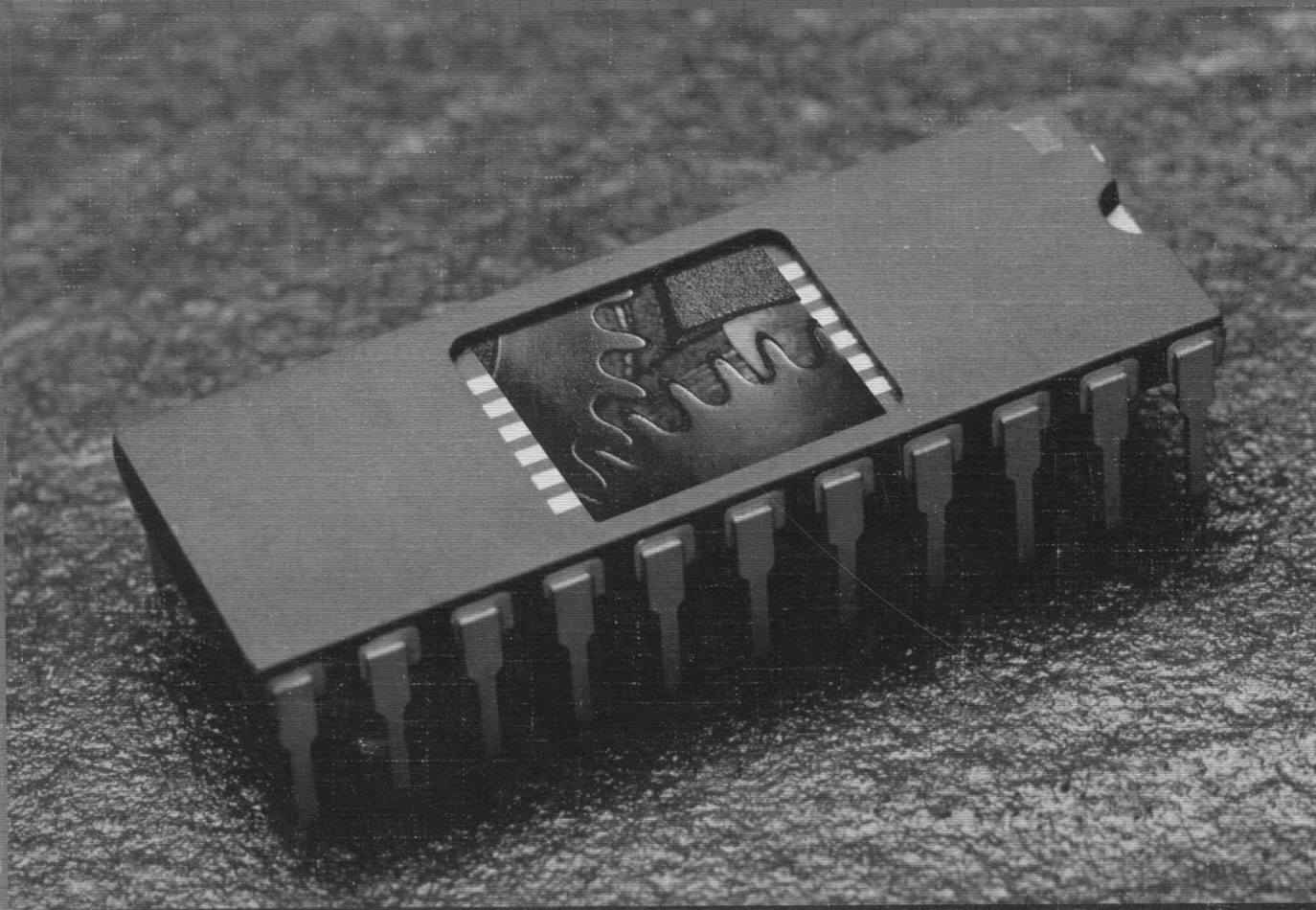


*Министерство образования и науки Украины*

# АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ УПРАВЛЕНИЯ И ПРИБОРЫ АВТОМАТИКИ



'2001  
124

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ

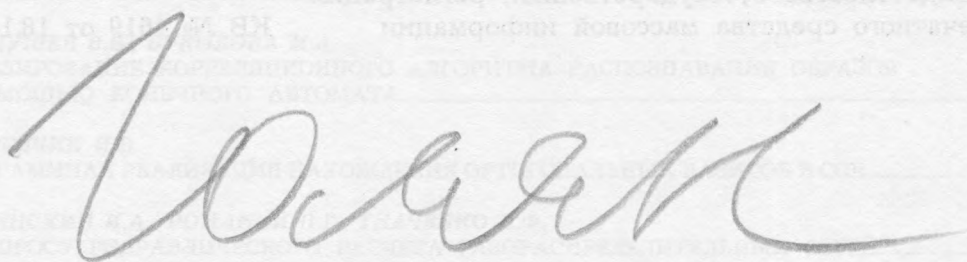
ХАРЬКОВСКИЙ НАЦИОНАЛЬНЫЙ  
УНИВЕРСИТЕТ РАДИОЭЛЕКТРОНИКИ

# АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ УПРАВЛЕНИЯ И ПРИБОРЫ АВТОМАТИКИ

Всеукраинский межведомственный  
научно-технический сборник

Основен в 1965 г.

Выпуск 124



Харьков

2003

## СОДЕРЖАНИЕ

|                                                                                                                                                                                                                                   |     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| НАТАЛУХА Ю.В., АФАНАСЬЕВ В.А., КЛИМОВА Н.П.<br>ОБ ОДНОМ КРИТЕРИИ ПРОВЕРКИ ЛИНЕЙНОЙ НЕЗАВИСИМОСТИ ФУНКЦИЙ.....                                                                                                                     | 4   |
| ЛЕЩИНСКАЯ Е.Л.<br>ИДЕНТИФИКАЦИЯ СЕМАНТИЧЕСКИХ ОШИБОК ПРИ СИНТАКСИЧЕСКОМ АНАЛИЗЕ.....                                                                                                                                              | 8   |
| ТИХОНОВ В.А., ЗУБКОВ О.В., СИДОРОВ Г.И.<br>ГЕНЕРИРОВАНИЕ КВАДРАТУРНЫХ СОСТАВЛЯЮЩИХ СЛУЧАЙНОГО УЗКОПОЛОСНОГО<br>АНАЛИТИЧЕСКОГО СИГНАЛА НА НЕСУЩЕЙ ЧАСТОТЕ ПО ЗАДАННЫМ ПАРАМЕТРАМ<br>СПЕКТРАЛЬНОЙ ПЛОТНОСТИ МОЩНОСТИ.....           | 18  |
| ДОРОВСКОЙ В.А.<br>ЭНТРОПИЙНАЯ МОДЕЛЬ ДИАЛОГОВОГО ПОВЕДЕНИЯ ПАРТНЕРОВ.....                                                                                                                                                         | 23  |
| ГОЛЯН В.В.<br>VHDL-МОДЕЛЬ ПРОЕКТИРОВАНИЯ ПРЕОБРАЗОВАТЕЛЕЙ КОДОВ.....                                                                                                                                                              | 27  |
| КАНЦЕДАЛ С.А., КОСТИКОВА М.В.<br>О ДВУХ АЛГОРИТМАХ ДЛЯ РЕШЕНИЯ ОБЩЕЙ ЗАДАЧИ ТЕОРИИ РАСПИСАНИЙ.....                                                                                                                                | 31  |
| СОРОЧАН М.В.<br>МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПРИ ПОСТРОЕНИИ ТРЕБОВАНИЙ<br>К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ.....                                                                                                                     | 37  |
| ГВОЗДИНСКИЙ А.Н., ЯКИМОВА Н.А.<br>ЛОГИКО-АЛГЕБРАИЧЕСКИЕ МОДЕЛИ СИСТЕМ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ.....                                                                                                                                   | 43  |
| АЛИПОВ Н.В., АЛИПОВ И.Н., ХИЛЬ М.И., РЕВЕЗЮК Л.Н.<br>ПРИМЕРЫ ПОСТРОЕНИЯ ПОСЛЕДОВАТЕЛЬНЫХ АЛГОРИТМОВ ПОИСКА ТОЧКИ<br>С ХАРАКТЕРНЫМ ПРИЗНАКОМ, ПОМЕХОУСТОЙЧИВЫХ К СИММЕТРИЧНЫМ<br>НЕРЕГУЛЯРНЫМ ВИРТУАЛЬНЫМ ПОСЛЕДОВАТЕЛЬНОСТЯМ..... | 47  |
| ДЮВКО Г.Ф., ХАРЧЕНКО С.Л.<br>КОМПОНЕНТНАЯ ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ ДЛЯ ОПИСАНИЯ МОДЕЛЕЙ<br>ЦИФРОВЫХ УСТРОЙСТВ.....                                                                                                             | 55  |
| СИНОТИН А.М.<br>ИССЛЕДОВАНИЕ ВЛИЯНИЯ ЭЛЕМЕНТОВ РАДИОЭЛЕКТРОННОГО АППАРАТА<br>НА ЭФФЕКТИВНУЮ ТЕПЛОПРОВОДНОСТЬ В ОСНОВНЫХ НАПРАВЛЕНИЯХ.....                                                                                         | 61  |
| СИТНИКОВ Д.Э., ВИЛЬЧИНСКАЯ О.С., КРАВЕЦ Н.С.<br>АЛГЕБРАИЧЕСКИЙ МЕТОД НАХОЖДЕНИЯ АППРОКСИМАЦИЙ ПРИБЛИЖЕННЫХ<br>МНОЖЕСТВ И ФОРМИРОВАНИЕ НА ИХ ОСНОВЕ ЛОГИЧЕСКИХ ПРАВИЛ.....                                                         | 66  |
| ДЮВКО Г.Ф., ПОЛУНИН Р.А.<br>РАЗРАБОТКА БАЗ ЗНАНИЙ ДЛЯ ПРОВЕДЕНИЯ ЛОГИЧЕСКОГО КОНТРОЛЯ<br>В КОМПЬЮТЕРНЫХ ОБУЧАЮЩИХ СИСТЕМАХ.....                                                                                                   | 72  |
| КЛИМУШЕВ В.В., БУКОЛОВА М.А.<br>МОДЕЛИРОВАНИЕ КОРРЕЛЯЦИОННОГО АЛГОРИТМА РАСПОЗНАВАНИЯ ОБРАЗОВ<br>С ПОМОЩЬЮ КОНЕЧНОГО АВТОМАТА.....                                                                                                | 80  |
| КУПРЕЙЧИК И.В.<br>ПРОГРАММНАЯ РЕАЛИЗАЦИЯ НАХОЖДЕНИЯ ОРТОГОНАЛЬНЫХ БАЗИСОВ В СОК.....                                                                                                                                              | 87  |
| БОЖИНСКИЙ И.А., РОМАНИЙ П.Г., ТКАЧЕНКО В.Ф.<br>К ВОПРОСУ ГИДРАВЛИЧЕСКОГО РАСЧЕТА ГАЗОРАСПРЕДЕЛИТЕЛЬНЫХ СЕТЕЙ.....                                                                                                                 | 91  |
| ПОЛОНСКИЙ А.Д.<br>ИНВАРИАНТНАЯ АЛГЕБРА РАНГОВЫХ ПРЕДИКАТОВ ДЛЯ СИНТЕЗА<br>КЛАССИФИКАТОРОВ АНАЛОГОВЫХ СИГНАЛОВ.....                                                                                                                | 99  |
| ГОЛОВАШИЧ С.А., ПОНОМАРЕВ Д.В.<br>ПОТОЧНЫЙ ШИФР VMGL И ОСОБЕННОСТИ ЕГО ПРОГРАММНОЙ РЕАЛИЗАЦИИ.....                                                                                                                                | 107 |
| АНПИЛГОВ Е.М., ДЗЮНДЗЮК В.В., СТЫЦЕНКО Т.Е.<br>ИНТЕРПРЕТАЦИЯ И АНАЛИЗ НАСЛЕДУЕМЫХ РАЗЛИЧНЫХ ФАКТОРОВ НА ОПЕРАЦИЯХ<br>ИЗГОТОВЛЕНИЯ СИСТЕМЫ.....                                                                                    | 113 |
| РЕФЕРАТИ.....                                                                                                                                                                                                                     | 116 |

$$B = A \cdot T_{ij}, C = T'_{ij} \cdot B = T'_{ij} \cdot A T_{ij} \quad (24)$$

вычисляются значения чисел  $c, s$ , а затем определяется матрица  $C$ . Процесс выбора на каждом шаге наибольшего внедиагонального элемента требует значительных затрат машинного времени, поэтому лучше использовать процесс с двустепенным выбором почти наибольшего внедиагонального элемента.

В этом процессе выбирается строка с наибольшей суммой квадратов внедиагональных элементов и затем в ней наибольший по модулю элемент, по которому выбирается очередное вращение. Выбранный таким образом элемент будет близок к наибольшему по модулю и будет не менее среднего квадратичного всех внедиагональных элементов.

Двустепенный выбор наибольшего из  $n$  и затем из  $n-1$  чисел значительно удобнее однократного выбора наибольшего из всех внедиагональных элементов. Тем самым достигается повышение быстродействия предложенного алгоритма в целом.

**Список литературы:** 1. Наталуха Ю.В. Метод сравнения в линейных динамических системах // Радиозлектроника и информатика. 1998. №1(02). С.61-63. 2. Бондаренко М.Ф., Шабанов-Кушнарченко Ю.П., Шляхов В.В. Предикаты  $n$ -мерной линейности и их свойства. 1982. 19с. Деп. в ВИНТИ, № 4764-82. 3. Рисс Ф., Секефальви-Надь Б. Лекции по функциональному анализу. М.: Мир, 1979. 587с. 4. Гельфанд И.М. Лекции по линейной алгебре. М.: Наука, 1966. 280 с. 5. Фадеев Д.К., Фадеева В.Н. Вычислительные методы линейной алгебры. М.-Л.: Физматгиз, 1963. 734 с. 6. Ахиезер Н.И., Глазман Н.М. Теория линейных операторов в гильбертовом пространстве. М.: Наука, 1966. 543 с.

Поступила в редколлегию 22.03.2003

**Наталуха Юрий Владимирович**, канд. техн. наук, доцент кафедры высшей математики ХНУРЭ. Научные интересы: алгебра, диагностика, управление, контроль и надежность в технических системах. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 702-13-72.

**Афанасьев Вадим Алексеевич**, канд. физ.-мат. наук, доцент кафедры высшей математики ХНУРЭ. Научные интересы: математические методы цифровой обработки сигналов. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 702-13-72.

**Климова Наталья Павловна**, канд. техн. наук, доцент кафедры высшей математики ХНУРЭ. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 702-13-72.

УДК 004.423.42

Е.Л. ЛЕЩИНСКАЯ

## ИДЕНТИФИКАЦИЯ СЕМАНТИЧЕСКИХ ОШИБОК ПРИ СИНТАКСИЧЕСКОМ АНАЛИЗЕ

Рассматриваются два метода идентификации семантических ошибок. Оба они призваны различными способами объединить этап обнаружения семантических ошибок и синтаксический анализ с максимальным уменьшением дополнительных затрат.

### 1. Актуальность исследования

Без ввода информации не обходится практически ни одна компьютерная система. А значит, необходимы надежные механизмы разбора введенных данных, обнаружения присутствующих в них ошибок и выдачи соответствующих информативных сообщений.

При этом возможны два случая (два типа) ошибок:

— ошибка синтаксическая (неправильно написана конструкция и, как результат, строка не выводится);

— ошибка семантическая (все конструкции в тексте правильные, строка выводится, но взаимное расположение конструкций, присутствие в данном контексте и т.д. недопустимы),

Примером первого типа может быть незакрытая фигурная скобка в программе, неправильно написанное ключевое слово, отсутствие точки с запятой и т.д. Сообщения о таких ошибках сформировать не очень сложно [1]. Они закладываются жестко в анализирующий механизм или (в простейшем случае) достаточно каждому нетерминалу сопоставить некоторый текстовый комментарий, отражающий его суть, а при обнаружении ошибки указать её позицию и комбинацию из сообщений, стоящих на границе этой позиции символов.

Как вариант примера второго типа ошибки интересно рассмотреть следующее: использование операторов `CONTINUE` и `BREAK` за пределами цикла, присвоение или выборка значения из необъявленной переменной, несоответствие типов переменных и т.д.

Основным механизмом анализа теста на сегодняшний момент является грамматика. В простейшем варианте она позволяет установить его корректность (отсутствие ошибок) и, в случае прохождения этого этапа, построить его промежуточную объектную модель (например, дерево разбора). И если выявление синтаксических ошибок не вызывает трудностей (достаточно лишь применять правила вывода до удачного завершения или до достижения ошибки), то семантические ошибки отслеживать труднее. Для этого *не хватает простых правил вывода грамматики*.

## 2. Постановка задачи. Цель исследования

Пусть существует задача разбора текста программы на формальном языке, в которой надо кроме синтаксических ошибок контролировать еще и семантические (не допускать использование операторов `break` и `continue` за пределами цикла, следить за применением в выражениях только объявленных ранее переменных, а также переменных только в их области видимости).

Язык допускает указания на:

- 1) заголовок и тело главной процедуры, содержащее собственно код программы;
- 2) следующие алгоритмические конструкции тела программы:
  - объявление переменных;
  - операции вычисления значения алгебраического выражения, присвоение значения переменной;
  - условный оператор ветвления;
  - циклический оператор `While` с возможностью использования безусловного оператора перехода на новую итерацию (`continue`) и завершения цикла (`break`);
  - безусловный оператор возврата из подпрограммы.

Построение грамматики, которая на этапе синтаксического анализа без особых ресурсных затрат сможет решить поставленную задачу, и является целью данного исследования.

## 3. Два метода решения поставленной задачи

### 3.1. Первый метод

Состоит в использовании атрибутивных грамматик для привязки к продукциям семантической информации. В атрибуты нетерминала закладываются сведения о контексте вывода, наличии или отсутствии определенных конструкций. Анализируя значения атрибутов в процессе вывода, можно сделать заключение о корректности разбираемого текста.

Каковы же возможности использования атрибутивных грамматик для идентификации семантических ошибок при анализе текста?

Возможности обычной атрибутивной грамматики ограничены [2], поэтому для их расширения следует внести несколько дополнений:

- продукция по-прежнему состоит из левой и правой частей;
- для описания продукции используется три блока:
  - 1) в первом указывается сама продукция (левая и правая часть);
  - 2) во втором – набор операций для формирования и вычисления атрибутов;

3) третий – блок действий, в котором анализируются значения атрибутов и выполняются операции, предусмотренные в ситуациях, наступление которых определяется совокупностью значений атрибутов. Управление передается блоку действий тогда, когда вычислены значения всех атрибутов в продукции, т.е., возможно, даже до завершения полного разбора;

– левая часть продукции – нетерминал, правая цепочка из терминалов и нетерминалов;

– каждый нетерминал может иметь неограниченное число атрибутов (свойств нетерминала, каждый из которых характеризуется набором признаков);

– количество атрибутов нетерминала не фиксировано жестко, а может изменяться динамически в зависимости от продукции или атрибутов других нетерминалов в ней;

– функции присвоения значений атрибутам, а следовательно, их наличие или отсутствие, задаются в продукции с помощью специальных операций копирования/вычисления из левой части продукции в правую или наоборот; таким образом можно передавать информацию в процессе вывода от одного нетерминала другому, сохраняя контекстную информацию (образно говоря, двигать атрибут по дереву вывода);

– операция может выполняться или не выполняться в зависимости от наложенного на неё условия. Здесь возможны два варианта:

- операция выполняется всегда (условия не указаны);
- операция выполняется при истинности определенного условия;
- каждый атрибут в продукции кодируется следующим набором признаков:

1) категория-имя атрибута (неуникальное в пределах нетерминала литеральное значение, несущее на себе информативную нагрузку о смысле атрибута, а также являющееся указателем на тип атрибута. Например, VAR – переменная, DVar – объявленная переменная);

2) идентификатор нетерминала, к которому привязан атрибут (уникальное в пределах продукции цифровое значение – номер по порядку нетерминала в продукции);

3) идентификатор атрибута (уникальное в пределах атрибутов нетерминала значение – например, номер атрибута в нетерминале);

4) тип данных атрибута (информативный признак);

5) функция вычисления значения атрибута (записанный на некотором встроенном скриптовом языке набор команд, после выполнения которых атрибут будет создан и ему присвоится определенное значение).

Пусть имеется продукция вида  $A_{a,a} \rightarrow B_a v C_a$ .

Здесь  $a$  – тип атрибута. Обратиться к первому атрибуту нетерминала  $A$  можно посредством нотации  $a(1,1)$ , ко второму –  $a(1,2)$ . Указать функцию присвоения им значений так:  $a(1,1) \leftarrow a(2,1) + a(3,1)$ . Тип  $a$  должен быть описан предварительно. Условия же применения правила создания и вычисления атрибута могут быть заданы следующим образом:

Общий вид:

Вариант первый : ? < Условие > : < Действие1 > [, < Действие2 > ]

Пояснения: Действие 1 выполняется при истинности Условия 1, а Действие 2 – при ложности.

Вариант второй : [ $? < \text{Условие} > : \text{Действие} >$ ]

Пояснения: Действие 1 выполняется при истинности Условия 1 или всегда при отсутствии Условия 1.

Действие N – условное обозначение операции или блока операций, заключенных в фигурные скобки.

Условие N – логическое выражение (комбинация одного или нескольких выражений). Принимает значение И или Л.

Пример для продукции:

Продукция:  $A_b \rightarrow BvCG_c$ .

Условие и правило:  $?b(1,1).Value <> c(4,1).Value : a(1,*) \leftarrow a(ind > 1,*)$

Здесь ключевое слово Value означает ссылку на значение атрибута (независимо от того, сборный его тип или простой). Помимо этого возможно обращение к отдельным полям сборного типа атрибута, если это необходимо по их имени. Например,  $c(1,1).Weight$ .

\* Означает допустимость любого значения и позволяет описывать условия проверки и действия не на один атрибут, а на группу атрибутов одного типа. В данном случае запись означает “скопировать атрибуты “типа а” всех нетерминалов правой части продукции в левую”.

ind – имя переменной-индекса, принимающей все возможные в своем диапазоне значения (в данном случае от 1 до 4, т.е. индексы всех нетерминалов продукции).

Динамические атрибуты, присутствующие в правой части, не переходят в левую (синтезируемые атрибуты) или из левой – в правую (наследуемые атрибуты), т.е. попросту не двигаются по дереву разбора, если не указано соответствующего действия или не выполнено условие преобразования.

Каждая продукция содержит принадлежащий ей блок действий, которые должны быть осуществлены в случае выполнения соответствующих им условий. Синтаксис описания каждой операции в блоке действий аналогичен синтаксису при вычислении атрибутов. В данной задаче этот блок упрощается благодаря единственному типу действия – выдаче информативного сообщения разработчику грамматики или оператору, вводящему текст. Например,

Продукция  $A_b \rightarrow BvCG_c$ .

Блок действий:  $?b(1,1):MESS \text{ “Значение атрибута } b \text{ при нетерминале } A \text{ равно ”} + b(1,1)$ .

Условие  $b(1,1)$  означает присутствие атрибута типа b у левого нетерминала продукции.

**Алгоритм работы грамматики** с динамическими атрибутами практически ничем не отличается от алгоритма функционирования обычной атрибутной грамматики, за исключением того, что количество атрибутов при каждом нетерминале не указано жестко, известно лишь то, что они могут быть определенных типов или отсутствовать вообще. Поэтому разработчику необходимо проверять, существует ли желаемый атрибут, прежде чем его использовать. Поскольку в грамматике все атрибуты синтезируемые, то дерево разбора строится перед их вычислением (нисходящий метод разбора). После построения дерева начинается его обход и вычисление атрибутов по правилам, предусмотренным в каждой продукции. После вычисления всех атрибутов в продукции выполнение переходит в “блок действий по условию”. После возврата из него анализ продолжается, если иного не было предусмотрено в блоке действий.

Теперь, когда все термины и обозначения введены, можно привести вариант решения поставленной в начале статьи задачи с использованием динамической атрибутивной грамматики.

Атрибуты каждой продукции будут приведены позже.

// Заголовок и тело главной процедуры

1. Code  $\rightarrow$  Type main(Pl){Pr}

2. Type  $\rightarrow$  int | void

3. Pl  $\rightarrow$  TypeVarPlr |  $\varepsilon$

4. Plr  $\rightarrow$ , TypeVarPlr |  $\varepsilon$

5.  $\text{Var} \rightarrow \hat{a} | \hat{b} | \hat{c}$

6. Pr  $\rightarrow \varepsilon$  | Block Pr

7. Block  $\rightarrow$  Op

// Алгебраическое выражение, оператор возврата, вызов процедуры, продукции для формирования AV и Proc опущены

Op  $\rightarrow$  AV; | return; | Proc; |

8. continue; | break

// Описание переменной

9. Block  $\rightarrow$  TypeVar;

// Ветвление

10. Block  $\rightarrow$  if(Cond)BodyENDIF

11. Cond  $\rightarrow$  AVMarkAV

// AV – арифметическое выражение

12. Mark  $\rightarrow$  > | < | = | <=

// знак сравнения

13. ENDIF  $\rightarrow$  elseBody |  $\varepsilon$

14. Body  $\rightarrow$  {Pr} | Op

// Цикл

15. Block  $\rightarrow$  WHILE(Cond)LBody

16. LBody  $\rightarrow$ ; | Body

17....

Ниже приведены правила той же грамматики, только с указанием атрибутов и функций их вычисления.

1. Code  $\rightarrow$  Type main(Pl){Pr}

?DVAR(5>0) <> VAR(8,j>0): VAR(1,\*)-VAR(8,j)

// Из списка используемых переменных убираются формальные параметры.

?VAR(1,\*): MESS “Неизвестная

переменная” + VAP(1,\*)

?NC(8,j>1): MESS “Использование оператора” + NC(8,j).Op + “вне цикла”

Type  $\rightarrow$  int

VART(1,1)  $\leftarrow$  “int”

Type  $\rightarrow$  void

VART(1,1)  $\leftarrow$  “void”

2. Pl  $\rightarrow$  Type Var Plr

DVAR(1,\*)  $\leftarrow$  DVAR(4,\*)

DVAR(1,\*)  $\leftarrow$  (VARN(3,1), VART(2,1))

Pl  $\rightarrow \varepsilon$

3. Plr  $\rightarrow$ , TypeVarPlr

DVAR(1,\*)  $\leftarrow$  DVAR(5,\*)

DVAR(1,\*)  $\leftarrow$  (VARN(4,1), VART(3,1))

Plr  $\rightarrow \varepsilon$

4. Var  $\rightarrow$  a | b | c

VARN(1,1)  $\leftarrow$  “a” | “b” | “c”

5. Pr  $\rightarrow$  Block Pr

DVAR(1,\*) = DVAR(2,\*)

?DVAR(1,i>0) <> VAR(j>1,k>0):

VAR(1,\*) = VAR(j,k)

NC(1,\*)  $\leftarrow$  NC(i>1,\*)

Pr  $\rightarrow \varepsilon$

// Алгебраическое выражение, оператор возврата, вызов процедуры, продукции для формирования AV и Proc опущены

6. Block  $\rightarrow$  Op

VAR(1,\*)  $\leftarrow$  VAR(2,\*)

7. Op  $\rightarrow$  AV; | return; | Proc;

VAR(1,\*)  $\leftarrow$  VAR(2,\*)

Op  $\rightarrow$  continue; | break;

NC(1,1) = (True, “continue”/”break”)

// Описание переменной

8. Block  $\rightarrow$  Type Var;

DVAR(1,1)  $\leftarrow$  (VARN(3,1), VART(2,1))

// Ветвление

9. Block  $\rightarrow$  if(Cond) Body ENDIF

NC(1,\*)  $\leftarrow$  NC(i>1,\*)

?VAR(6,i>1) <> DVAR(6,j>1):

VAR(1,\*)  $\leftarrow$  VAR(6,i)

VAR(1,\*)  $\leftarrow$  VAR(4,\*)

VAR(1,\*)  $\leftarrow$  VAR(7,\*)

10. Cond  $\rightarrow$  AV Mark AV

// AV – арифметическое выражение

```

VAR(1,*) ← VAR(2,*)
VAR(1,*) ← VAR(4,*)
11. Mark -> >|<|<=|>=
// знак сравнения
12. ENDIF → else Body
NC(1,*) ← NC(2,*)
? VAR(2,i>1) <> DVAR(2,j>1):
VAR(1,*) ← VAR(2,i)
ENDIF → ε
13. Body → {Pr}
NC(1,*) ← NC(3,*)
VAR(1,*) ← VAR(3,*)
Body → Op
NC(1,*) ← NC(2,*)
VAR(1,*) ← VAR(2,*)
//Цикл
14. Block → WHILE(Cond)LBody

```

VAR(1,1) ← VAR(i>1,\*)  
Копирование атрибута NC не происходит, так как цикл найден.

```

15. LBody -> Body
?VAR(2,i>1) <> DVAR(2,j>1):
VAR(1,*) ← VAR(2,i)

```

LBody -> ;

Значения имен атрибутов:

NC – признак необходимости цикла  
VAR – переменная была использована  
DVAR – переменная была объявлена  
VARN – имя переменной  
VART – тип переменной

Употребление \* указывает на то, что условие проверяется для каждого экземпляра атрибута, удовлетворяющего маске. При его выполнении производится операция копирования атрибута. Таким образом достигается его движение по дереву вывода до тех пор, пока не будет применен блок сообщений по условию. Ниже приводится пример кода и соответствующее ему дерево разбора (рис.1).

На рис. 1 заштрихованные точками вершины обозначают путь движения атрибута VAR(1,1)="s", до вершины дерева на его пути не встретилось соответствующего атрибута DVAR(1,1)="s". Поэтому после выполнения блока действий будет выдано сообщение об ошибке.

Серым цветом закрашены вершины, обозначающие путь движения атрибута NC(1,1)=(True,"continue"). Вершина Block (родительская для LBody) сформирована благодаря срабатыванию продукции 16, которая представляет собой заголовок цикла. Поэтому в данной вершине атрибут NC уже отсутствует. Если бы на пути NC не встретился заголовок цикла, то он бы проследовал по дереву разбора вверх до корня, где при проверке также было бы выдано сообщение об ошибке.



Для поставленной в начале статьи задачи в терминах такой системы может быть предложено следующее решение.

На этапе синтаксического анализа предлагается использовать грамматику, рассмотренную в первом методе (без динамических атрибутов). В качестве постоянных атрибутов при всех вариантах блоков кода (Op,if,while) вводятся следующие свойства:

*Level* – уровень вложенности внутри блока (уникальное числовое значение, определяющее, в какой области видимости внутри функции находится анализируемый код; уровень вложенности изменяется внутри фигурных скобок в теле цикла или операторе ветвления). Его значение формируется случайным образом и для него сохраняется информация о вложенности в другие уровни (также посредством предикатов);

*LN* – номер строки, в которой находится анализируемый код.

Для контроля применения только объявленных переменных могут быть предложены следующие факты:

*VAR(X, LN, Level)* – использована переменная с именем *X* в строке *LN*, уровень вложенности которой *level*. Этот факт добавляется в рабочую БЗ при срабатывании правила 8 грамматики. Значения *LN*, *Level* извлекаются из одноименных атрибутов соответствующих нетерминалов правила.

*DVAR(X, LN, Level)* – объявлена переменная. Факт добавляется при срабатывании правил 3,4,9 грамматики.

Для контроля использования операторов *break*, *continue* только в пределах цикла можно предложить следующие факты:

*Op(name, LN, Level)* – использован оператор *name* на уровне вложенности *level*. Добавляется при срабатывании правила 8.

*Lop(level)* – использован цикл, породивший уровень вложенности *level*. Добавляется при срабатывании правила 15.

*InBlock(Level1, level2)* – блок *Level2* является вложенным в *Level1* (такой непосредственный факт добавляется при срабатывании правил, создающих новый уровень вложенности *Body* → {Pr} и *Code* → Type main(Pl){Pr}).

База правил при этом должна содержать следующие правила (запись хорновских дизъюнктов ведется в прологовской нотации).

Правила общего назначения:

*InBlock(L1, L2)* : - *L1* = *L2*.

*InBlock(L1, L2)* : - *InBlock(L1, L3), InBlock(L3, L2)*.

Для контроля использования только объявленных переменных:

*VAR(X, LN1, Level1)* : - *DVAR(X, LN2, Level2), LN2* < *LN1, InBlock(Level2, Level1)*.

Для контроля использования операторов *break* и *continue* только в пределах циклов:

*Op(continue, Level1)* : - *Lop(Level2), InBlock(Level2, Level1)*.

*Op(break, Level1)* : - *Lop(Level2), InBlock(Level2, Level1)*.

При анализе данным механизмом примера, описанного в первом методе, будет сформирована следующая рабочая БЗ:

*DVAR(v, 2, 1)*

*DVAR(sum, 3, 1)*

*VAR(v, 4, 1)*

*VAR(sum, 5, 1)*

*VAR(v, 6, 1)*

*Lop(2)*

```

InBlock(1,2)
DVAR(tmp,7,2)
VAR(tmp,8,2)
VAR(v,8,2)
VAR(v,9,2)
VAR(tmp,10,2)
VAR(sum,11,2)
VAR(tmp,11,2)
Op(continue,13,2)
VAR(s,15,1)

```

Рассмотреть можно только факты, добавленные при логическом анализе в рабочую базу из базы правил для переменной *s* и оператора *continue*.

```
→DVAR(v, LN2, Level2), LN2 < 4, InBlock(Level2, 1)
```

```
→Lop(Level2), InBlock(Level2, 2)
```

При унификации для цели 17 будет получен пустой дизъюнкт при унификации с фактом 1, а для цели 18 – с фактом 6.

#### 4. Результаты и сравнение с аналогами

Попытка решить поставленную задачу с использованием простой неатрибутивной грамматики привела к следующему результату.

Грамматика для разбора текста программ на формальном языке:

// Заголовок и тело главной процедуры

1. Code → Type main(Pl){Pr}

2. Type → int | void

3. Pl → TypeVarPlr | ε

4. Plr → ,TypeVarPlr | ε

5. Var → a | b | c

6. Pr → ε | Block Pr

// Алгебраическое выражение, оператор возврата, вызов процедуры, продукции для формирования AV и Proc опущены

7. Block → Op

8. Op → AV; | return; | Proc;

// Описание переменной

9. Block → TypeVar;

// Ветвление

10. Block → if(Cond)BodyENDIF

11. Cond → AVMarkAV // AV – арифметическое выражение

12. Mark → > | < | = | <= // знак сравнения

13. ENDIF → elseBody | ε

14. Body → {Pr} | Op

// Цикл

15. Block → WHILE(Cond)LBody

16. LBody → ; | {LPr} | Op | continue; | break;

17. LPr → LBlock LPr | ε

18. LBlock → Block | continue; | break;

19....

В данной грамматике возможность использования операторов `break` и `continue` только в теле цикла контролируется с помощью его правил вывода, которые для реализации такой функциональности частично дублируют аналогичные правила для тела программы с добавлением возможности использования `break` и `continue` (см. правила 16-18). Это делает грамматику запутанной и сложно анализируемой для составителя и пользователя, что может привести к трудно обнаружимым ошибкам. Возможность контролировать использование в операторах вычисления значений только объявленных переменных на этапе синтаксического анализа отсутствует в данной грамматике вообще, и должна быть заложена в последующие этапы анализа и преобразования кода.

Приведенный пример убеждает в том, что выявить семантические ошибки на этапе синтаксического анализа, используя только стандартные средства синтаксического разбора, невозможно.

Грамматика, приведенная в первом из предлагаемых в статье методов, проще и короче, чем простая неатрибутная, так как нет необходимости дублировать одни и те же правила для изменения возможностей вывода внутри цикла (правило 16 упрощено, а 17-18 отсутствуют). При этом и в первом, и во втором методе решения достигается требуемая функциональность.

## 5. Выводы

Затруднительно однозначно ответить на вопрос, какой из двух описанных в статье методов эффективнее в использовании. Первый метод однозначен в своей работе и не требует дополнительных затрат при выборе очередного шага функционирования, но зато требует дополнительной памяти для хранения дублирующих значений атрибутов при каждой из вершин по пути в дереве разбора. Второй вариант более компактный, но, как ожидается, будет затрачивать на анализ больше времени.

При описании задачи в терминах каждого из методов второй оказался легче формализуемым. Это произошло потому, что, несмотря на идентичность отслеживаемых фактов (смысловая нагрузка атрибутов грамматики и фактов БЗ), при составлении атрибутивной грамматики аналитику необходимо самому следить за тем, какие атрибуты могут появиться в правой (левой) части правила и какие из них должны переходить в левую (правую). Таким образом, при составлении продукции он вынужден держать в голове все дерево разбора от корня до листьев, в то время как во втором варианте необходимо лишь следить за добавлением соответствующих фактов в БЗ при срабатывании продукции. База правил в этом случае содержит всего несколько логических зависимостей (в рассматриваемом примере их количество равно 5), позволяющих отследить правомерность появления того или иного факта в рабочей БЗ. Зато в первом варианте все необходимые на данном этапе анализа факты присутствуют в явном виде в продукции, не требуя никакого вывода.

Поэтому выбор решения в каждом конкретном случае зависит от поставленных целей и желаемых характеристик разрабатываемой системы.

**Список литературы:** 1. Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов. М.: Мир, 1997. 654 с. 2. Josef Grosch Cocktail Toolbox for Compiler Construction. Achern, Germany: CoCoLab - Datenverarbeitung. 1994. 40 с. // [www.cocolab.com](http://www.cocolab.com) 3. А. Тэйз, П. Грибомонт, Дж. Льюис Логический подход к искусственному интеллекту М.: Мир, 1990. 429 с.

Поступила в редколлегию 21.03.2003

Лещинская Елена Леонидовна, студентка 5-го курса специальности ПОАС, факультет компьютерных наук ХНУРЭ. Научные интересы: разработка трансляторов, САПР, системы ИИ, технологии БД и БЗ. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 702-16-46; 24-39-25; e-mail: [leo@starnet.com.ua](mailto:leo@starnet.com.ua), [leshinskayahelen@yahoo.com](mailto:leshinskayahelen@yahoo.com)

Для преподавателей университетов,  
научных работников, аспирантов

## Тематика:

- **Компьютерная инженерия**
- **Математическое моделирование**
- **Оптимизация и процессы управления**
- **Автоматизация проектирования и диагностика**
- **Информационные интеллектуальные системы**
- **Проектирование интегральных схем и микросистем**
- **Компьютерные технологии в образовании**

Адрес редакции: 61166, Харьков, просп. Ленина, 14,  
ХНУРЭ, тел. 40-93-26, e-mail: [ri@kture.kharkov.ua](mailto:ri@kture.kharkov.ua)