Студент: Дубинин А.О.
Группа: М8О-206Б
Номер по списку: 7

«СИСТЕМЫ ПРОГРАММИРОВАНИЯ»
Курсовой проект 2019.
Часть 2.

Для языка МИКРОЛИСП на базе класса tSM сконструировать семантический анализатор, реализующий правила, записанные в файле SemanticRules19.rtf

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 mlisp19.

В отчете представить:
- текст задания;
- распечатку файла semantics.cpp;
- блок-схемы алгоритмов анализа, аннотированные именами тестовых файлов;
- протоколы тестирования;
- выводы.

Распечатка файла semantics.cpp:

```
/* $mlisp19 */
#include "semantics.h"
#include "semempty.cpp"

using namespace std;

void tSM::init() {
   globals.clear();
   locals.clear();
   params.clear();
   scope = 0; // вне процедуры
```

```
//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["eq?"] =
          tgName(PROC | DEFINED | BUILT, 2);
// ...
// только те процедуры, которые использованы
// в СВОИХ контрольных задачах
   return;
}

      int tSM::p01() { //      S -> PROG
   bool error = false;
   ferror_message = "";
   string name;
   tgName elem;
   for (tGlobal::iterator it = globals.begin();
        it != globals.end();
        ++it) {
       // 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
       // ...
```

```
      name = it->first;
      elem = it->second;
      if (elem.test(USED) && !elem.test(DEFINED)) {
         error = true;
         if (elem.test(PROC))
            ferror_message += "[!]Procedure application:"
                        "  '" + name +
                        "' was used and not defined\n";
         else
            ferror_message += "[!]Variable application:"
                        "  '" + name +
                        "' was used and not defined\n";
      }
      if (!elem.test(USED) && elem.test(DEFINED) && !
elem.test(BUILT)) {
         if (elem.test(PROC))
            ferror_message += "[?]Procedure application:"
                        "  '" + name +
                        "' was defined and not used\n";
         else
            ferror_message += "[?]Variable application:"
                        "  '" + name +
                        "' was defined and not used\n";
      }

   }//for...
   if (error) return 1;
   return 0;
}

int tSM::p11() { //      E -> $id
   string name = S1->name;
   if (scope == 2) {// локальная область (let ...)
      if (locals.count(name)) {// локальное имя
         //okay
         return 0;
      }

   }
   if (scope >= 1) {// область параметров
      if (params.count(name)) {// имя параметра
         //okay
```

```
            return 0;
         }
      }

      //найти имя в глобальной таблице
      tgName &ref = globals[name];
      //имя найдено
      if (ref.test(PROC)) {//процедура
         //процедуру нельзя использовать
         //только VAR или константу или параметр

         ferror_message = "[!]Procedure application:"
                     "  '" + name +
                     "' is a procedure and can not be used as a
variable";
         return 1;
      }
      if (ref.empty()) {
         //ранее имя не встречалось
         //отметить использование
         ref.set(USED);
         ref.set(VAR);
         //процедуру нельзя использовать так
         //только VAR
         return 0;
      }
      if (!ref.test(USED))
         //встречалось и не использовалось
         ref.set(USED);

      return 0;
}

int tSM::p45() { //   CPROC -> HCPROC )
      string name = S1->name;
      int count = S1->count;
      if (scope > 1) {// внури тела let
         if (locals.count(name)) {// локальное имя
            //p45-1.ss
            ferror_message = "[!]Procedure application:"
                        " local variable '" + name +
                        "' shadows the procedure!";
```

```
      return 1;
    } // if locals ...
  } // if scope ...
  if (scope > 0) {// внутри процедуры
    if (params.count(name)) {// имя параметра
      //p45-2.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
      //p45-3.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 << "' "
        //p45-4.ss
        << (ref.test(DEFINED) ? "expects " // процедура
          //                              уже определена
          //p45-5.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;
}

int tSM::p46() { //  HCPROC -> ( $id
   //валидация процедуры $id проводиться в p45
   S1->name = S2->name;
   S1->count = 0;
   return 0;
}

int tSM::p47() { //  HCPROC -> HCPROC E
   //валидация E была проведена
   ++S1->count;
   return 0;
}

int tSM::p49() { //    BOOL -> $idq

   string name = S1->name;
   if (scope == 2) {// локальная область (let ...)
      if (locals.count(name)) {// локальное имя
         //okay
         return 0;
      }
   }
   if (scope >= 1) {// область параметров
      if (params.count(name)) {// имя параметра
         //okay
         return 0;
      }
```

```
   }

   //переменую типа BOOL обьявить нельзя
   //и как следствие использовать
   ferror_message = "[!]BOOL var:"
               "  '" + name +
               "' boolean variable can not exist";
   return 1;

}

int tSM::p55() { //   CPRED -> HCPRED )
   //валидация вызова предиката
   //как для композиции в p45
   //с дополнительным просмотром типов параметров
   string name = S1->name;
   int count = S1->count;
   int types = S1->types;
   if (scope > 1) {// внури тела let
      if (locals.count(name)) {// локальное имя
         ferror_message = "[!]Predicate application:"
                     " local variable '" + name +
                     "' shadows the predicate!";
         return 1;
      } // if locals ...
   } // if scope ...
   if (scope > 0) {// внутри процедуры
      if (params.count(name)) {// имя параметра
         ferror_message = "[!]Predicate application:"
                     " parameter '" + name +
                     "' shadows the predicate!";
         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, types);
         break;
      }
```

```
      // имя найдено
      if (!ref.test(PROC)) {//не процедура
         ferror_message = "[!]Predicate application:"
                     "  '" + name +
                     "' is not a predicate!";
         return 1;
      }

      if (ref.arity != count) {//число аргументов
         //              не равно числу параметров
         std::ostringstream buf;
         buf << "[!]Predicate 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;
      }

      if (ref.types != types && ref.types != (ID | IDQ) )
{//типы ожидаемых параметров
         //не равен введенным
         std::ostringstream buf;
         buf << "[!]Predicate application: '" << name << "' ";
         buf << "expects ";
         if ((ref.types & ID) == ID) {
            buf << "id ";
         }
         if ((ref.types & IDQ) == IDQ) {
            buf << "idq";
         }

         buf << ", given: ";
         if ((types & ID) == ID) {
            buf << "id ";
```

```
            }
            if ((types & IDQ) == IDQ) {
               buf << "idq";
            }
            buf << " !";

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

      return 0;
   }

   int tSM::p56() { //  HCPRED -> ( $idq
      //вызов предиката
      //валидация будет проведена в p55
      S1->name = S2->name;
      S1->count = 0;
      return 0;
   }

   int tSM::p57() { //  HCPRED -> HCPRED ARG
      ++S1->count;
      S1->types = S2->types;
      return 0;
   }

   int tSM::p58() { //    ARG -> E
      S1->types |= ID;
      return 0;
   }

   int tSM::p59() { //    ARG -> BOOL
      S1->types |= IDQ;
      return 0;
   }

   int tSM::p74() { //    SET -> ( set! $id E )
```

```
   string name = S3->name;
   if(scope == 2) {
      if (!params.count(name) && !locals.count(name)) {
         ferror_message = "[!]Assignment disallowed;:"
                     "  '" + name +
                     "' cannot set variable";
         return 1;
      }
   }

   if (scope == 1) {
      if (!params.count(name)) {
         ferror_message = "[!]Assignment disallowed;:"
                     "  '" + name +
                     "' cannot set variable, missing in
parameters";
         return 1;
      }
   }

   //найти имя в глобальной таблице
   tgName &ref = globals[name];
   //имя найдено
   if (ref.test(PROC)) {//процедура
      //процедуру нельзя использовать
      ferror_message = "[!]Set application:"
                  "  '" + name +
                  "' is a procedure and can not be redefined";
      return 1;
   }
   if (ref.empty()) {
      //ранее имя не встречалось
      //нельзя изменить переменную, которая не
существует
      ferror_message = "[!]Assignment disallowed;:"
                  "  '" + name +
                  "' cannot set variable before its definition";
      return 1;
   }

   return 0;
```

```
}

int tSM::p87() { //    PRED -> HPRED BOOL )

   string name = S1->name;
   int count = S1->count;
   int types = S1->types;
   tgName &ref = globals[name];
   if (ref.empty()) {
      //не было упоминаний
      ref = tgName(PROC | DEFINED, count, types);
   } else if (ref.test(VAR)) {
      ferror_message = "[!]Procedure initialization:"
                  " '" + name +
                  "' was already initialized as variable";
      return 1;
   } else if (ref.test(DEFINED)) {
      ferror_message = "[!]Procedure initialization:"
                  " '" + name +
                  "' was already initialized";
      return 1;
   } else {
      if (ref.arity != count) {
         std::ostringstream buf;
         buf << "[!]Procedure application: '" << name << "' "
            << (ref.test(DEFINED) ? "expects "
                              : "has been called already\n\t with "
            )
            << ref.arity << " argument"
            << (ref.arity != 1 ? "s" : "")
            << ", given: " << count << " !";
         ferror_message = buf.str();
         return 1;
      } else if (ref.types != types && types != (ID | IDQ) ) {
         //тип ожидаемых параметров
         //не равен введенным
         std::ostringstream buf;
         buf << "[!]Procedure application: '" << name << "'
";
         buf << "given: ";
         if ((ref.types & ID) == ID) {
            buf << "id ";
```

```
            }
            if ((ref.types & IDQ) == IDQ) {
               buf << "idq";
            }

            buf << ", expects: ";
            if ((types & ID) == ID) {
               buf << "id ";
            }
            if ((types & IDQ) == IDQ) {
               buf << "idq";
            }
            buf << " !";

            ferror_message = buf.str();
            return 1;
         } else {
            ref.set(DEFINED);
         }
      }
      params.clear();
      scope = 0;
      return 0;
}

int tSM::p88() { //   HPRED -> PDPAR )
      scope = 1;
      return 0;
}

int tSM::p89() { //   PDPAR -> ( define ( $idq
      //определение предиката
      //валиадация имени произойдет в p87
      S1->name = S4->name;
      S1->count = 0;
      return 0;
}

int tSM::p90() { //   PDPAR -> PDPAR $idq
      if (params.count(S2->name)) {
         ferror_message =
               "[!]Predicate definition: in '"
```

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

int tSM::p91() { //   PDPAR -> PDPAR $id
   if (params.count(S2->name)) {
      ferror_message =
            "[!]Predicate definition: in '"
            + S1->name +
            "' duplicate parameter identifier '"
            + S2->name + "'!";
      return 1;
   }
   params.insert(S2->name);
   ++S1->count;
   S1->types |= ID;
   return 0;
}

int tSM::p92() { //    VAR -> ( define $id CONST )
   //определение глобальной переменной
   string name = S3->name;
   tgName &ref = globals[name];
   if (ref.empty()) {//неизвестное имя
      //создать новую учетную запись
      ref.set(VAR);
      ref.set(DEFINED);
      return 0;
   } else if (ref.test(VAR)) {
      if (ref.test(DEFINED)) {
         ferror_message = "[!]Global variable initialization:"
                  " Trying to reinitialize '" + name +
                  "'";
         return 1;
      } else {
```

```
         ref.set(DEFINED);
         return 0;
      }
   } else {
      ferror_message = "[!]Global variable initialization:"
                  " '" + name +
                  "' was already used not as variable";
      return 1;
   }
   return 0;
}

int tSM::p93() { //    PROC -> HPROC LET )
   string name = S1->name;
   int count = S1->count;
   tgName &ref = globals[name];
   if (ref.empty()) {
      //не было упоминаний
      ref = tgName(PROC | DEFINED, count);
   } else if (ref.test(VAR)) {
      ferror_message = "[!]Procedure initialization:"
                  " '" + name +
                  "' was already initialized as variable";
      return 1;
   } else if (ref.test(DEFINED)) {
      ferror_message = "[!]Procedure initialization:"
                  " '" + name +
                  "' was already initialized";
      return 1;
   } else {
      if (ref.arity != count) {
         std::ostringstream buf;
         buf << "[!]Procedure application: '" << name << "' "
            << (ref.test(DEFINED) ? "expects "
                              : "has been called already\n\t with "
            )
            << ref.arity << " argument"
            << (ref.arity != 1 ? "s" : "")
            << ", given: " << count << " !";
         ferror_message = buf.str();
         return 1;
      } else {
```

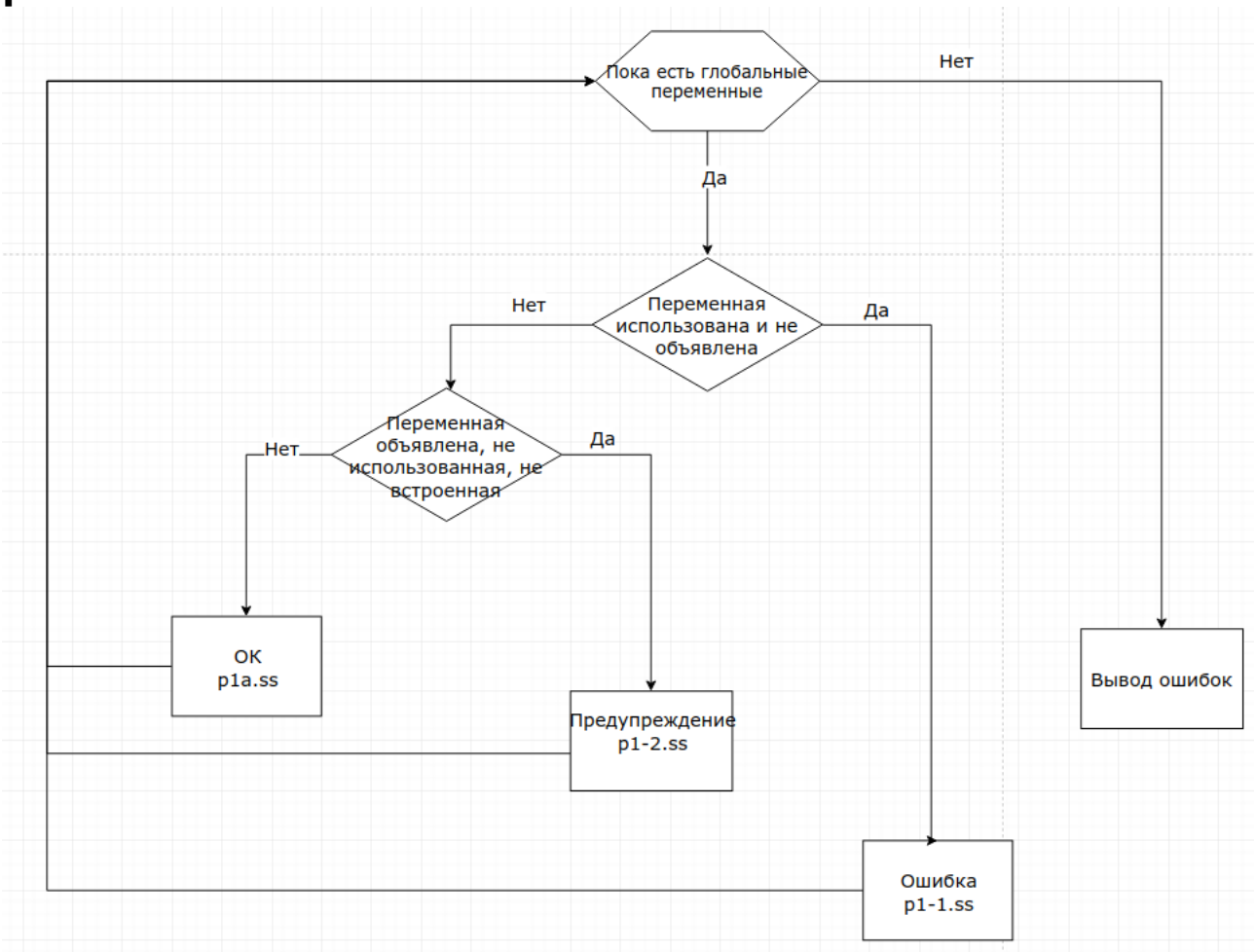
```
            ref.set(DEFINED);
        }
    }
    params.clear();
    scope = 0;
    return 0;
}

int tSM::p94() { //    PROC -> HPROC E )
    string name = S1->name;
    int count = S1->count;
    tgName &ref = globals[name];
    if (ref.empty()) {
        //не было упоминаний
        ref = tgName(PROC | DEFINED, count);
    } else if (ref.test(VAR)) {
        ferror_message = "[!]Procedure initialization:"
                    " '" + name +
                    "' was already initialized as variable";
        return 1;
    } else if (ref.test(DEFINED)) {
        ferror_message = "[!]Procedure initialization:"
                    " '" + name +
                    "' was already initialized";
        return 1;
    } else {
        if (ref.arity != count) {
            std::ostringstream buf;
            buf << "[!]Procedure application: '" << name << "' "
                << (ref.test(DEFINED) ? "expects "
                                : "has been called already\n\t with "
                )
                << ref.arity << " argument"
                << (ref.arity != 1 ? "s" : "")
                << ", given: " << count << " !";
            ferror_message = buf.str();
            return 1;
        } else {
            ref.set(DEFINED);
        }
    }
    params.clear();
```

```
   scope = 0;
   return 0;
}

int tSM::p95() { //   HPROC -> PCPAR )
   scope = 1;
   return 0;
}

int tSM::p97() { //   PCPAR -> ( define ( $id
   S1->name = S4->name;
   S1->count = 0;
   return 0;
}

int tSM::p98() { //   PCPAR -> PCPAR $id
   if (params.count(S2->name)) {
      //p98-1.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::p99() { //    LET -> HLET E )

   locals.clear();
   return 0;
}

int tSM::p100() { //    HLET -> LETLOC )
   scope = 2;
   return 0;
}

int tSM::p102() { //  LETLOC -> ( let (
```

```
    return 0;
}

int tSM::p104() { //  LETVAR -> ( $id E )
    if (locals.count(S2->name)) {
        ferror_message =
                "[!]Local variables definition: '"
                "' duplicate variable initialization'"
                + S2->name + "'!";
        return 1;
    }
    locals.insert(S2->name);
    ++S1->count;
    return 0;
}
//_____________________________
```

Блок схемы и протоколы тестирования:

p1: S -> PROG

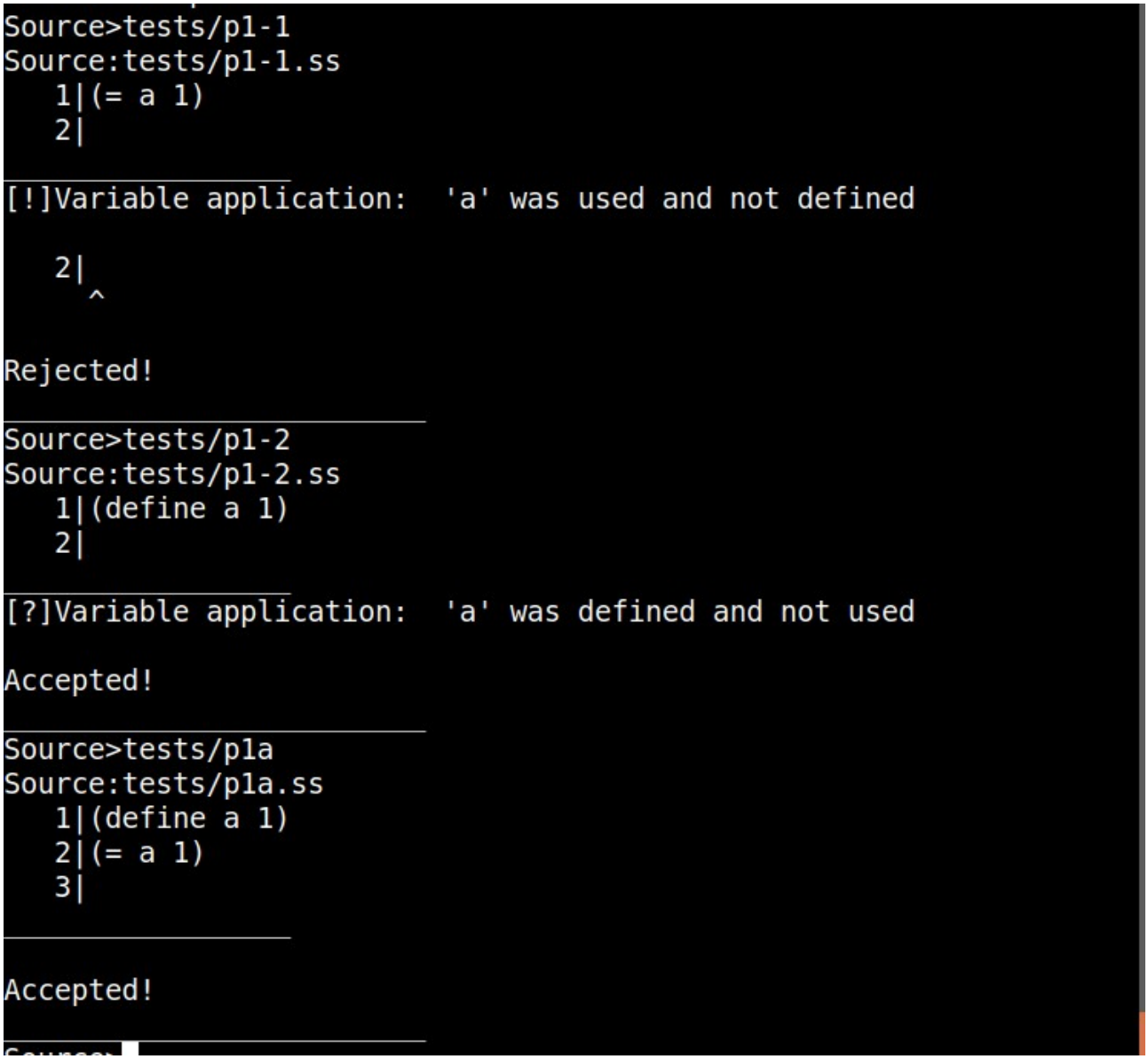


Протокол тестирования:

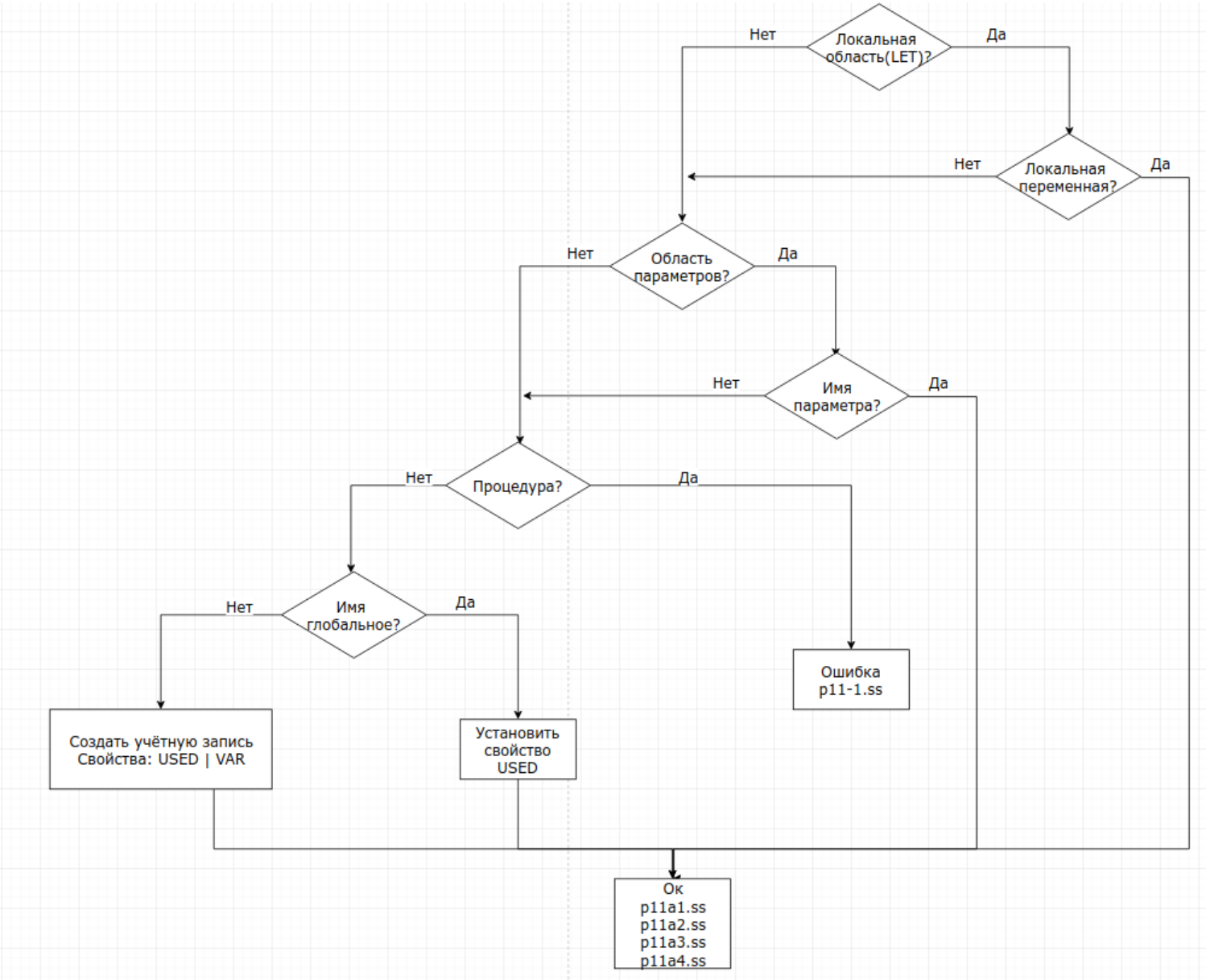
p1-1.ss
(= a 1)

p1-2.ss
(define a 1)

p1a.ss
(define a 1)
(= a 1)



p11: E -> $id



Протокол тестирования:

p11-1.ss
(sin abs)

p11a1.ss
(define (fun) (abs a))
(define a 0)
(fun)

p11a2.ss
(define a 3)
a

p11a3.ss

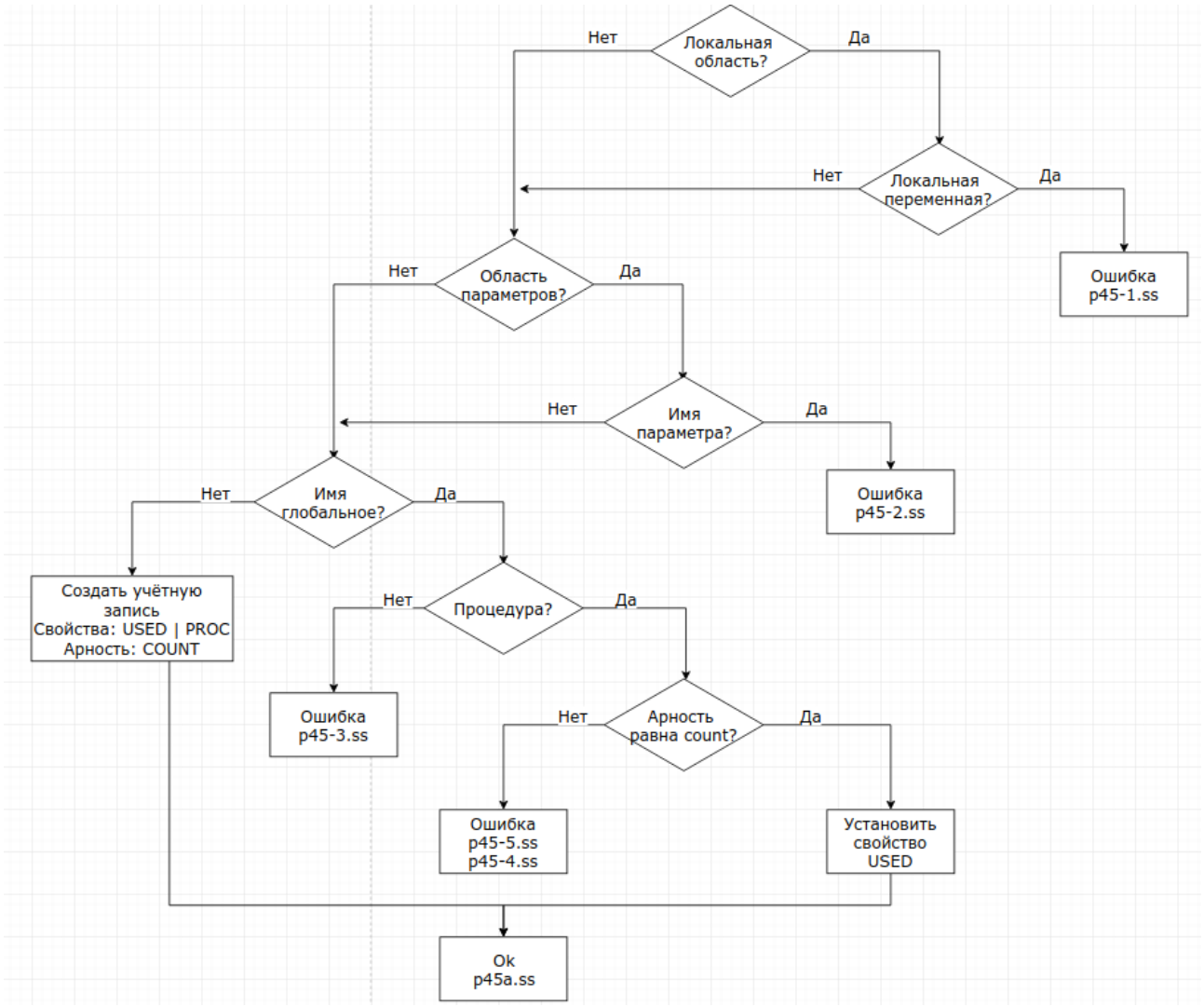
(define (fun a) (+ 1 a))
(fun 4)

p11a4.ss
(define (fun)
(let ((a 1))
a
))(fun)

```
_____________________
Source>tests/p11-1
Source:tests/p11-1.ss
   1|(sin abs)
   2|
_____________________
[!]Procedure application:  'abs' is a procedure and can not be used
 as a variable
   1|(sin abs)
            ^

Rejected!

_____________________
Source>tests/p11a1
Source:tests/p11a1.ss
   1|(define (fun) (abs a))
   2|(define a 0)
   3|(fun)
   4|
_____________________

Accepted!

_____________________
Source>tests/p11a2
Source:tests/p11a2.ss
   1|(define a 3)
   2|a
   3|
_____________________

Accepted!

_____________________
Source>tests/p11a3
Source:tests/p11a3.ss
   1|(define (fun a) (+ 1 a))
   2|(fun 4)
   3|
_____________________

Accepted!
```

```
Source>tests/p11a4
Source:tests/p11a4.ss
   1|(define (fun)
   2| (let ((a 1))
   3|  a
   4| )
   5|)
   6|
   7|(fun)
   8|

Accepted!
```

p45: CPROC -> HCPROC)
В процедурах не смотрится тип параметров, так как параметры процедуры только E.



Протокол тестирования:
p45-1.ss
(define(f x)(let((abs 1))(abs x)))

p45-2.ss
(define(f x abs)(abs x))

p45-3.ss
(e 0) pi

p45-4.ss
(abs 1 2) 0

p45-5.ss
(define(g x) (f x (f x)))
(define(f x) (* x x))

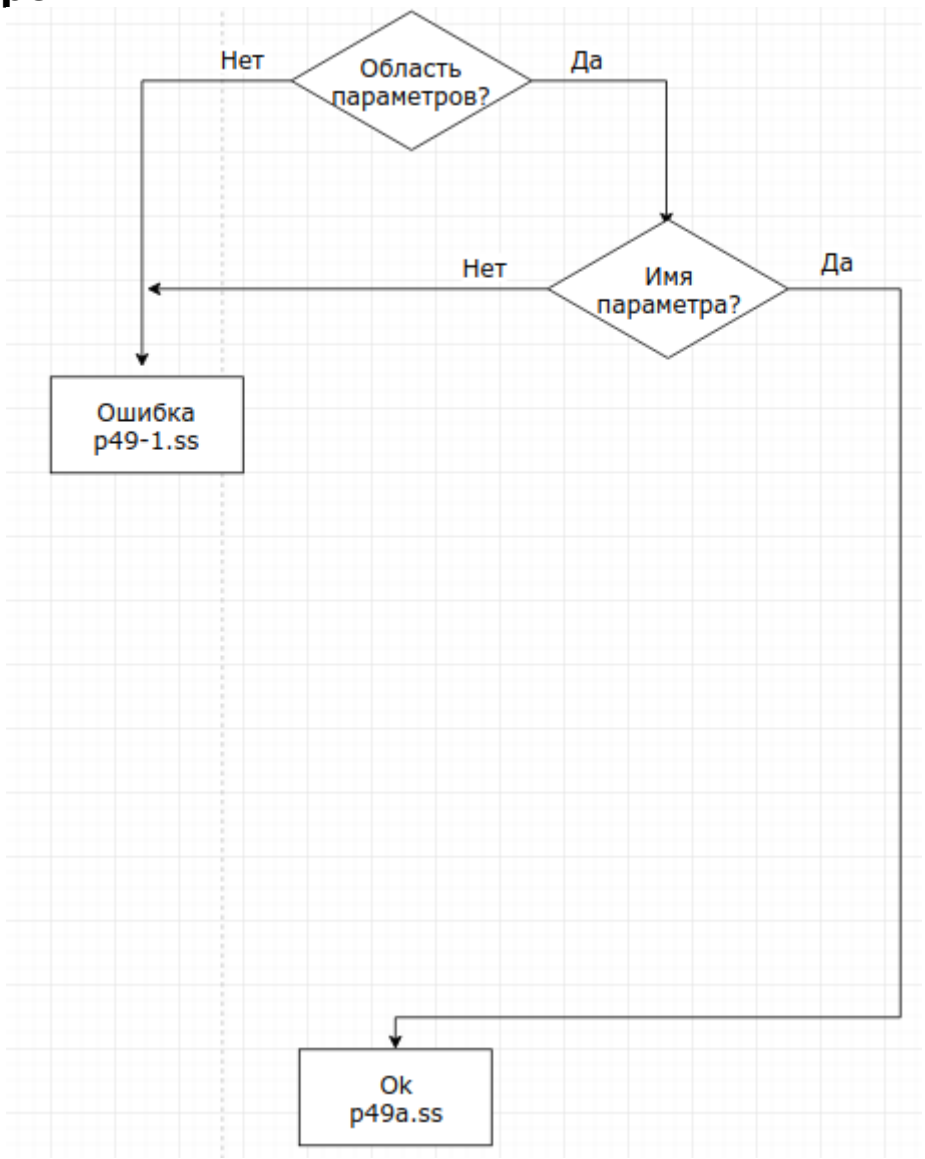
p45a.ss
(abs pi)

```
Source>tests/p45-1
Source:tests/p45-1.ss
   1|; p45-1
   2|(define(f x)(let((abs 1))(abs x)))
   3|
_____________________
[!]Procedure application: local variable 'abs' shadows the procedur
e!
   2|(define(f x)(let((abs 1))(abs x)))
                                  ^

Rejected!
_____________________________
Source>tests/p45-2
Source:tests/p45-2.ss
   1|; p45-2
   2|(define(f x abs)(abs x))
   3|
_____________________
[!]Procedure application: parameter 'abs' shadows the procedure!
   2|(define(f x abs)(abs x))
                        ^

Rejected!
_____________________________
Source>tests/p45-3
Source:tests/p45-3.ss
   1|; p45-3
   2|(e 0) pi
   3|
_____________________
[!]Procedure application:  'e' is not a procedure!
   2|(e 0) pi
         ^

Rejected!
```

```
_____________________
Source>tests/p45-4
Source:tests/p45-4.ss
   1|; p45-4
   2|(abs 1 2) 0
   3|

_________________
[!]Procedure application: 'abs' expects 1 argument, given: 2 !
   2|(abs 1 2) 0
              ^

Rejected!

_____________________
Source>tests/p45-5
Source:tests/p45-5.ss
   1|; p45-5
   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>tests/p45a
Source:tests/p45a.ss
   1|; p45a
   2|(abs pi)
   3|

_________________

Accepted!
```

p49: BOOL -> $idq

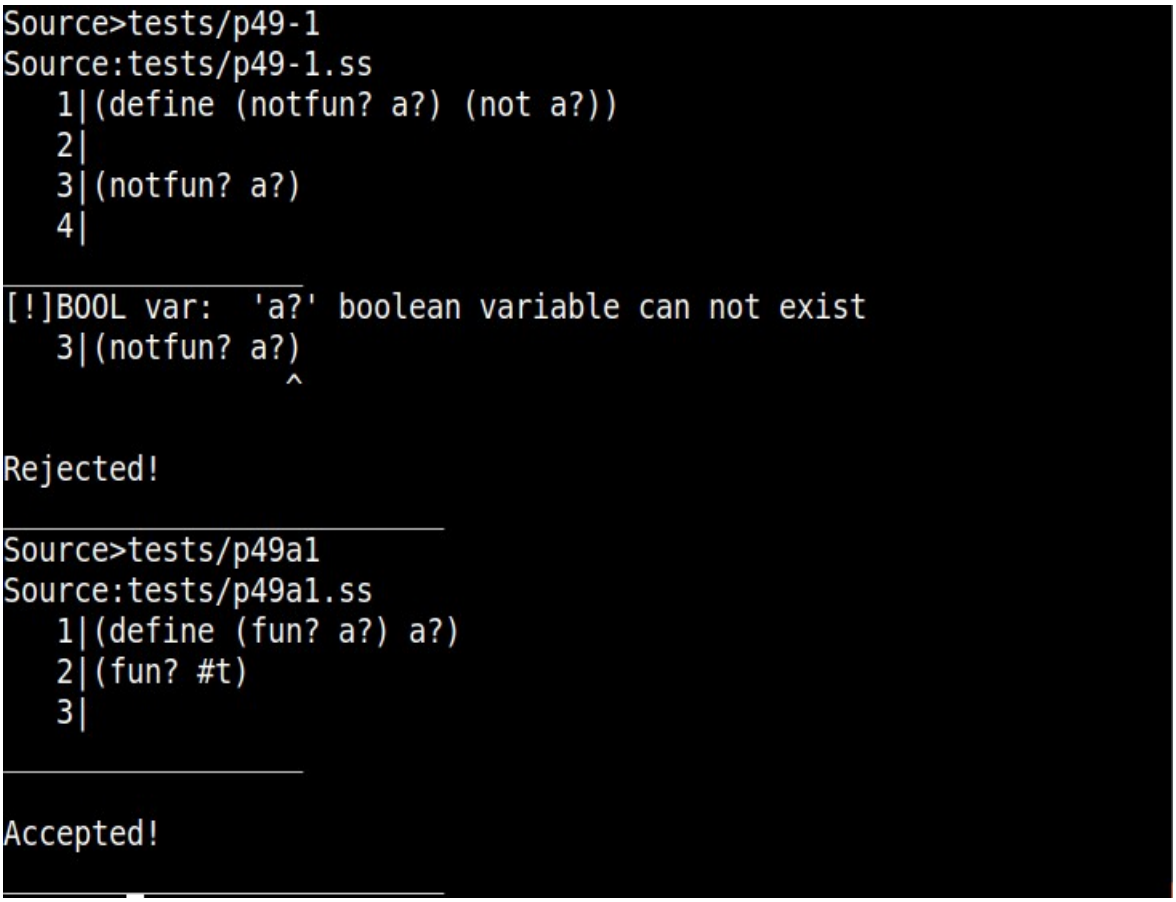
Так как в грамматике mlisp19 переменные типа BOOL не могут объявляться как глобальные и локальные, $idq может существовать только в рамках предиката в области параметров.



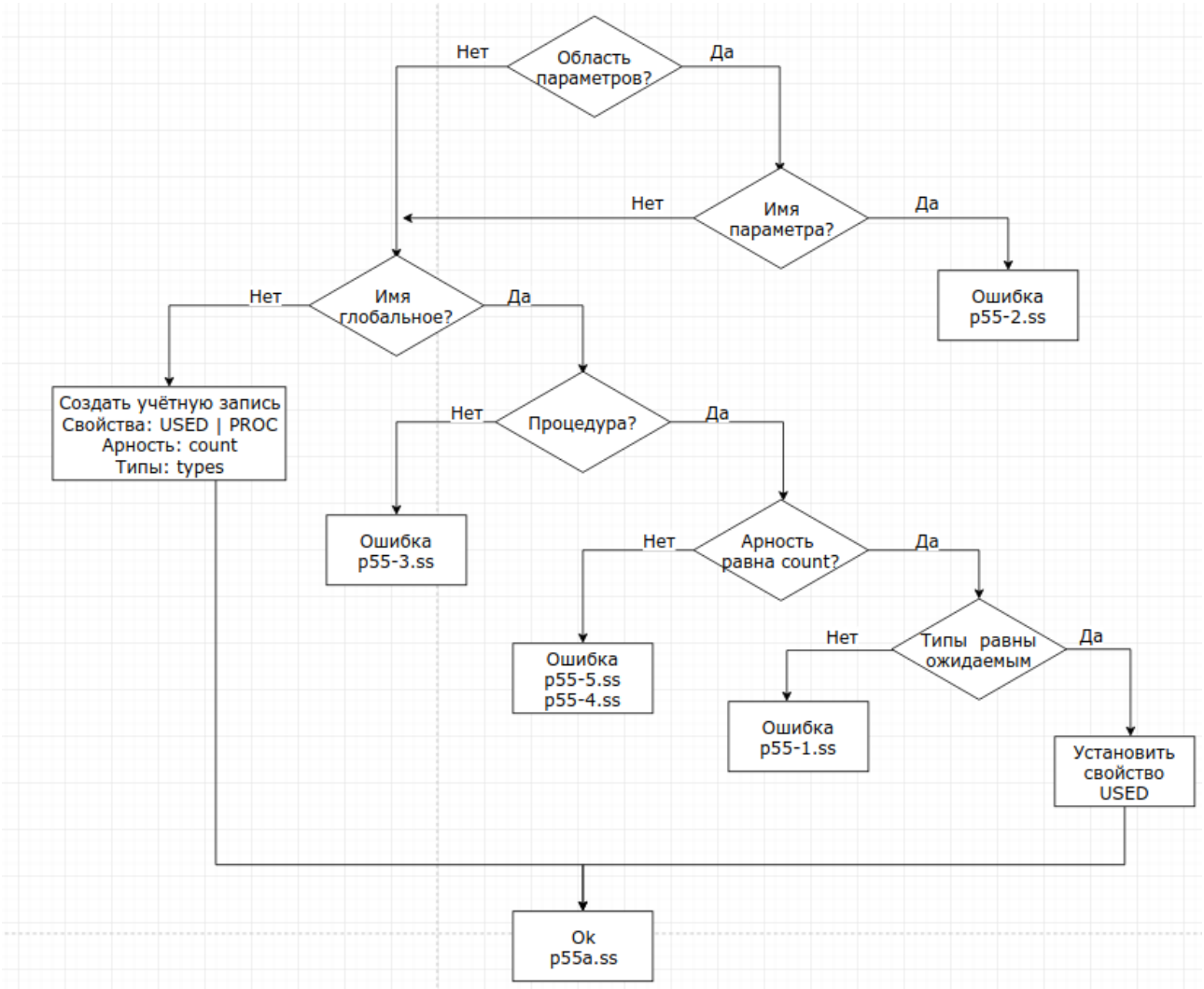
Протокол тестирования:

p49a1.ss
(define (fun? a?) a?)
(fun? #t)

p49-1.ss
(define (notfun? a?) (not a?))
(notfun? a?)



p55: CPRED -> HCPRED)



Протокол тестирования:

p55-1.ss
(define(f? x) (eq? x 3))
(f? #t)

p55-2.ss
(define(f? x eq?)(eq? x))

p55-3.ss
(define(f? x?) (x? 3))

p55-4.ss
(define(f? x) (eq? x 3))
(f? 3 4)

p55-5.ss

(define(g? x y) (f? x y))
(define(f? x) (eq? x 3))

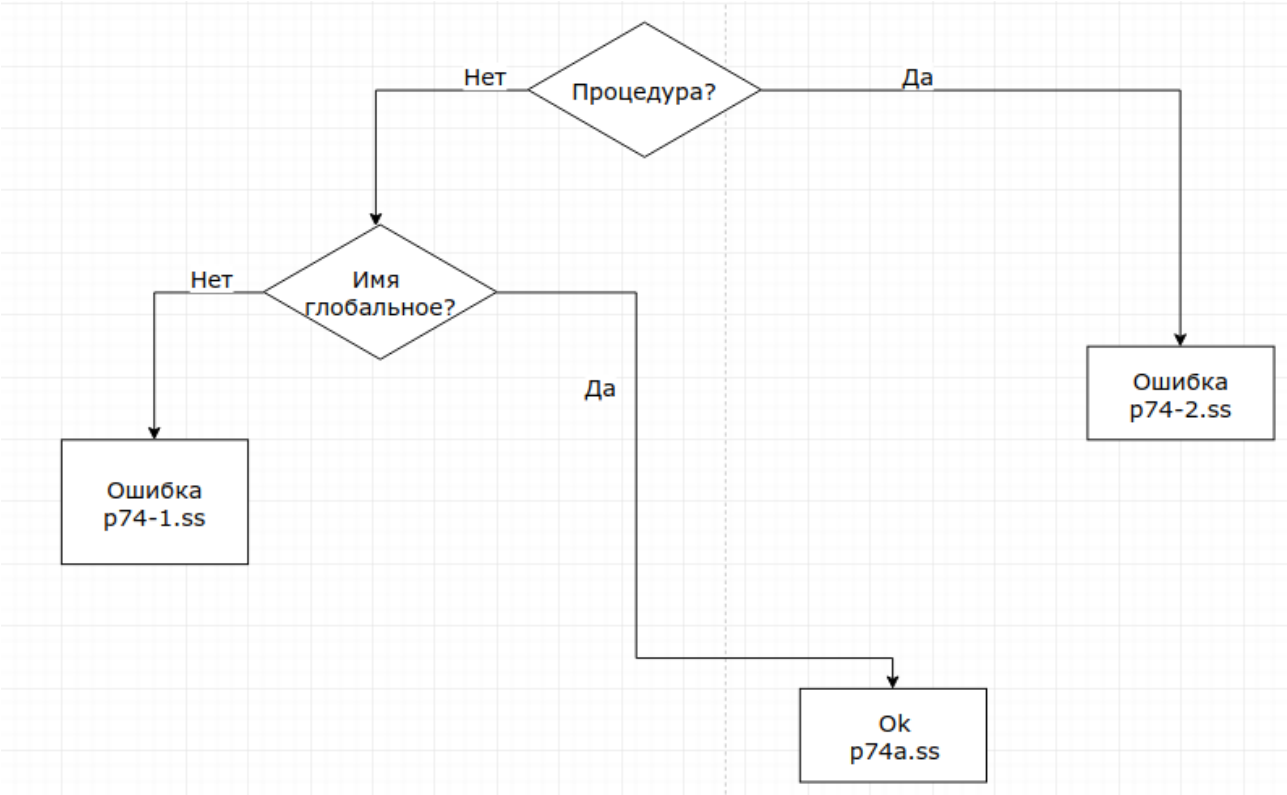
p55a.ss
(define(f? x) (eq? x 3))
(f? 3)

```
Source>tests/p55-1
Source:tests/p55-1.ss
   1|(define(f? x) (eq? x 3))
   2|(f? #t)
   3|
____________________
[!]Predicate application: 'f?' expects id , given: idq !
   3|
     ^

Rejected!
_____________________________
Source>tests/p55-2
Source:tests/p55-2.ss
   1|(define(f? x eq?)(eq? x))
   2|
____________________
[!]Predicate application: parameter 'eq?' shadows the predicate!
   1|(define(f? x eq?)(eq? x))
                          ^

Rejected!
_____________________________
Source>tests/p55-3
Source:tests/p55-3.ss
   1|(define(f? x?) (x? 3))
   2|
____________________
[!]Predicate application: parameter 'x?' shadows the predicate!
   1|(define(f? x?) (x? 3))
                     ^

Rejected!
_____________________________
```

```
_____________________________
Source>tests/p55-4
Source:tests/p55-4.ss
   1|(define(f? x) (eq? x 3))
   2|(f? 3 4)
   3|
   4|

_____________________
[!]Predicate application: 'f?' expects 1 argument, given: 2 !
   4|
     ^

Rejected!

_____________________________
Source>tests/p55-5
Source:tests/p55-5.ss
   1|(define(g? x y) (f? x y))
   2|(define(f? x) (eq? x 3))
   3|

_____________________
[!]Procedure application: 'f?' has been called already
         with 2 arguments, given: 1 !
   3|
     ^

Rejected!

_____________________________
Source>tests/p55a
Source:tests/p55a.ss
   1|(define(f? x) (eq? x 3))
   2|(f? 3)
   3|

____________________

Accepted!

_____________________________
```

p74: SET -> (set! $id E)

Грамматика не позволяет писать SET в функциях.



Протокол тестирования:

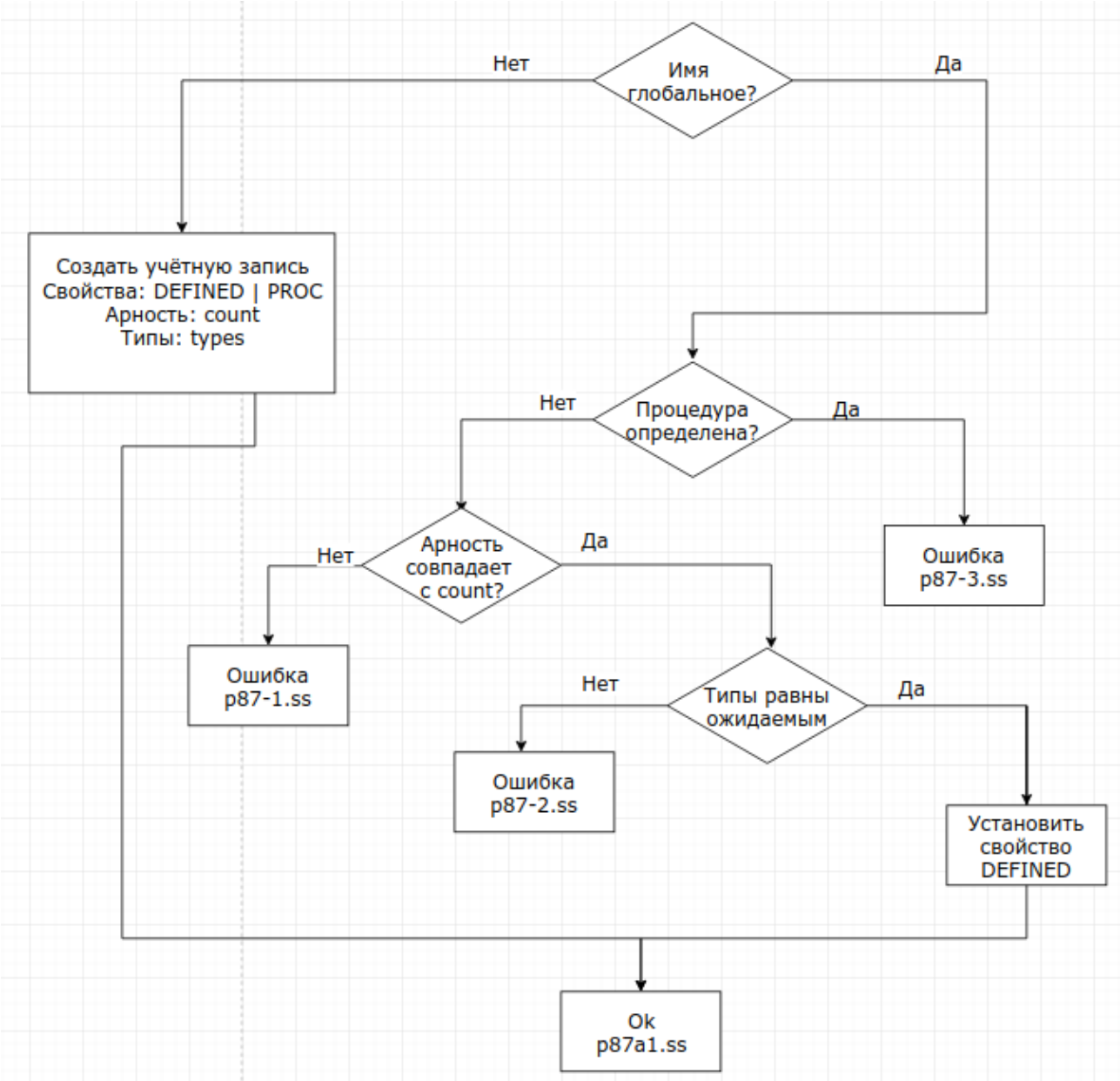
p74-1.ss
(set! a 20)

p74-2.ss
(define (f x) x)
(set! f 4)

p74a.ss
(define a 3)
a
(set! a 4)
a

```
Source>tests/p74a
Source:tests/p74a.ss
   1|(define a 3)
   2|a
   3|(set! a 4)
   4|a
   5|
_____________________

Accepted!

_____________________________
Source>tests/p74-1
Source:tests/p74-1.ss
   1|(set! a 20)
   2|
_____________________
[!]Assignment disallowed;:  'a' cannot set variable before its def
inition
   2|
     ^

Rejected!

_____________________________
Source>tests/p74-2
Source:tests/p74-2.ss
   1|(define (f x) x)
   2|
   3|(set! f 4)
   4|
_____________________
[!]Set application:  'f' is a procedure and can not be redefined
   4|
     ^

Rejected!

_____________________________
Source>
```

p87: PRED -> HPRED BOOL)



Протокол тестирования:

p87-1.ss
(define(g? x) (f? x (f? x)))
(define(f? x) (eq? x x))

p87-2.ss
(define(g? x) (f? x))
(define(f? x?) (eq? x? x?))

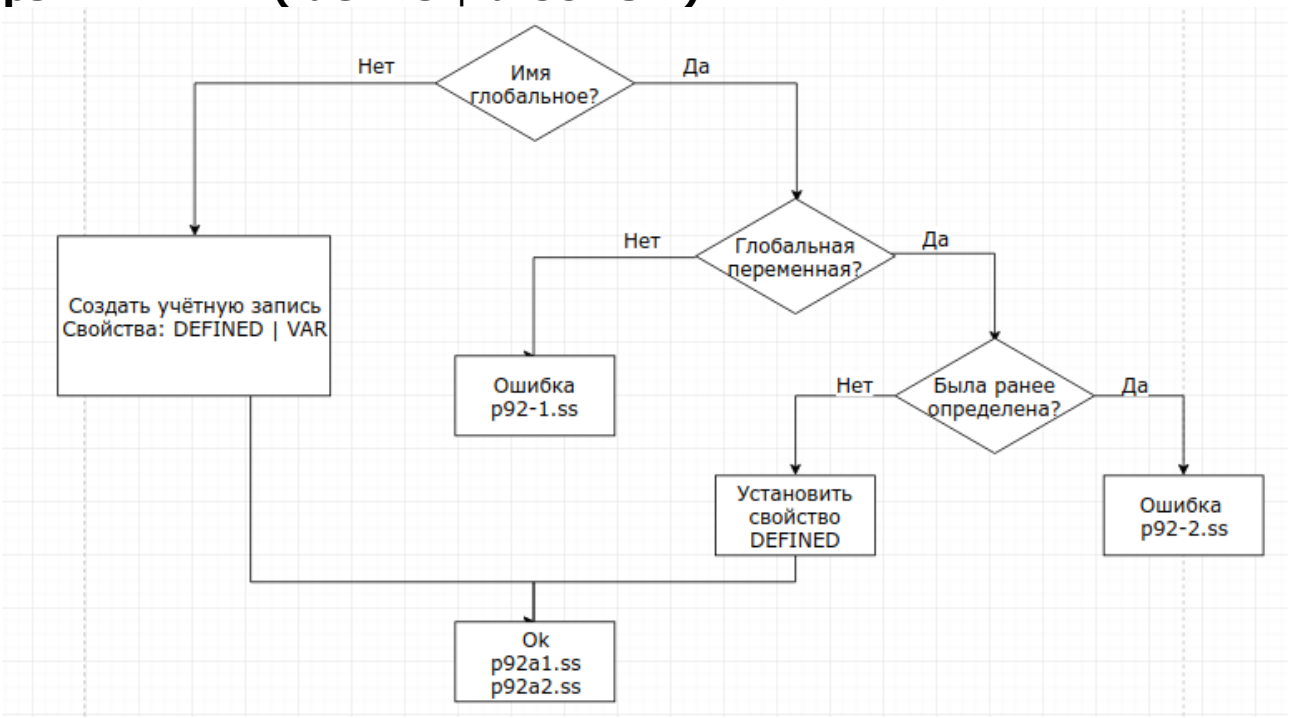
p87-3.ss
(define (f? e) (eq? e e))
(define (f? e y) (eq? e y))

p87a.ss
(define (f? w) (eq? w w))
(f? 3)

```
_____________________________
Source>tests/p87-1
Source:tests/p87-1.ss
   1|(define(g? x) (f? x (f? x)))
   2|(define(f? x) (eq? x x))
   3|
_____________________
[!]Predicate application: 'f?' has been called already
         with 1 argument, given: 2 !
   1|(define(g? x) (f? x (f? x)))
                           ^

Rejected!
_____________________________
Source>tests/p87-2
Source:tests/p87-2.ss
   1|(define(g? x) (f? x))
   2|(define(f? x?) (eq? x? x?))
   3|
_____________________
[!]Procedure application: 'f?' given: id , expects: idq !
   3|
     ^

Rejected!
_____________________________
Source>tests/p87-3
Source:tests/p87-3.ss
   1|(define (f? e) (eq? e e))
   2|(define (f? e y) (eq? e y))
   3|
_____________________
[!]Procedure initialization: 'f?' was already initialized
   3|
     ^

Rejected!
_____________________________
```

```
Source>tests/p87a
Source:tests/p87a.ss
   1|(define (f? w) (eq? w w))
   2|(f? 3)
   3|

Accepted!
```

p92: VAR -> (define $id CONST)



p92-1.ss
(define (fun) 0)
(define fun 2)

p92-2.ss
(define fun 0)
(define fun 2)

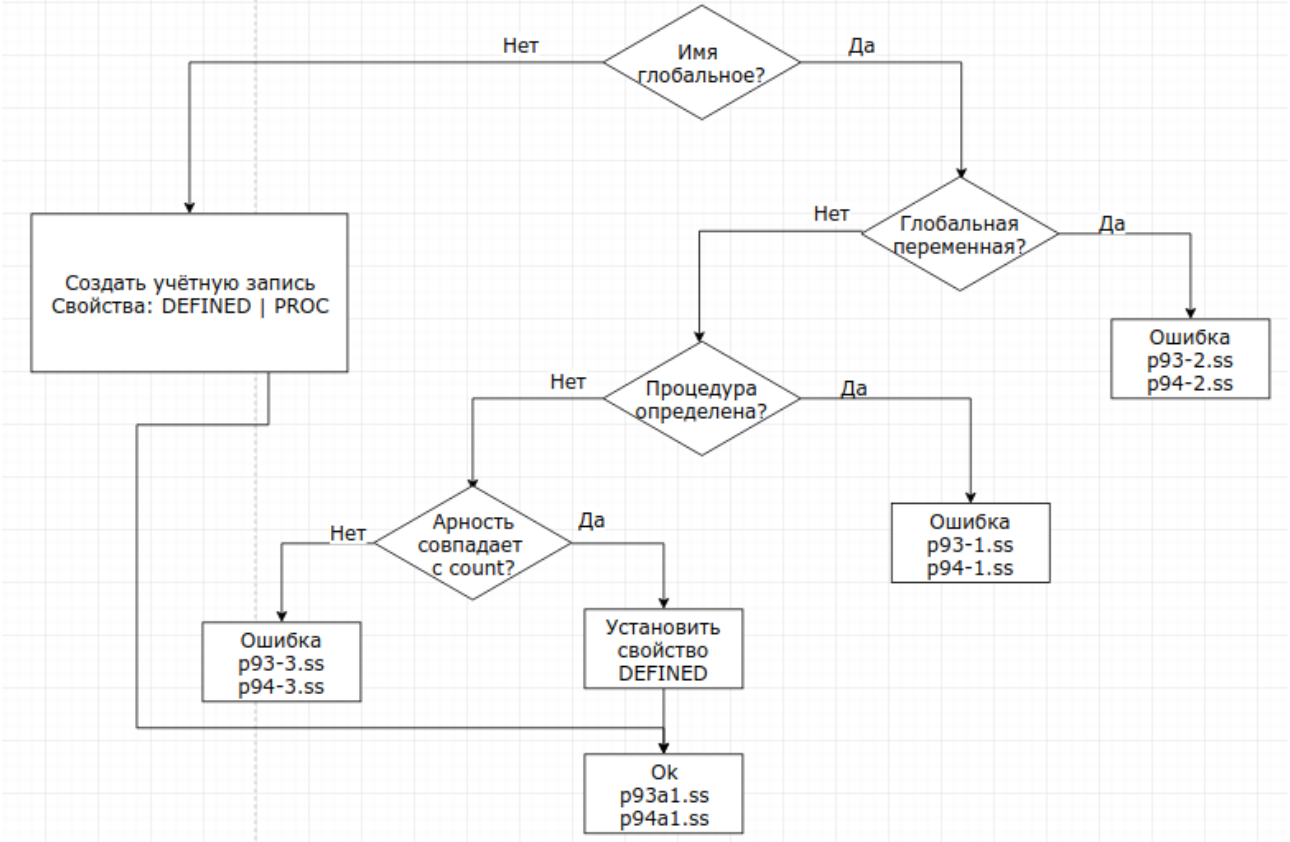
p92a1.ss
(define a 1)

p92a2.ss
(define (fun) p)
(define p 1)(fun)

```
_____________________________
Source>tests/p92-1
Source:tests/p92-1.ss
   1|(define (fun) 0)
   2|(define fun 2)
   3|

___________________
[!]Global variable initialization: 'fun' was already used not as va
riable
   3|
     ^

Rejected!

_____________________________
Source>tests/p92-2
Source:tests/p92-2.ss
   1|(define fun 0)
   2|(define fun 2)
   3|

___________________
[!]Global variable initialization: Trying to reinitialize 'fun'
   3|
     ^

Rejected!

_____________________________
Source>tests/p92a1
Source:tests/p92a1.ss
   1|(define a 1)
   2|

___________________
[?]Variable application:  'a' was defined and not used

Accepted!

_____________________________
```

```
_____________________________
Source>tests/p92a2
Source:tests/p92a2.ss
   1|(define (fun) p)
   2|(define p 1)(fun)
   3|

___________________

Accepted!
```

p93: PROC -> HPROC LET)
p94: PROC -> HPROC E)
Диаграмма для них аналогична:



Протокол тестирования:
p93-1.ss
(define (fun a b) 0)
(define (fun) 1)
p94-1.ss
(define (fun a b) 0)
(define (fun) (let((a 1)) 1))

p93-2.ss
(define fun 0)
(define (fun) 1)
p94-2.ss
(define fun 0)
(define (fun) (let((a 1)) 1))

p93-3.ss
(define (fun) (calc))
(define (calc a) a)
(fun)
p93-3.ss

(define (fun) (calc))
(define (calc a) (let((a 1)) 1))
(fun)

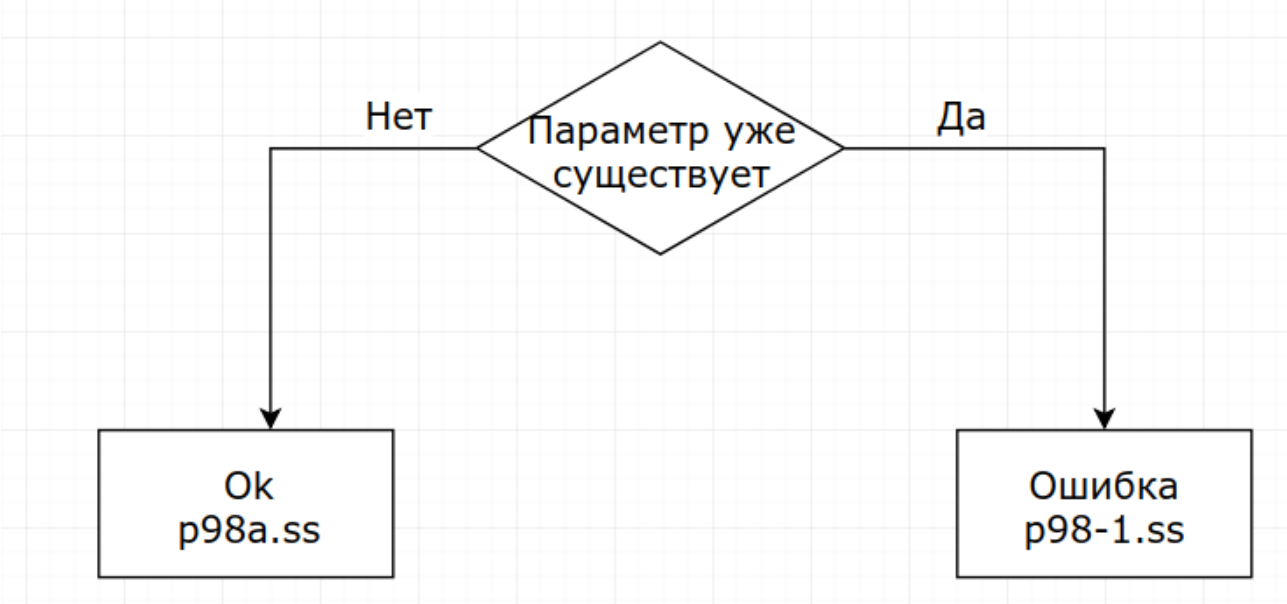
p93a1.ss
(define (fun) 1)
p93a1.ss
(define (fun) (let((a 1)) 1))

```
_____________________
Source>tests/p93-1
Source:tests/p93-1.ss
   1|(define (fun a b) 0)
   2|(define (fun) 1)
   3|

_________________
[!]Procedure initialization: 'fun' was already initialized
   3|
     ^


Rejected!

_____________________
Source>tests/p94-1
Source:tests/p94-1.ss
   1|(define (fun a b) 0)
   2|(define (fun) (let
   3|               ((a 1))
_____________________
Source>tests/p93-2
Source:tests/p93-2.ss
   1|(define fun 0)
   2|(define (fun) 1)
   3|

_________________
[!]Procedure initialization: 'fun' was already initialized as varia
ble
   3|
     ^


Rejected!

_____________________
Source>tests/p94-2
Source:tests/p94-2.ss
   1|(define fun 0)
   2|(define (fun) (let((a 1)) 1))
   3|

_________________
[!]Procedure initialization: 'fun' was already initialized as varia
ble
   3|
     ^


Rejected!
```

```
Source>tests/p93-3
Source:tests/p93-3.ss
   1|(define (fun) (calc))
   2|(define (calc a) a)
   3|(fun)
   4|

[!]Procedure application: 'calc' has been called already
        with 0 arguments, given: 1 !
   3|(fun)
     ^

Rejected!

Source>tests/p94-3
Source:tests/p94-3.ss
   1|(define (fun) (calc))
   2|(define (calc a) (let((a 1)) 1))
   3|(fun)
   4|

[!]Procedure application: 'calc' has been called already
        with 0 arguments, given: 1 !
   3|(fun)
     ^

Rejected!
```

```
Source>tests/p93a1
Source:tests/p93a1.ss
   1|(define (fun) 1)
   2|

[?]Procedure application:  'fun' was defined and not used

Accepted!

Source>tests/p94a1
Source:tests/p94a1.ss
   1|(define (fun) (let((a 1)) 1) )
   2|

[?]Procedure application:  'fun' was defined and not used

Accepted!
```

p98: PCPAR -> PCPAR $id



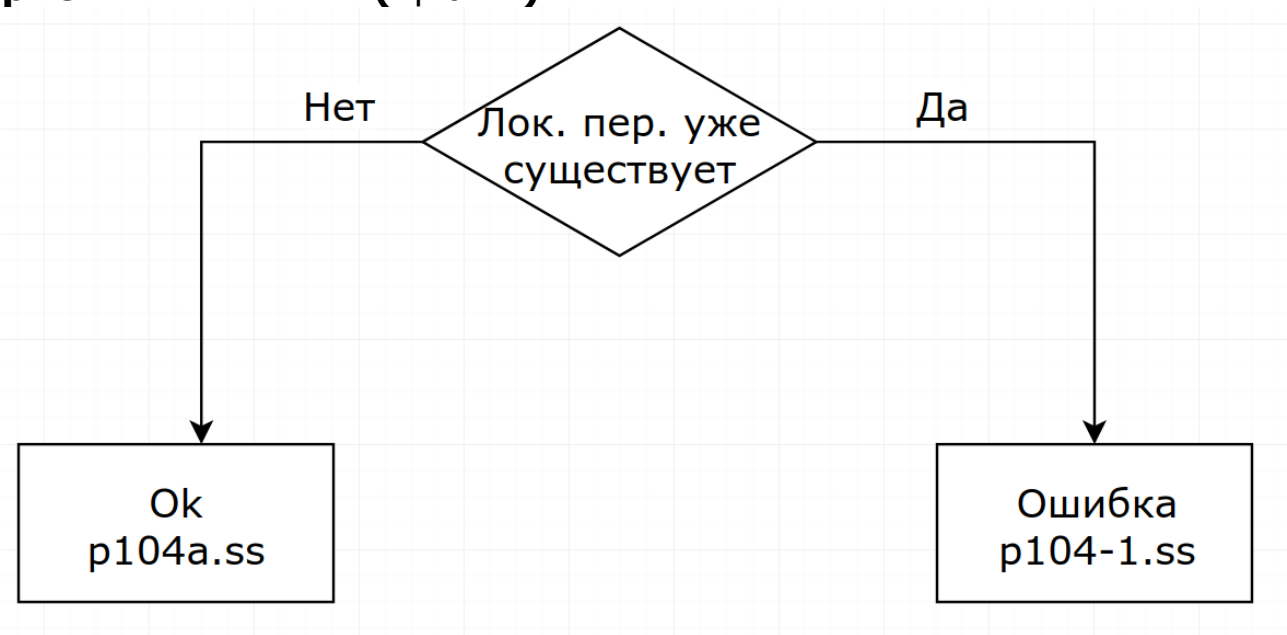
Протокол тестирования:

p98-1.ss
(define(f x x) x)

p98a.ss
(define(f x y) y)

```
Source>tests/p98-1
Source:tests/p98-1.ss
   1|; p98-1
   2|(define(f x x) x )
   3|

[!]Procedure definition: in 'f' duplicate parameter identifier 'x'!
   2|(define(f x x) x )
                  ^

Rejected!

Source>tests/p98a
Source:tests/p98a.ss
   1|; p98a
   2|(define(f x y) y)
   3|

[?]Procedure application:  'f' was defined and not used

Accepted!
```

p104: LETVAR -> ($id E)



p104-1.ss
(define (f) (let((a 1) (a 2)) a))

p104a.ss
(define (f) (let((a 1) (b 2)) a))

```
Source>tests/p104-1
Source:tests/p104-1.ss
   1|(define (f) (let( (a 1) (a 2) ) a) )
   2|
_________________
[!]Local variables definition: '' duplicate variable initialization
'a'!
   1|(define (f) (let( (a 1) (a 2) ) a) )
                                  ^

Rejected!
_____________________________
Source>tests/p104a
Source:tests/p104a.ss
   1|(define (f) (let( (a 1) (b 2) ) a) )
   2|
_________________
[?]Procedure application:  'f' was defined and not used

Accepted!
_____________________________
```

Выводы по проделанной работе.

Проделав курсовую работу, я изучил и сконструировал семантический анализатор, реализующий заданные правила, для языка МИКРОЛИСП на базе класса tSM для грамматики mlisp19. Для каждого алгоритма анализа мною был разработан сценарий тестирования, покрывающий все ветвления алгоритма, однако, поскольку большое количество тестов очевидны, однотипны или имеют невероятно большой объём незначащего текста, некоторые из них только упомянуты в данном отчёте. Для удобства навигации имя каждого тестового файла содержит имя продукции атрибутов, которой предназначен тест.

В процессе работы над курсовым проектом, я тщательно проанализировал заданные мне правила. В процессе написания кода оказалось, что многие части анализатора очень похожи друг на друга, а отдельные алгоритмы даже имеют одинаковую реализацию, это наблюдение сыграло большую роль, поскольку код можно было заимствовать из других элементов программы и проверять работу одной части анализатора на примере другой.

Отдельной сложностью при разработке стал тот факт, что локальная область видимости вложена в область видимости параметров, которая в свою очередь вложена в глобальную область, поэтому приходится часто проверять в какой из областей находится текущая позиция.

Я познакомился с реализацией такого сложного и комплексного понятия как семантический анализатор. Мне кажется, в моей дальнейшей учебной и рабочей деятельности пригодятся навыки, которые я приобрел.

Также при создании этого проекта мной был разработан серьёзный тестовый сценарий: при тестах я проверял не только написанной мной код, но и неоднократно редактровал составленные диаграммы. Причём, как правило после этого требовалось проверить еще больше кода и составить еще больше тестов для новых элементов анализатора.

Семантический анализатор прекрасно себя показал на контрольных задачах, примерах и множественных тестах. Работу осуществил в полном объёме.