Студент: А.А. Довженко
Группа: М8О-207Б
Номер по списку: 6

«СИСТЕМЫ ПРОГРАММИРОВАНИЯ»
Курсовой проект 2018.
Часть 2.

Для языка МИКРОЛИСП на базе класса tSM сконструировать семантический анализатор, реализующий правила, записанные в файле SemanticRules.doc

Для каждого алгоритма анализа разработать сценарий тестирования, покрывающий все ветвления алгоритма. Имя тестового файла должно содержать имя продукции атрибутов, которой предназначен тест.

Анализатор протестировать с помощью приложения mlispsem, настроенного на грамматику mlisp18.

Распечатка файла semantics.cpp

```
/* $mlisp18 */
#include "semantics.h"
#include "semempty.cpp"

using namespace std;

void tSM::init() {
   globals.clear();
   locals.clear();
   params.clear();
   scope = 0; // глобальная область
//константы:
   globals["e"] =
//        properties
   tgName(VAR|DEFINED|BUILT);
```

```
   globals["pi"] =
   tgName(VAR|DEFINED|BUILT);
//
// элементарные процедуры:
   globals["remainder"] =
//         properties      arity
   tgName(PROC|DEFINED|BUILT, 2);
   globals["abs"] =
   tgName(PROC|DEFINED|BUILT, 1);
   globals["expt"] =
   tgName(PROC|DEFINED|BUILT, 2);
   globals["display"] =
   tgName(PROC|DEFINED|BUILT, 1);
   globals["newline"] =
   tgName(PROC|DEFINED|BUILT, 0);
   globals["quotient"] =
   tgName(PROC|DEFINED|BUILT, 2);
// ...
// только те процедуры, которые использованы
// в СВОИХ контрольных задачах
   return;
}

// +++ пункт 1
int tSM::p01() { //      S -> PROG
   bool error = false;
   for (tGlobal::iterator it = globals.begin(); it !=
globals.end(); ++it) {
      if ((it->second.test(USED)) && !(it-
>second.test(DEFINED)) && (it->second.test(PROC))) {//
процедура вызвана, но не определена
         ferror_message += "[!]Procedure application:"
                        " procedure '" + it->first +
                        "' used, but not defined!\n";
         error = true;
      }
      if ((it->second.test(USED)) && !(it-
>second.test(DEFINED)) && (it->second.test(VAR))) { //
переменная использована, но не определена
         ferror_message += "[!]Variable application:"
                        " variable '" + it->first +
                        "' used, but not defined!\n";
```

```
         error = true;
      }

      if (!(it->second.test(USED)) && (it-
>second.test(DEFINED)) && !(it->second.test(BUILT)) &&
(it->second.test(PROC))) {
         ferror_message += "[?]Procedure application:"
                  "procedure '" + it->first +
                  "' defined, but not used!\n";
      }
      if (!(it->second.test(USED)) && (it-
>second.test(DEFINED)) && !(it->second.test(BUILT)) &&
(it->second.test(VAR))) {
         ferror_message += "[?]Variable application:"
                  "variable '" + it->first +
                  "' defined, but not used!\n";
      }
// Просмотреть таблицу глобальных имен
// и в сообщении об ошибках указать имена
// ВСЕХ вызванных, но не определенных процедур,
// а также использованных, но не определенных
// глобальных переменных. Сообщения отметить [!].

// Кроме того, ПРЕДУПРЕДИТЬ обо ВСЕХ определенных,
// но не использованных процедурах и переменных,
// за исключением встроенных. Сообщения отметить [?].
//   it->first   имя
//   it->second  учетная запись
// ...
   } //for...
   if (error)
      return 1;
   return 0;
}

// идентификаторы переменных
int tSM::p13() { //      E -> $id
   string name = S1->name;
   if (scope > 1) { // локальная область
      if (locals.count(name)) { // имя локальное
         return 0;
      }
```

```
   }

   if (scope > 0) { // область параметров
      if (params.count(name)) { // имя параметра
         return 0;
      }
   }

   //найти имя в глобальной таблице
   tgName& ref = globals[name];
   if (ref.empty()) { // неизвестное имя
      ref.set(VAR);
      ref.set(USED); // новая запись
      return 0;
   }

   if (ref.test(PROC)) { // правило 13
      ferror_message = "[!]Procedure application:"
                  "name of procedure '" + name + "' cann't be
passed as parameter";
      return 1;
   }

   ref.set(USED);
   return 0;
}


int tSM::p21() { //      E -> CPROC
   string name = S1->name;
   int    count = S1->count;
   if (scope > 1) {// локальная область
      if (locals.count(name)) {// локальное имя
//p21-1.ss
         ferror_message="[!]Procedure application:"
         " local variable '" + name +
         "' shadows the procedure!";
         return 1;
      } // if locals ...
   } // if scope ...

   if (scope > 0) {// область параметров
```

```
      if (params.count(name)) {// имя параметра
//p21-2.ss
         ferror_message="[!]Procedure application:"
                     " parameter '" + name +
                     "' shadows the procedure!";
         return 1;
      }// if params...
   }// if scope...
   do {
//  найти имя в глобальной таблице
      tgName& ref = globals[name];
      if (ref.empty()) {//неизвестное имя
//       создать новую учетную запись
         ref = tgName(PROC|USED, count);
         break;
      }

// имя найдено
      if (!ref.test(PROC)) {//не процедура
//p21-3.ss
         ferror_message="[!]Procedure application:"
                     "  '" + name +
                     "' is not a procedure!";
         return 1;
      }

      if (ref.arity != count) {//число аргументов
//             не равно числу параметров
         std::ostringstream buf;
         buf << "[!]Procedure application: '" << name << "' "
//p21-4.ss
         << (ref.test(DEFINED) ? "expects " // процедура
//                              уже определена
//p21-5.ss

// процедура еще не определена, но уже вызывалась
ранее
               : "has been called already\n\t with "
            )
         << ref.arity <<" argument"
         << (ref.arity != 1 ? "s" : "")
         << ", given: " << count << " !";
```

```
         ferror_message = buf.str();
         return 1;
      }
// ошибок нет
      ref.set(USED);//имя использовано
   } while(false);

   return 0;
}

// вызов процедуры
int tSM::p50() { //  HCPROC -> ( $id
   S1->name = S2->name;
   S1->count = 0;
   return 0;
}

int tSM::p51() { //  HCPROC -> HCPROC E
   ++S1->count;
   return 0;
}

// имя предиката содержит ?, имена переменных и имена
параметров не могут его содержать, потому что это $id
int tSM::p58() { //   CPRED -> HCPRED )
   string name = S1->name;
   int count = S1->count;
   do {
   //  найти имя в глобальной таблице
      tgName& ref = globals[name];
      if (ref.empty()) { // неизвестное имя
         ref = tgName(PROC|USED, count); // создать новую
учетную запись
         return 0;
      }

      if (ref.arity != count){ //число аргументов не равно
числу параметров
         std::ostringstream buf;
         buf << "[!]Procedure application: '" << name << "' "
            << (ref.test(DEFINED) ? "expects " // процедура
   //                                 уже определена
```

```
   // процедура еще не определена, но уже вызывалась
ранее
                : "has been called already\n\t with "
                )
             << ref.arity << " argument"
             << (ref.arity != 1 ? "s" : "")
             << ", given: " << count << " !";
          ferror_message = buf.str();
          return 1;
       }
   // ошибок нет
      ref.set(USED);//имя использовано
   } while(false);

   return 0;
}


// вызов предиката
int tSM::p59() { //  HCPRED -> ( $idq
   S1->name = S2->name;
   S1->count = 0;
   return 0;
}

int tSM::p60() { //  HCPRED -> HCPRED E
   ++S1->count;
   return 0;
}

// присваивание
int tSM::p78() { //    HSET -> ( set! $id
   string name = S3->name;
   if (scope > 1) { // локальная область
      if (locals.count(name)) { // имя локальное
         return 0;
      }
   }

   if (scope > 0) { // область параметров
      if (params.count(name)) { // имя параметра
         return 0;
```

```
      }
   }

   // найти имя в глобальной таблице
   tgName& ref = globals[name];
   if (ref.empty()) { //неизвестное имя
      ref = tgName(VAR|USED); // создать новую учетную
запись
      return 0;
   }
   if (ref.test(PROC)) { //
      ferror_message = "[!]Procedure application:"
                  " procedure '" + name +
                  "' cann't be redefined";
      return 1;
   }
   if (ref.test(VAR) && ref.test(BUILT)) {
      ferror_message = "[!]Procedure application:"
                  " constant '" + name +
                  "' cann't be redefined";
      return 1;
   }

   ref.set(USED); // имя использовано
   return 0;
}

// определение предиката
int tSM::p89() { //    PRED -> HPRED BOOL )
   string name = S1->name;
   int count = S1->count;
   tgName& ref = globals[name]; // найти имя в
глобальной таблице
   if (ref.empty()) { //неизвестное имя
      ref = tgName(PROC|DEFINED, count); // создать
новую учетную запись
      params.clear();
      scope = 0;
      return 0;
   }

   if (ref.test(DEFINED)) { // p6
```

```
      ferror_message = "[!]Procedure defenition:"
                  " procedure '" + name+
                  "' was defined";
      return 1;
   }

   if (ref.arity != count) { //число аргументов не равно
числу параметров p4
      std::ostringstream buf;
      buf << "[!]Procedure application: '" << name << "' "
         << (ref.test(DEFINED) ? "expects " // процедура
уже определена
            : "has been called already\n\t with ") //
процедура еще не определена, но уже вызывалась ранее
         << ref.arity <<" argument"
         << (ref.arity != 1 ? "s" : "")
         << ", given: " << count << " !";
      ferror_message = buf.str();
      return 1;
   }

   ref.set(DEFINED);
   params.clear();
   scope = 0;
   return 0;
}

int tSM::p90() { //   HPRED -> PDPAR )
   scope = 1;
   return 0;
}

int tSM::p91() { //   PDPAR -> ( define ( $idq
   S1->name = S4->name;
   S1->count = 0;
   return 0;
}

// параметры
int tSM::p92() { //   PDPAR -> PDPAR $id
   if (params.count(S2->name)){ // p8
      ferror_message = "[!]Predicate definition:"
```

```
                    "in '" + S1->name +
                    "' duplicate parameter identifier '" + S2-
>name + "'!";
      return 1;
   }
   params.insert(S2->name);
   ++S1->count;
   return 0;
}

// определение глобальной переменной
int tSM::p95() { //    HVAR -> ( define $id
   string name = S3->name;
   tgName& ref = globals[name];
   if (ref.empty()) {//неизвестное имя
      //создать новую учетную запись
      ref = tgName(VAR|DEFINED);
      return 0;
   }

   if (ref.test(DEFINED)) { //p6
      ferror_message = "[!]Global variable init:"
         " redefinition '" + name + "'";
         return 1;
   }

   if (ref.test(PROC) && ref.test(USED)) { //p2
      ferror_message = "[!]Global variable init:"
                    " procedure '" + name +
                    "' was used";
      return 1;
   }

   ref.set(DEFINED);
   return 0;
}

// определение процедуры
int tSM::p96() { //    PROC -> HPROC LETLOC )
   string name = S1->name;
   int count = S1->count;
   tgName& ref = globals[name];
```

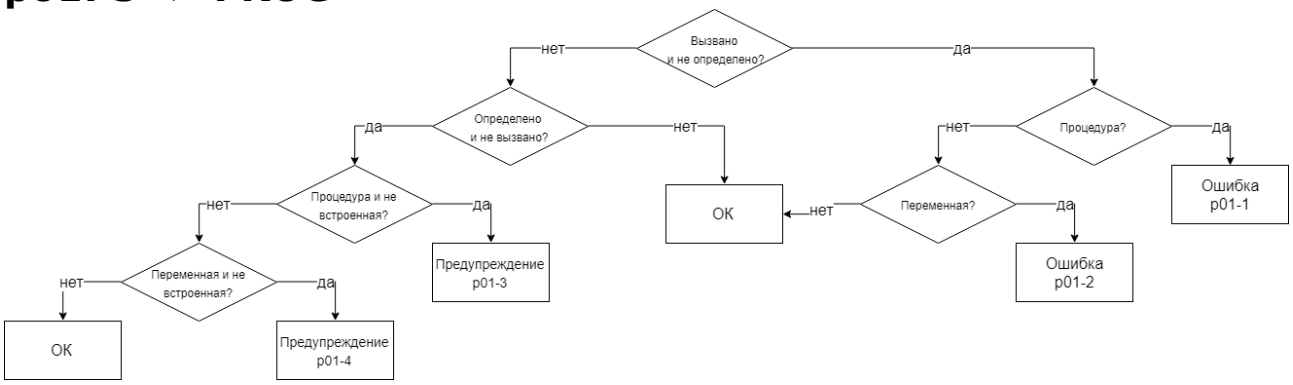
```
   if (ref.empty()){
      //не было упоминаний
      ref = tgName(PROC|DEFINED, count);
      params.clear();
      scope = 0;
      return 0;
   }
   if (ref.test(VAR)) { //p6
      ferror_message = "[!]Procedure init:"
                  " variable '" + name +
                  "' was initializated";
      return 1;
   }
   if(ref.test(DEFINED)) { //p6
      ferror_message = "[!]Procedure init:"
                  " procedure '" + name +
                  "' was initializated";
      return 1;
   }

   if (ref.arity != count) {
         std::ostringstream buf;
         buf << "[!]Procedure application: '" << name << "' "
         << "has been called already\n\t with "
         << ref.arity << " argument"
         << (ref.arity != 1 ? "s" : "")
         << ", given: " << count << " !";
      ferror_message = buf.str();
      return 1;
   }

   ref.set(DEFINED);
   params.clear();
   scope = 0;
   return 0;
}

// определение процедуры
int tSM::p97() { //    PROC -> HPROC E1 )
   string name = S1->name;
   int count = S1->count;
   tgName& ref = globals[name];
```

```
   if (ref.empty()) {
      //не было упоминаний
      ref = tgName(PROC|DEFINED, count);
      params.clear();
      scope = 0;
      return 0;
   }
   if (ref.test(VAR)) {
      ferror_message = "[!]Procedure init:"
                  " variable '" + name +
                  "' was initialized";
      return 1;
   }
   if (ref.test(DEFINED)) {
      ferror_message = "[!]Procedure init:"
                  " '"+ name +
                  "' was initialized";
      return 1;
   }
   if (ref.arity != count) {
      std::ostringstream buf;
         buf << "[!]Procedure application: '" << name << "' "
         << "has been called already\n\t with "
         << ref.arity << " argument"
         << (ref.arity != 1 ? "s" : "")
         << ", given: " << count << " !";
      ferror_message = buf.str();
      return 1;
   }

   ref.set(DEFINED);
   params.clear();
   scope = 0;
   return 0;
}

int tSM::p98() { //   HPROC -> PCPAR )
   scope = 1;
   return 0;
}

int tSM::p100() { //   PCPAR -> ( define ( $id
```

```
   S1->name = S4->name;
   S1->count = 0;
   return 0;
}

// параметры
int tSM::p101() { //   PCPAR -> PCPAR $id
   if (params.count(S2->name)) {
//p101-1.ss
     ferror_message = "[!]Procedure definition: in '"
                 + S1->name +
                 "' duplicate parameter identifier '"
                 + S2->name + "'!";
     return 1;
   }
   params.insert(S2->name);
   ++S1->count;
   return 0;
}

int tSM::p102() { //  LETLOC -> HLETLOC E1 )
   locals.clear();
   return 0;
}

int tSM::p103() { // HLETLOC -> LTVAR )
   scope = 2;
   return 0;
}

// определение локальной переменной
int tSM::p105() { //   LTVAR -> ( let ( CPROC
   S1->name = S4->name;
   S1->count = 1;
   locals.insert(S4->name);
   return 0;
}

// определение локальных переменных
int tSM::p106() { //   LTVAR -> LTVAR CPROC
   if (locals.count(S2->name)){
      ferror_message = "[!]Local variables definition: in '"
```

```
                    + S1->name +
                    "' duplicate variable initialization'"
                    + S2->name + "'!";
      return 1;
    }
    locals.insert(S2->name);
    ++S1->count;
    return 0;
}
//_____________________________
```

Блок-схемы алгоритмов анализа, аннотированные именами тестовых файлов и протоколы тестирования. p01: S -> PROG



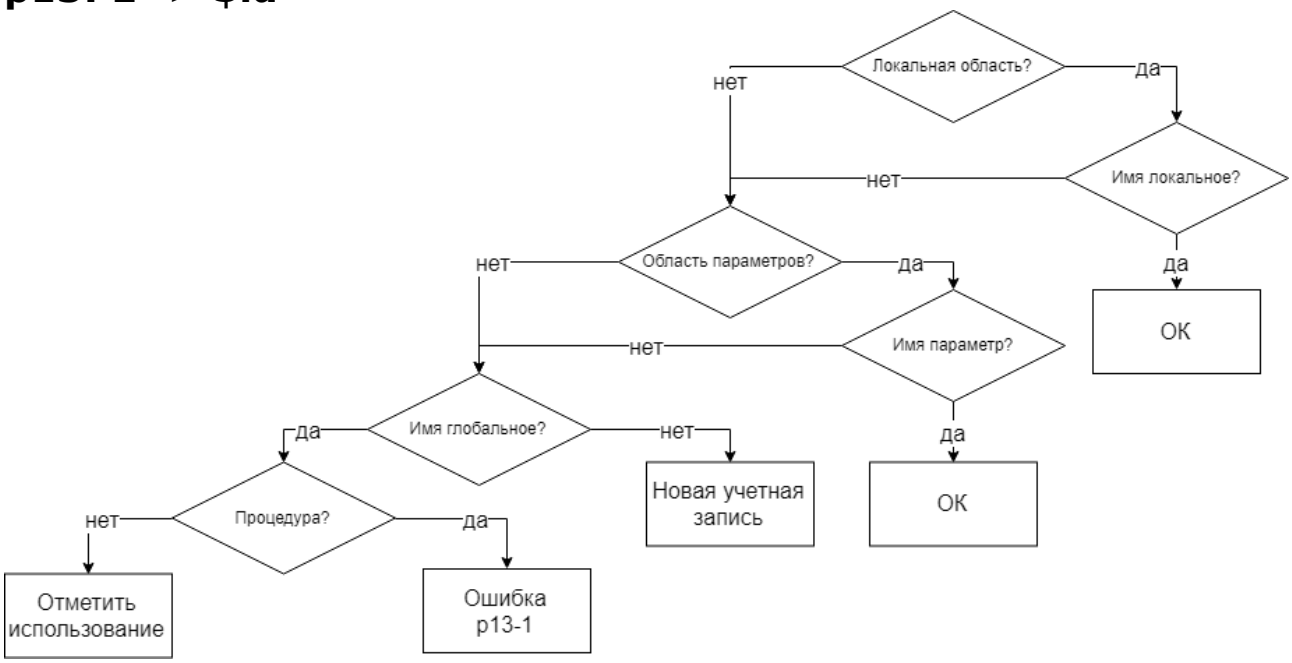
karma@DESKTOP-K0CDBM7: /mnt/c/Users/Karma/Desktop/sp18/curs2new

```
_____________________
Source>p01-1
Source:p01-1.ss
   1|(f)
   2|
_________________
[!]Procedure application: procedure 'f' used, but not defined!

   2|
     ^

Rejected!
_____________________
Source>p01-2
Source:p01-2.ss
   1|(abs x)
   2|
_________________
[!]Variable application: variable 'x' used, but not defined!

   2|
     ^

Rejected!
_____________________
Source>p01-3
Source:p01-3.ss
   1|(define (f) pi)
   2|
_________________
[?]Procedure application:procedure 'f' defined, but not used!

Accepted!
_____________________
```

karma@DESKTOP-K0CDBM7: /mnt/c/Users/Karma/Desktop/sp18/curs2new

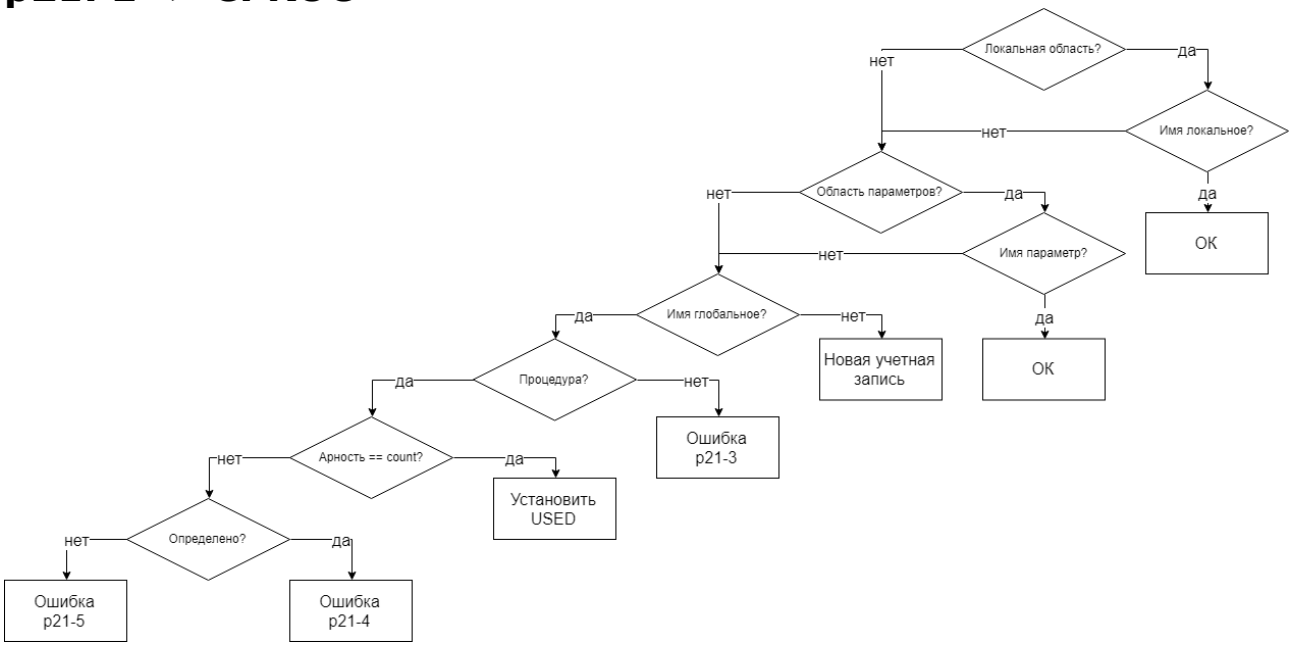
```
_____________________
Source>p01-4
Source:p01-4.ss
   1|(define x 1)
   2|
_________________
[?]Variable application:variable 'x' defined, but not used!

Accepted!
_____________________
Source>p01a
Source:p01a.ss
   1|;p01a
   2|(define y 2)
   3|(define (f x) (- y x))
   4|
   5|(f 1)
   6|
_________________

Accepted!
```

p13: E -> $id



```
karma@DESKTOP-K0CDBM7: /mnt/c/Users/Karma/Desktop/sp18/curs2new
_____________________
Source>p13-1
Source:p13-1.ss
   1|(define (f) 1)
   2|(display f)
   3|
_________________
[!]Procedure application:name of procedure 'f' cann't be passed as parameter
   2|(display f)
              ^

Rejected!
_____________________
Source>p13a
Source:p13a.ss
   1|;p13a1
   2|
   3|(define x 1)
   4|(define (f) x)
   5|(f)
   6|
_________________

Accepted!
```

p21: E -> CPROC



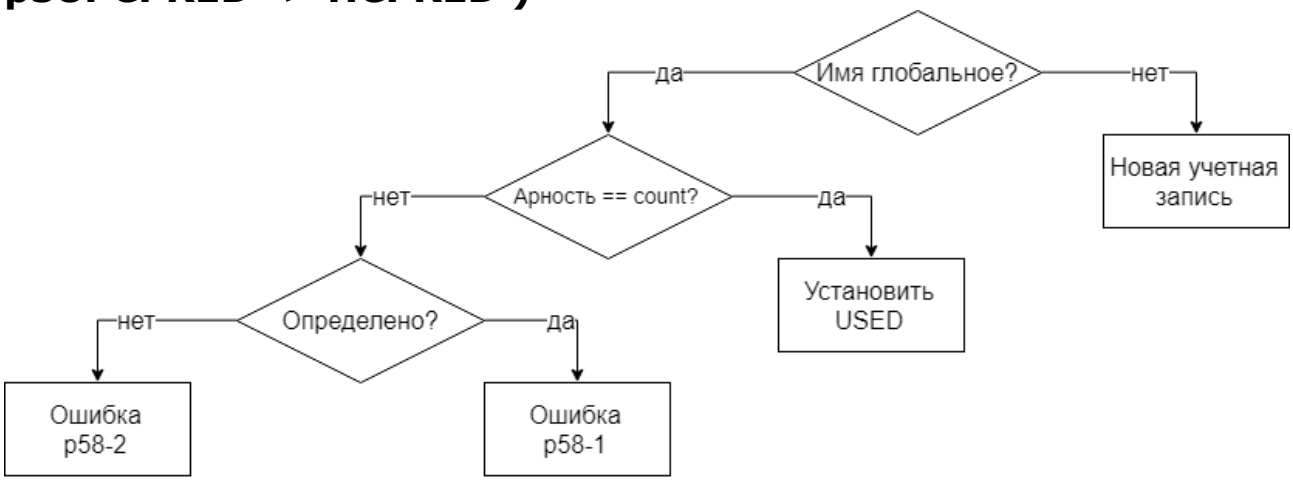
karma@DESKTOP-K0CDBM7: /mnt/c/Users/Karma/Desktop/sp18/curs2new

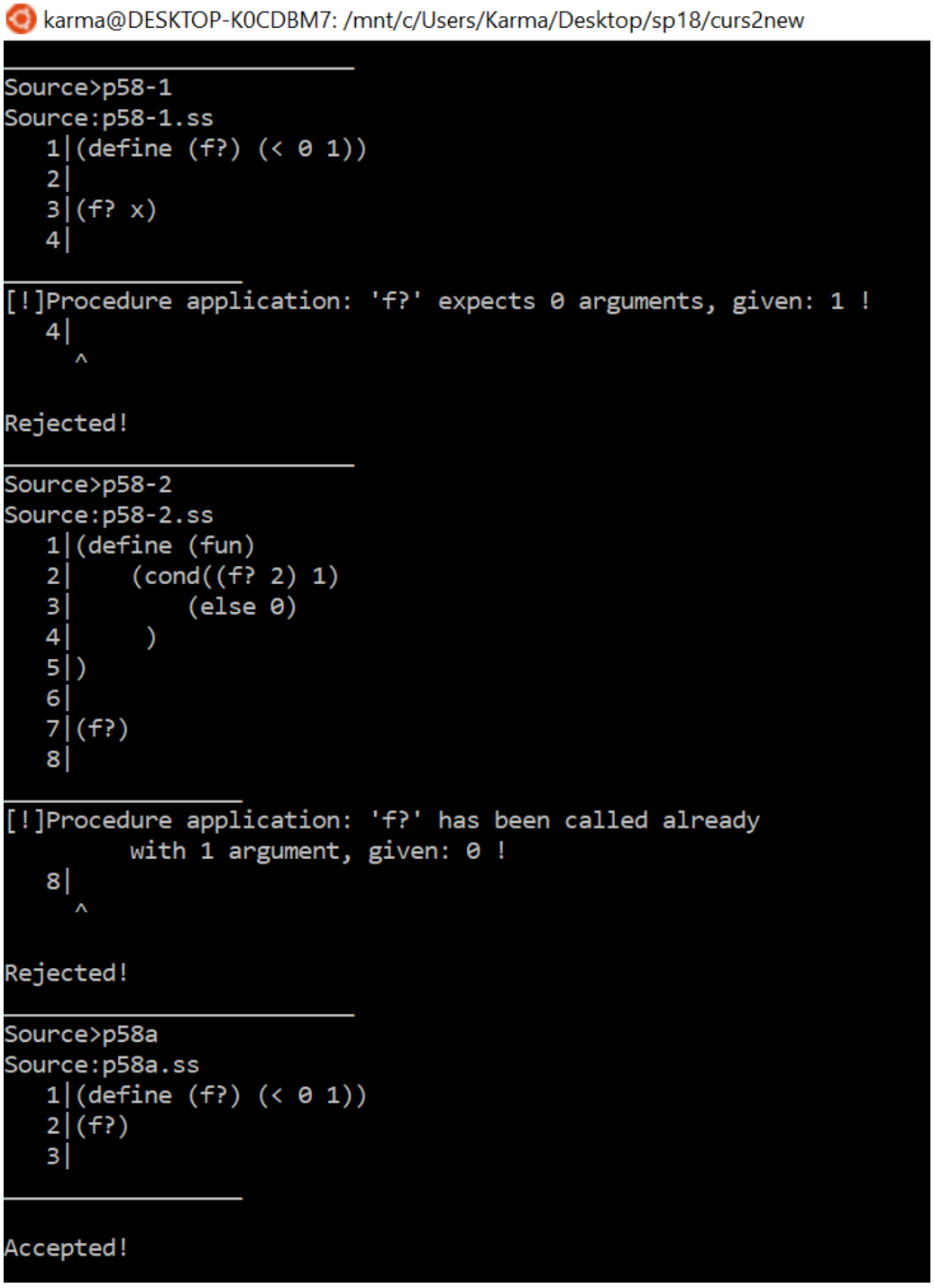
```
_____________________________
Source>p21-1
Source:p21-1.ss
   1|; p21-1
   2|(define(f x)(let((abs 1))(abs x)))
   3|
_________________
[!]Procedure application: local variable 'abs' shadows the procedure!
   2|(define(f x)(let((abs 1))(abs x)))
                                  ^

Rejected!
_____________________________
Source>p21-2
Source:p21-2.ss
   1|; p21-2
   2|(define(f x abs)(abs x))
   3|
_________________
[!]Procedure application: parameter 'abs' shadows the procedure!
   2|(define(f x abs)(abs x))
                        ^

Rejected!
_____________________________
Source>p21-3
Source:p21-3.ss
   1|; p21-3
   2|(e 0) pi
   3|
_________________
[!]Procedure application:  'e' is not a procedure!
   2|(e 0) pi
         ^

Rejected!
_____________________________
```

karma@DESKTOP-K0CDBM7: /mnt/c/Users/Karma/Desktop/sp18/curs2new

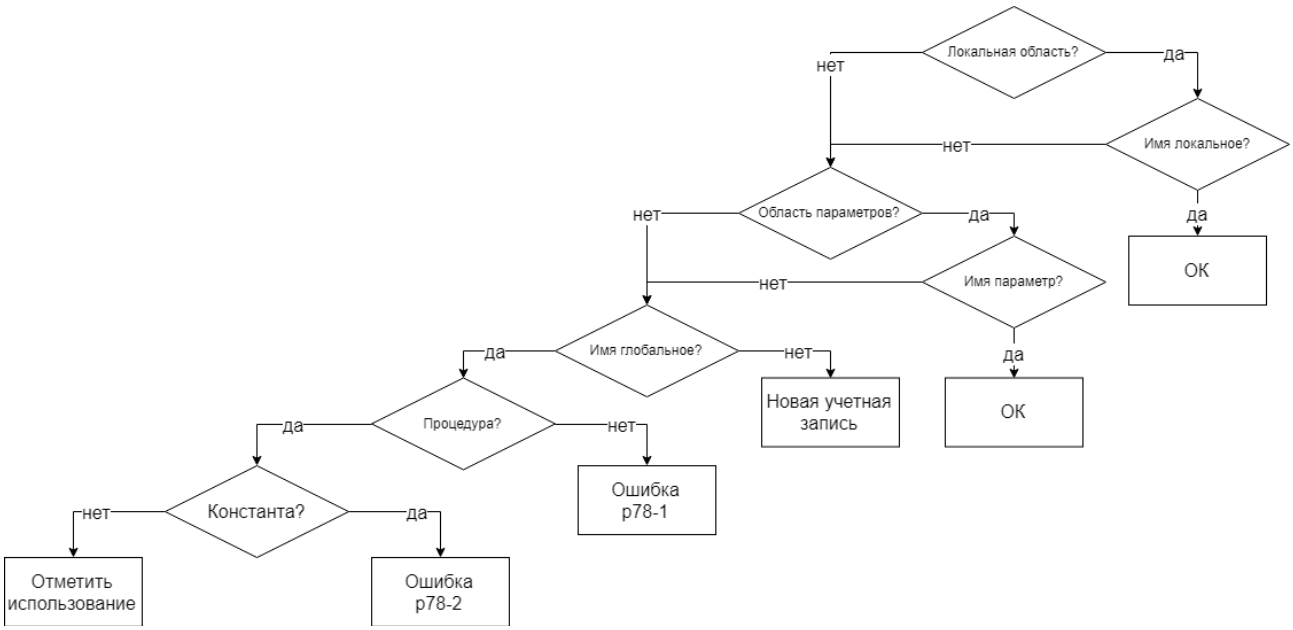
```
_____________________
Source>p21-4
Source:p21-4.ss
   1|; p21-4
   2|(abs 1 2) 0
   3|
_________________
[!]Procedure application: 'abs' expects 1 argument, given: 2 !
   2|(abs 1 2) 0
              ^

Rejected!
_____________________
Source>p21-5
Source:p21-5.ss
   1|; p21-5
   2|(define(g x) (f x (f x)))
   3|(define(f x) (* x x))
   4|
_________________
[!]Procedure application: 'f' has been called already
         with 1 argument, given: 2 !
   2|(define(g x) (f x (f x)))
                           ^

Rejected!
_____________________
Source>p21a
Source:p21a.ss
   1|; p21a
   2|(abs pi)
   3|
_________________

Accepted!
```

p58: CPRED -> HCPRED)





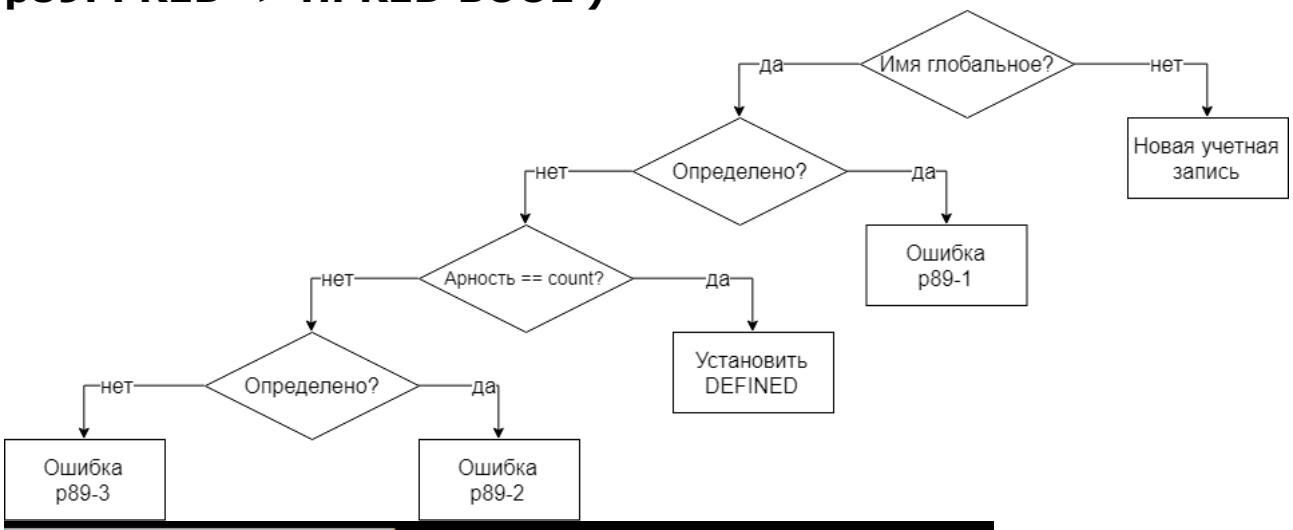
p78: HSET -> (set! $id



Выбрать karma@DESKTOP-K0CDBM7: /mnt/c/Users/Karma/Desktop/sp18/curs2new

```
_____________________
Source>p78-1
Source:p78-1.ss
   1|(define (f) 1)
   2|(set! f 1)
   3|
_____________________
[!]Procedure application: procedure 'f' cann't be redefined
   2|(set! f 1)
             ^

Rejected!
_____________________
Source>p78-2
Source:p78-2.ss
   1|(set! e 4)
   2|
_____________________
[!]Procedure application: constant 'e' cann't be redefined
   1|(set! e 4)
             ^

Rejected!
_____________________
Source>p78a
Source:p78a.ss
   1|(define f 1)
   2|(set! f 1)
   3|
_____________________

Accepted!
```

p89: PRED -> HPRED BOOL)




```
Source>p89-1
Source:p89-1.ss
   1|(define (f?) (< 0 1))
   2|(define (f? a b) (< a b))
   3|
_________________
[!]Procedure defenition: procedure 'f?' was defined
   3|
     ^

Rejected!
_____________________________
Source>p89-2
Source:p89-2.ss
   1|(define (f?) (< 0 1))
   2|(define (fun)
   3|    (cond((f? 2) 1)
   4|        (else 0)
   5|    )
   6|)
   7|
_________________
[!]Procedure application: 'f?' expects 0 arguments, given: 1 !
   3|    (cond((f? 2) 1)
                  ^

Rejected!
```

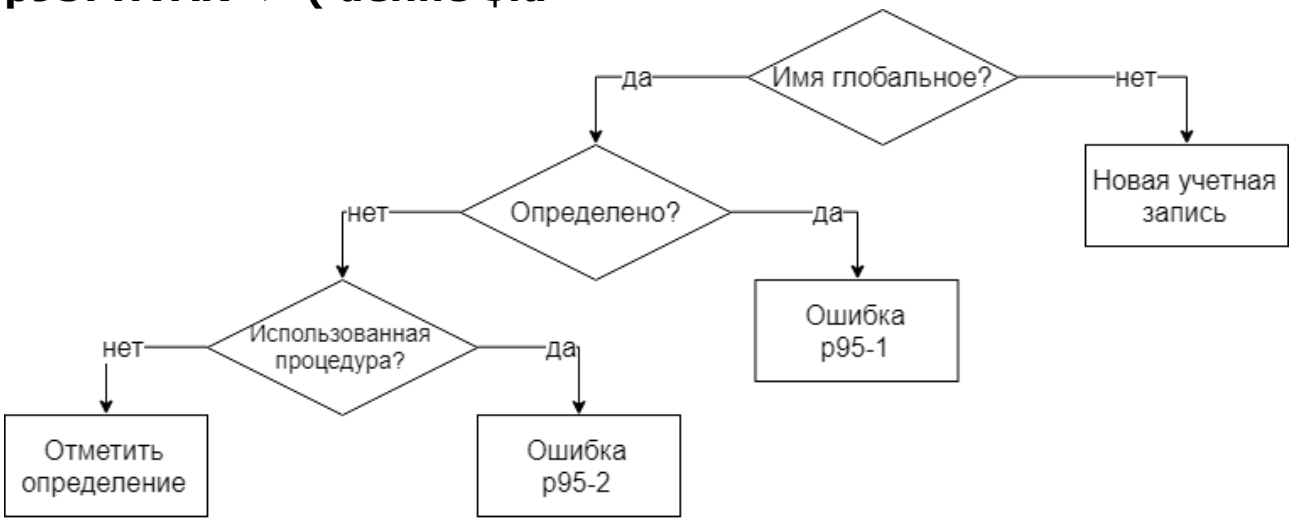
```
Выбрать karma@DESKTOP-K0CDBM7: /mnt/c/Users/Karma/Desktop/sp
Rejected!
_____________________________
Source>p89-3
Source:p89-3.ss
   1|(define (fun)
   2|    (cond((f? 2) 1)
   3|        (else 0)
   4|    )
   5|)
   6|
   7|(define (f?) (< 0 1))
   8|
_________________
[!]Procedure application: 'f?' has been called already
        with 1 argument, given: 0 !
   8|
     ^

Rejected!
_____________________________
Source>p89a
Source:p89a.ss
   1|(define (f?) (< 0 1))
   2|(define (fun)
   3|    (cond((f?) 1)
   4|        (else 0)
   5|    )
   6|)
   7|(fun)
   8|
_________________

Accepted!
```

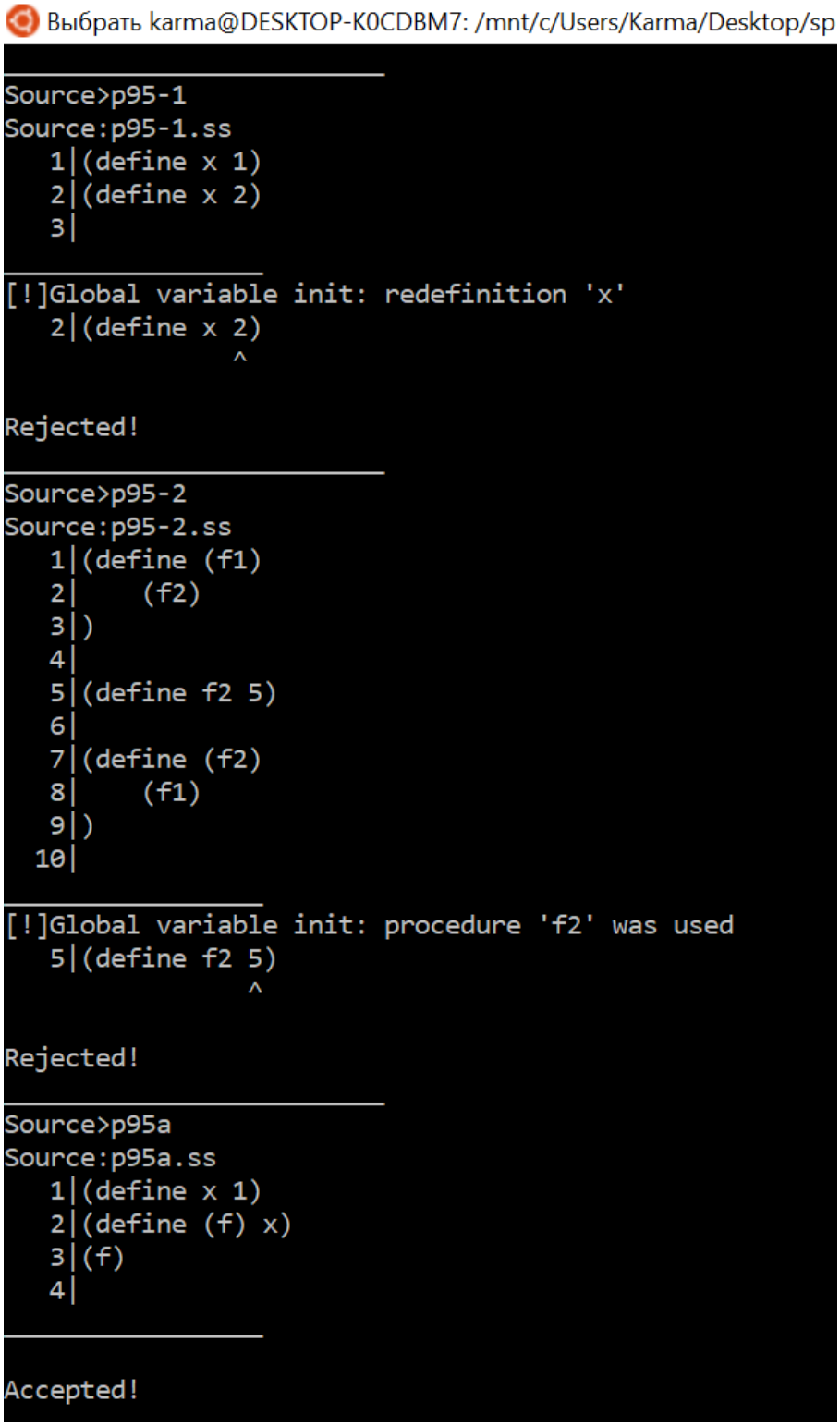
p92: PDPAR -> PDPAR $id

Выбрать karma@DESKTOP-K0CDBM7: /mnt/c/Users/Karma/Desktop/sp18/curs2new

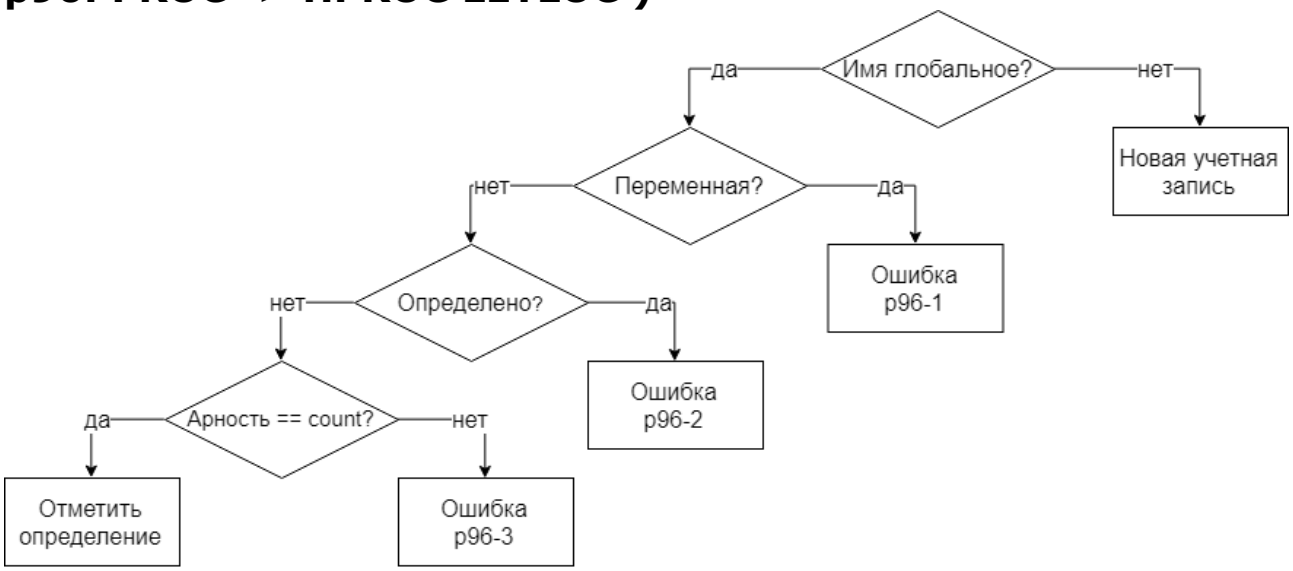
```
_____________________
Source>p92
Source:p92.ss
   1|(define (f? x x) (< x x))
   2|
_____________________
[!]Predicate definition:in 'f?' duplicate parameter identifier 'x'!
   1|(define (f? x x) (< x x))
                    ^

Rejected!

_____________________
Source>p92a
Source:p92a.ss
   1|(define (f? x y) (< x y))
   2|(f? 1 2)
   3|
_____________________

Accepted!
```

p95: HVAR -> (define $id




Выбрать karma@DESKTOP-K0CDBM7: /mnt/c/Users/Karma/Desktop/sp
Source>p95-1
Source:p95-1.ss
1|(define x 1)
2|(define x 2)
3|
[!]Global variable init: redefinition 'x'
2|(define x 2)
^
Rejected!
Source>p95-2
Source:p95-2.ss
1|(define (f1)
2| (f2)
3|)
4|
5|(define f2 5)
6|
7|(define (f2)
8| (f1)
9|)
10|
[!]Global variable init: procedure 'f2' was used
5|(define f2 5)
^
Rejected!
Source>p95a
Source:p95a.ss
1|(define x 1)
2|(define (f) x)
3|(f)
4|
Accepted!

p96: PROC -> HPROC LETLOC)

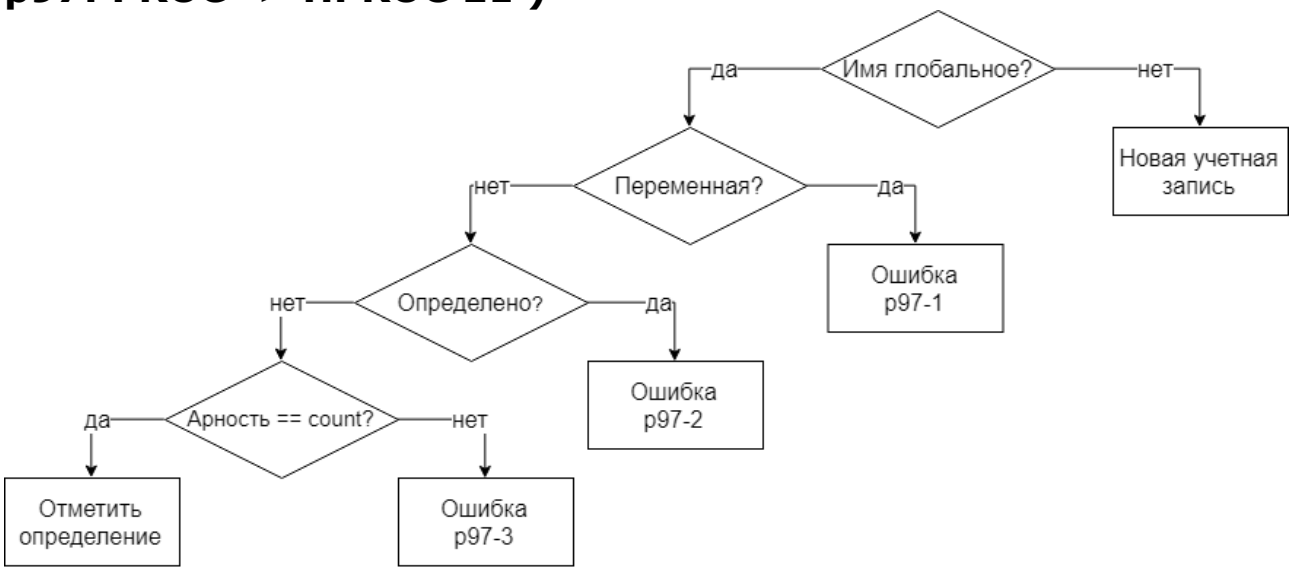


Выбрать karma@DESKTOP-K0CDBM7: /mnt/c/Users/Karma/Desktop

```
_____________________
Source>p96-1
Source:p96-1.ss
   1|(define f 0)
   2|
   3|(define (f)
   4| (let((temp 1))
   5|      temp
   6| )
   7|)
   8|
_________________
[!]Procedure init: variable 'f' was initialized
   8|
     ^

Rejected!
_____________________
Source>p96-2
Source:p96-2.ss
   1|(define (f) 0)
   2|
   3|(define (f)
   4| (let((temp 1))
   5|      temp
   6| )
   7|)
   8|
_________________
[!]Procedure init: procedure 'f' was initialized
   8|
     ^

Rejected!
```

```
Выбрать karma@DESKTOP-K0CDBM7: /mnt/c/Users/Karma/Desktop/sp1
_____________________
Source>p96-3
Source:p96-3.ss
   1|(define (fun) (f))
   2|
   3|(define (f x)
   4| (let((temp 1))
   5|      temp
   6| )
   7|)
   8|
_________________
[!]Procedure application: 'f' has been called already
        with 0 arguments, given: 1 !
   8|
     ^

Rejected!
_____________________
Source>p96a
Source:p96a.ss
   1|(define (f)
   2| (let((temp 1))
   3|    temp
   4| )
   5|)
   6|
   7|(f)
   8|
_________________

Accepted!
```

p97: PROC -> HPROC E1)



Выбрать karma@DESKTOP-K0CDBM7: /mnt/c/Users/Karma/Desktop/sp

```
_____________________________
Source>p97-1
Source:p97-1.ss
   1|(define f 0)
   2|(define (f) 1)
   3|
_________________
[!]Procedure init: variable 'f' was initialized
   3|
     ^

Rejected!
_____________________________
Source>p97-2
Source:p97-2.ss
   1|(define (f) 0)
   2|(define (f) 1)
   3|
_________________
[!]Procedure init: 'f' was initialized
   3|
     ^

Rejected!
_____________________________
Source>p97-3
Source:p97-3.ss
   1|(define (fun) (f))
   2|(define (f x) 1)
   3|
_________________
[!]Procedure application: 'f' has been called already
        with 0 arguments, given: 1 !
   3|
     ^

Rejected!
```

Выбрать karma@DESKTOP-K0CDBM7:

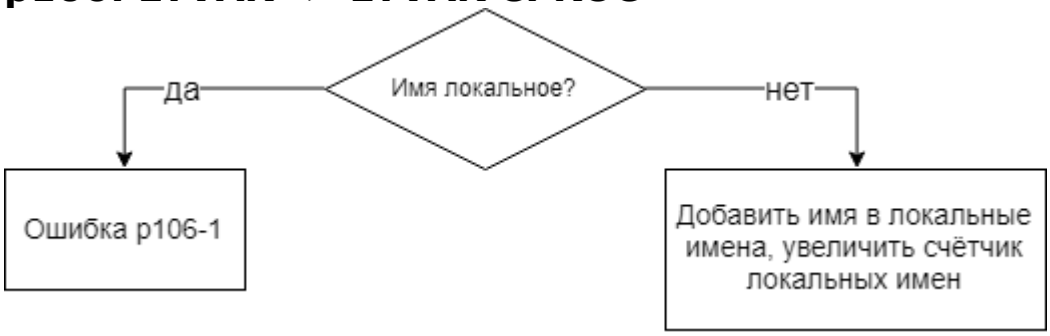
```
_____________________________
Source>p97a
Source:p97a.ss
   1|(define (f x y)
   2|    (+ x y)
   3|)
   4|
   5|(f 1 2)
   6|
_____________________

Accepted!
```

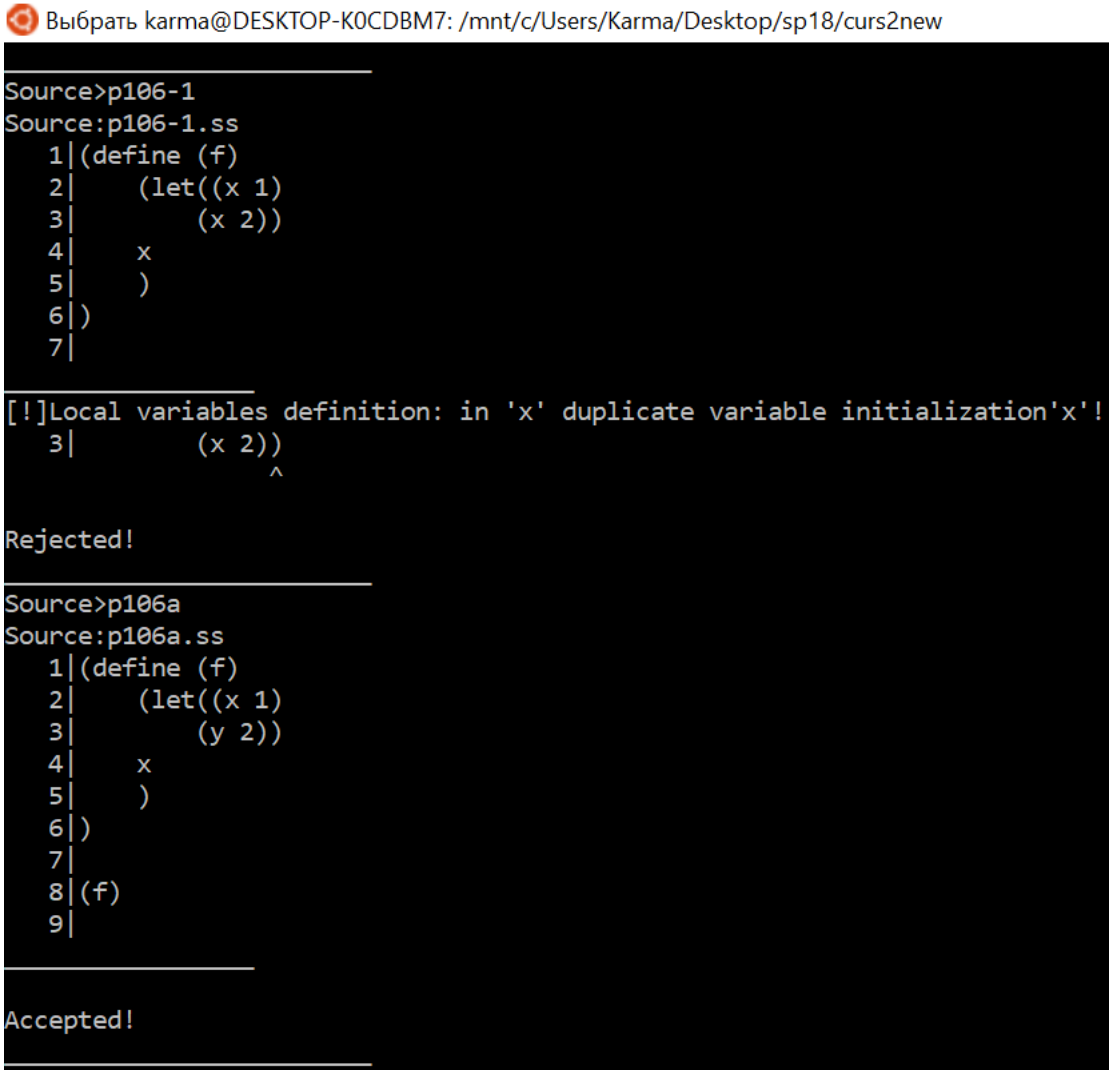
p101: PCPAR -> PCPAR $id

Выбрать karma@DESKTOP-K0CDBM7: /mnt/c/Users/Karma/Desktop/sp18/curs2new

```
_____________________________
Source>p101-1
Source:p101-1.ss
   1|; p101-1
   2|(define(f x x) x )
   3|
_________________
[!]Procedure definition: in 'f' duplicate parameter identifier 'x'!
   2|(define(f x x) x )
                  ^

Rejected!
_____________________________
Source>p101a
Source:p101a.ss
   1|; p101a
   2|(define(f x y) y)
   3|
_________________
[?]Procedure application:procedure 'f' defined, but not used!

Accepted!
```

p106: LTVAR -> LTVAR CPROC





Тесты на файлах even-odd.ss, half-interval.ss и coin.ss.

karma@DESKTOP-K0CDBM7: /mnt/c/Users/Karma/Desktop/sp18/curs2new

```
Source>even-odd
Source:even-odd.ss
  1|(define(even-bits n)
  2|  (cond((= n 0)1)
  3|        ((=(remainder n 2)0)
  4|           (even-bits (quotient n 2)))
  5|        (else(odd-bits(quotient n 2)))
  6|        ))
  7|(define(odd-bits n)
  8|  (cond((= n 0)0)
  9|        ((=(remainder n 2)0)
 10|           (odd-bits (quotient n 2)))
 11|        (#t(even-bits(quotient n 2)))
 12|        ))
 13|(define(display-bin n)
 14|  (display(remainder n 2))
 15|  (if(= n 0)0 (display-bin (quotient n 2)))
 16|        )
 17|(define(report-results n)
 18|  (display "Happy birthday to you!\n\t")
 19|  (display n)(display " (decimal)\n")
 20|  (display "\teven?\t")(display (if(=(even-bits n) 1) "yes" "no"))
 21|  (newline)
 22|  (display "\todd?\t")(display (if(=(odd-bits n) 1) "yes" "no"))
 23|  (newline)
 24|  (set! n(display-bin n))(display "(reversed binary)\n")
 25|  0
 26|        )
 27|;***** Date of YOUR birthday *******
 28|(define dd 2)
 29|(define mm 12)
 30|(define yyyy 1997)
 31|;*********************************
 32|(report-results (+ (* dd 1000000)
 33|                   (* mm 10000)
 34|                   yyyy))
 35|
_____________

Accepted!
```

karma@DESKTOP-K0CDBM7: /mnt/c/Users/Karma/Desktop/sp18/curs2new

```
Gramma:mlisp18.txt
Source>half-interval
Source:half-interval.ss
  1|;  half-interval.ss 2018
  2|(define (half-interval-metod a b)
  3|  (let((a-value (fun a))
  4|       (b-value (fun b))
  5|      )
  6|     (cond((and(< a-value 0)(> b-value 0))
  7|                (try a b))
  8|         ((and(> a-value 0)(< b-value 0))
  9|                (try b a))
 10|         (else(+ b 1))
 11|     )
 12|  )
 13|)
 14|(define(try neg-point pos-point)
 15| (let(
 16|       (midpoint (average neg-point pos-point))
 17|       (test-value 0)
 18|     )
 19|     (display "+")
 20|     (cond((close-enough? neg-point pos-point) midpoint)
 21|        (#t (set! test-value (fun midpoint))
 22|            (cond((> test-value 0)(try neg-point midpoint))
 23|                 ((< test-value 0)(try midpoint pos-point))
 24|                 (else midpoint))
 25|         )
 26|     )
 27| )
 28|)
 29|(define (close-enough? x y)
 30|  (<(abs (- x y))tolerance))
 31|(define (average x y)(/(+ x y)2.0))
 32|(define (root a b)
 33| (display"interval=\t[")
 34| (display a)
 35| (display" , ")
 36| (display b)
 37| (display"]\n")
 38| (let((temp (half-interval-metod a b)))
```

karma@DESKTOP-K0CDBM7: /mnt/c/Users/Karma/Desktop/sp18/curs2new

```
 37| (display"]\n")
 38| (let((temp (half-interval-metod a b)))
 39|      (newline)
 40|      (display"discrepancy=\t")
 41|      (display(fun temp))(newline)
 42|      (display"root=\t\t")
 43|      (display(if(=(- temp b 1)0)"[bad]" "[good]"))
 44|      temp
 45| )
 46|)
 47|(define tolerance 0.00001)
 48|(define(fun z)
 49|  (set! z (- z (/ 106 107)(/ e)))
 50|  (+ (* 0.25 (expt z 3))
 51|   (- z 1.2502)
 52|  )
 53|)
 54|
 55|
 56|" AAD variant 6"
 57|(root 2 3)
 58|
```

Accepted!

karma@DESKTOP-K0CDBM7: /mnt/c/Users/Karma/Desktop/sp18/curs2new

```
(define dd 2)
(define mm 12)
(define LAGEST-COIN 10)

(define (cc amount largest-coin)
    (cond((or (= amount 0)(= largest-coin 1))
        1)
        ((not (and (> amount 0) (> largest-coin 0)))
        0)
        (else (+ (cc amount (next-coin largest-coin)) (cc (- amount largest-coin) largest-coin)
        ))
    )
)

(define (count-change amount)
    (cc amount LAGEST-COIN)
)

(define (next-coin coin)
    (cond((= coin 10) 5)
        ((= coin 5) 3)
        ((= coin 3) 2)
        ((= coin 2) 1)
        (else 0)
    )
)

(define (GR-AMOUNT) (+ (* 100 mm) dd))

(display " AAD variant 6")(newline)
(display " 1-2-3-5-10")(newline)
(display "count__change for 100 \t= ")
(display (count-change 100))(newline)
(display "count__change for ")
(display (GR-AMOUNT))
(display " \t= ")
(display(count-change (GR-AMOUNT)))(newline)

```

Accepted!

Выводы.

В ходе выполнения курсового проекта был разработан семантический анализатор МИКРОЛИСПа, алгоритмы анализа которого покрывают заданные правила семантики. Для каждого такого алгоритма составлена блок-схема, на которой наглядно можно увидеть все возможные ветви, а также соответствующие им тестовые примеры. С помощью тестов можно убедиться, что анализатор работает корректно.

Самым сложным в выполнении курсового проекта было сложить все предложенные правила в единую картину. Хоть и некоторые части анализатора почти полностью копируют друг друга, подчас было непросто отследить, что уже реализованные алгоритмы не пересекаются с другими или что случайно не появились лишние ветви, которые анализатор никогда не исполнит. Но чем больше времени я писала этот анализатор, тем меньше ошибок обнаруживала. Отдельной сложностью было то, что в правилах не делается различий между процедурами и предикатами. Проверка многих правил выполняется не одним алгоритмом, а несколькими, это тоже нужно было внимательно отслеживать. Это была очень кропотливая работа, которая потребовала от меня полной концентрации и отдачи.

На всех моих тестовых примерах и контрольных задачах анализатор успешно работает. Не исключаю, что я могла упустить какие-то семантические конструкции, которые запрещены правилами семантики. Вторая часть курсового проекта выполнена в полном объеме.