\ ]\ [\![, \text{ID} : \text{TYPE}\ [\ \texttt{<-}\ expr\ ]]\!]^*\ \textbf{in}\ expr \\
| \ & \textbf{case}\ expr\ \textbf{of}\ [\![\text{ID} : \text{TYPE} \texttt{=>} expr;]\!]^+\textbf{esac} \\
| \ & \textbf{new}\ \text{TYPE} \\
| \ & \textbf{isvoid}\ expr \\
| \ & expr + expr \\
| \ & expr - expr \\
| \ & expr * expr \\
| \ & expr / expr \\
| \ & \tilde{}\ expr \\
| \ & expr < expr \\
| \ & expr <= expr \\
| \ & expr = expr \\
| \ & \textbf{not}\ expr \\
| \ & (expr) \\
| \ & \text{ID} \\
| \ & \text{integer} \\
| \ & \text{string} \\
| \ & \textbf{true} \\
| \ & \textbf{false}
\end{aligned}
$$

yacc 代码：

```
expression
        : OBJECTID ASSIGN expression
                { $$ = assign($1, $3); }
        | expression '.' OBJECTID '(' expression_list1 ')'
                { $$ = dispatch($1, $3, $5); }
        | expression '@' TYPEID '.' OBJECTID '(' expression_list1 ')'
                { $$ = static_dispatch($1, $3, $5, $7); }
        | OBJECTID '(' expression_list1 ')'
                { $$ = dispatch(object(idtable.add_string("self")), $1, $3); }
        | IF expression THEN expression ELSE expression FI
                { $$ = cond($2, $4, $6); }
        | WHILE expression LOOP expression POOL
                { $$ = loop($2, $4); }
        | NEW TYPEID
                { $$ = new_($2); }
        | ISVOID expression
                { $$ = isvoid($2); }
```

```
        | expression '+' expression
                { $$ = ::plus($1, $3); }
        | expression '-' expression
                { $$ = sub($1, $3); }
        | expression '*' expression
                { $$ = mul($1, $3); }
        | expression '/' expression
                { $$ = divide($1, $3); }
        | '~' expression
                { $$ = neg($2); }
        | expression '<' expression
                { $$ = lt($1, $3); }
        | expression LE expression
                { $$ = leq($1, $3); }
        | expression '=' expression
                { $$ = eq($1, $3); }
        | NOT expression
                { $$ = comp($2); }
        | '{' expression_list2 '}'
                { $$ = block($2); }
        | '(' expression ')'
                { $$ = $2; }
        | OBJECTID
                { $$ = object($1); }
        | INT_CONST
                { $$ = int_const($1); }
        | STR_CONST
                { $$ = string_const($1); }
        | BOOL_CONST
                { $$ = bool_const($1); }
        | LET let
                { $$ = $2; }
        | CASE expression OF case_list ESAC
                { $$ = typcase($2, $4); }
;
```

测试结果：

```
GuoSongjian(Fri May 19 21:19:20):~/compiler_exp/cool/cool/assignments/PA3$
../../bin/reference-lexer good.cl | ./parser
#12
_program
  #12
  _class
    Main
    IO
    "good.cl"
    (
    #11
    _method
      main
      #2
      _formal
        i
        INT
      SELF_TYPE
      #10
      _block
        #6
        _loop
          #4
          _leq
```

```
            #4
            _object
              i
            : _no_type
            #4
            _int
              100
            : _no_type
          : _no_type
          #6
          _assign
            i
            #6
            _plus
              #5
              _object
                i
              : _no_type
```

```
              #5
              _int
                1
              : _no_type
            : _no_type
          : _no_type
        : _no_type
        #9
        _cond
          #7
          _lt
            #7
            _int
              0
            : _no_type
            #7
            _object
              i
            : _no_type
          : _no_type
          #8
          _dispatch
            #8
            _object
              self
            : _no_type
            out_int
            (
            #8
            _object
              i
            : _no_type
            )
          : _no_type
          #9
          _int
            5
          : _no_type
        : _no_type
      : _no_type
    )
GuoSongjian(Fri May 19 21:19:26):~/compiler_exp/cool/cool/assignments/PA3$
```

（7）与标准语法分析器的对比

```
GuoSongjian(Fri May 19 21:33:06):~/compiler_exp/cool/cool/assignments/PA3$ python3 comp.py ../../examples/arith.cl
Output is the same as reference-parser. Pass test.
GuoSongjian(Fri May 19 21:33:46):~/compiler_exp/cool/cool/assignments/PA3$ python3 comp.py ../../examples/life.cl
Output is the same as reference-parser. Pass test.
GuoSongjian(Fri May 19 21:34:06):~/compiler_exp/cool/cool/assignments/PA3$ python3 comp.py ../../examples/hairyscary.cl
Output is the same as reference-parser. Pass test.
GuoSongjian(Fri May 19 21:34:38):~/compiler_exp/cool/cool/assignments/PA3$ python3 comp.py ../../examples/complex.cl
Output is the same as reference-parser. Pass test.
```

comp.py 是用于比较自己实现的语法分析器与标准语法分析器输出的 python 脚本。经测试，examples 文件夹下的 cool 程序的语法分析输出结果与标准语法分析器相同。

三、源代码

（1）语法分析器-cool.y

```
/*
 *  cool.y
 *              Parser definition for the COOL language.
 *
 */
%{
#include <iostream>
#include "cool-tree.h"
#include "stringtab.h"
#include "utilities.h"

using namespace std;

extern char *curr_filename;

void yyerror(char *s);        /*  defined below; called for each
parse error */
extern int yylex();           /*  the entry point to the lexer  */

/************************************************************************
******/
/*               DONT CHANGE ANYTHING IN THIS SECTION
*/

Program ast_root;             /* the result of the parse  */
Classes parse_results;        /* for use in semantic analysis */
```

```
int omerrs = 0;               /* number of errors in lexing and
parsing */
%}

/* A union of all the types that can be the result of parsing
actions. */
%union {
   Boolean boolean;
   Symbol symbol;
   Program program;
   Class_ class_;
   Classes classes;
   Feature feature;
   Features features;
   Formal formal;
   Formals formals;
   Case case_;
   Cases cases;
   Expression expression;
   Expressions expressions;
   char *error_msg;
}

/*
   Declare the terminals; a few have types for associated lexemes.
   The token ERROR is never used in the parser; thus, it is a parse
   error when the lexer returns it.

   The integer following token declaration is the numeric constant
used
   to represent that token internally.  Typically, Bison generates
these
   on its own, but we give explicit numbers to prevent version
parity
   problems (bison 1.25 and earlier start at 258, later versions --
at
   257)
*/
%token CLASS 258 ELSE 259 FI 260 IF 261 IN 262
%token INHERITS 263 LET 264 LOOP 265 POOL 266 THEN 267 WHILE 268
%token CASE 269 ESAC 270 OF 271 DARROW 272 NEW 273 ISVOID 274
%token <symbol>  STR_CONST 275 INT_CONST 276
%token <boolean> BOOL_CONST 277
%token <symbol>  TYPEID 278 OBJECTID 279
```

```
%token ASSIGN 280 NOT 281 LE 282 ERROR 283

/*  DON'T CHANGE ANYTHING ABOVE THIS LINE, OR YOUR PARSER WON'T
WORK      */
/**************************************************************
********/

  /* Complete the nonterminal list below, giving a type for the
semantic
     value of each non terminal. (See section 3.6 in the bison
     documentation for details). */

/* Declare types for the grammar's non-terminals. */
%type <program> program
%type <classes> class_list
%type <class_> class
%type <formals> formal_list

/* You will want to change the following line. */
%type <features> feature_list
%type <feature>  feature
%type <formal> formal
%type <expression> assignment
%type <expression> expression
%type <expressions> expression_list1
%type <expressions> expression_list2
%type <expression> let
%type <cases> case_list
%type <case_> case

/* Precedence declarations go here. */
%right ASSIGN
%right NOT
%nonassoc '<' '=' LE
%left '+' '-'
%left '*' '/'
%left ISVOID
%left '~'
%left '@'
%left '.'

%%
/*
  Save the root of the abstract syntax tree in a global variable.
```

```
*/
program : class_list
            { ast_root = program($1); }
;

class_list
      : class           /* single class */
            { $$ = single_Classes($1);
              parse_results = $$; }
      | class_list class  /* several classes */
            { $$ = append_Classes($1,single_Classes($2));
              parse_results = $$; }
;

/* If no parent is specified, the class inherits from the Object
class. */
class   : CLASS TYPEID '{' feature_list '}' ';'
            { $$ = class_($2,idtable.add_string("Object"),$4,
                          stringtable.add_string(curr_filename)); }
      | CLASS TYPEID INHERITS TYPEID '{' feature_list '}' ';'
            { $$ =
class_($2,$4,$6,stringtable.add_string(curr_filename)); }
      | error ';'
            { yyerrok; }
;

/* Feature list may be empty, but no empty features in list. */
feature_list
      :     { $$ = nil_Features(); }
      | feature_list feature
            { $$ = append_Features($1, single_Features($2)); }
;

feature : OBJECTID '(' formal_list ')' ':' TYPEID '{' expression
'}' ';'
            { $$ = method($1, $3, $6, $8); }
      | OBJECTID ':' TYPEID assignment ';'
            { $$ = attr($1, $3, $4); }
;

assignment
      :     { $$ = no_expr(); }
      | ASSIGN expression
            { $$ = $2; }
```

```
;

formal_list
       :      { $$ = nil_Formals(); }
       | formal
              { $$ = single_Formals($1); }
       | formal_list ',' formal
              { $$ = append_Formals($1, single_Formals($3)); }
;

formal  : OBJECTID ':' TYPEID
              { $$ = formal($1, $3); }
;

expression_list1
       :      { $$ = nil_Expressions(); }
       | expression
              { $$ = single_Expressions($1); }
       | expression_list1 ',' expression
              { $$ = append_Expressions($1,
single_Expressions($3)); }
;

expression_list2
       : expression ';'
              { $$ = single_Expressions($1); }
       | expression_list2 expression ';'
              { $$ = append_Expressions($1,
single_Expressions($2)); }
       | error ';'
              { yyerrok; }
;

expression
       : OBJECTID ASSIGN expression
              { $$ = assign($1, $3); }
       | expression '.' OBJECTID '(' expression_list1 ')'
              { $$ = dispatch($1, $3, $5); }
       | expression '@' TYPEID '.' OBJECTID '(' expression_list1
')'
              { $$ = static_dispatch($1, $3, $5, $7); }
       | OBJECTID '(' expression_list1 ')'
              { $$ = dispatch(object(idtable.add_string("self")),
$1, $3); }
```

```
       | IF expression THEN expression ELSE expression FI
            { $$ = cond($2, $4, $6); }
       | WHILE expression LOOP expression POOL
            { $$ = loop($2, $4); }
       | NEW TYPEID
            { $$ = new_($2); }
       | ISVOID expression
            { $$ = isvoid($2); }
       | expression '+' expression
            { $$ = ::plus($1, $3); }
       | expression '-' expression
            { $$ = sub($1, $3); }
       | expression '*' expression
            { $$ = mul($1, $3); }
       | expression '/' expression
            { $$ = divide($1, $3); }
       | '~' expression
            { $$ = neg($2); }
       | expression '<' expression
            { $$ = lt($1, $3); }
       | expression LE expression
            { $$ = leq($1, $3); }
       | expression '=' expression
            { $$ = eq($1, $3); }
       | NOT expression
            { $$ = comp($2); }
       | '{' expression_list2 '}'
            { $$ = block($2); }
       | '(' expression ')'
            { $$ = $2; }
       | OBJECTID
            { $$ = object($1); }
       | INT_CONST
            { $$ = int_const($1); }
       | STR_CONST
            { $$ = string_const($1); }
       | BOOL_CONST
            { $$ = bool_const($1); }
       | LET let
            { $$ = $2; }
       | CASE expression OF case_list ESAC
            { $$ = typcase($2, $4); }
;
```

```
let    : OBJECTID ':' TYPEID assignment IN expression
              { $$ = let($1, $3, $4, $6); }
       | OBJECTID ':' TYPEID assignment ',' let
              { $$ = let($1, $3, $4, $6); }
       | error IN expression
              { yyerrok; }
       | error ',' let
              { yyerrok; }
;

case_list
       : case
              { $$ = single_Cases($1); }
       | case_list case
              { $$ = append_Cases($1, single_Cases($2)); }
;

case   : OBJECTID ':' TYPEID DARROW expression ';'
              { $$ = branch($1, $3, $5); }
;

/* end of grammar */
%%

/* This function is called automatically when Bison detects a parse
error. */
void yyerror(char *s)
{
   extern int curr_lineno;

   cerr << "\"" << curr_filename << "\", line " << curr_lineno <<
": " \
        << s << " at or near ";
   print_cool_token(yychar);
   cerr << endl;
   omerrs++;

   if(omerrs > 50) { fprintf(stdout, "More than 50 errors\n");
exit(1); }
}
```

(2) 与标准语法分析器的对比-comp.py

```
import os
import sys
```

```
args = sys.argv
command = f"../../bin/reference-lexer {args[1]} | ./parser >
test.output"
os.system(command)
command = f"../../bin/reference-lexer {args[1]}
| ../../bin/reference-parser > standard.output"
os.system(command)
flag = True
with open('test.output', 'r') as f1, open('standard.output', 'r')
as f2:
    for line1, line2 in zip(f1, f2):
        if line1 != line2:
            flag = False
            print(f"yourtest: {line1.strip()}")
            print(f"standard: {line2.strip()}")
if flag:
    print("Output is the same as reference-parser. Pass test.")
```