МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
"Ижевский государственный технический университет имени М.Т.Калашникова"
(ФГБОУ ВПО «ИжГТУ имени М.Т.Калашникова»)

Кучуганов В.Н., Касимов Д.Р.

МАТЕМАТИЧЕСКАЯ ЛИНГВИСТИКА
МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №2
«РАЗРАБОТКА СИНТАКСИЧЕСКОГО АНАЛИЗАТОРА»

Рекомендовано учебно-методическим советом ФГБОУ ВПО «ИжГТУ имени М.Т. Калашникова» для использования в учебном процессе в качестве
элемента ЭУМКД для студентов обучающихся по направлению 230100.62 «Информатика и вычислительная техника», профилям «Автоматизированные системы обработки информации и управления», «Системы автоматизированного проектирования» при изучении дисциплин «Математическая лингвистика», «Лингвистическое обеспечение САПР»

Ижевск 2013

Составители: Кучуганов Валерий Никонорович, доктор технических наук, профессор
Касимов Денис Рашидович, ассистент

УДК 681.3

Математическая лингвистика: методические указания к выполнению лабораторной работы №2 «Разработка синтаксического анализатора» по курсам «Математическая лингвистика», «Лингвистическое обеспечение САПР» профилей «Автоматизированные системы обработки информации и управления», «Системы автоматизированного проектирования» направления 230100.62 «Информатика и вычислительная техника».
Составители: Кучуганов В.Н., Касимов Д.Р., Ижевский государственный технический университет имени М.Т. Калашникова. Ижевск, 2013. – 12 с.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Синтаксический анализ – это процесс, который определяет, принадлежит ли некоторая последовательность лексем языку, порождаемому грамматикой. В принципе, по любой грамматике можно построить синтаксический анализатор, но грамматики, используемые на практике, имеют специальную форму. Например, известно, что для любой контекстно-свободной грамматики может быть построен анализатор, сложность которого не превышает $O(n3)$ для входной строки длины n, но в большинстве случаев по заданному языку программирования мы можем построить такую грамматику, которая позволит сконструировать и более быстрый анализатор. Анализаторы реально используемых языков обычно имеют линейную сложность; это достигается, например, за счет просмотра исходной программы слева направо с заглядыванием вперед на один терминальный символ (лексический класс).

Вход синтаксического анализатора – последовательность лексических классов и таблицы, например, таблица внешних представлений, которые являются выходом лексического анализатора.

Выход синтаксического анализатора – дерево разбора и таблицы, например, таблица идентификаторов и таблица типов, которые являются входом для следующего просмотра лингвистического процессора (например, это может быть просмотр, осуществляющий контроль типов).

Совсем необязательно, чтобы фазы лексического и синтаксического анализа выделялись в отдельные просмотры. Обычно эти фазы взаимодействуют друг с другом на одном просмотре. Основной фазой такого просмотра считается фаза синтаксического анализа, при этом синтаксический анализатор обращается к лексическому анализатору каждый раз, когда у него появляется потребность в очередном терминальном символе.

Одним из наиболее простых и потому одним из наиболее популярных методов нисходящего синтаксического анализа является метод рекурсивного спуска.

Цель описанной ниже лабораторной работы – ознакомиться с теоретическими и практическими основами построения блока синтаксического анализа лингвистического процессора.

1. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ №2

Тема работы: «Разработка синтаксического анализатора»

Цель работы: ознакомиться с теоретическими и практическими основами построения блока синтаксического анализа лингвистического процессора.

Используемые программные средства: система программирования Delphi 7.0 или выше; графический редактор Microsoft Visio.

Задание по лабораторной работе заключается в разработке синтаксического анализатора для предложенного варианта контекстно-свободной грамматики (КС-грамматики) методом предсказывающего рекурсивного спуска.

Содержание отчета:

1) титульный лист;

2) текст задания, включающий вариант задания;

3) преобразование КС–грамматики по исключению левой рекурсии;

4) исходный текст синтаксического анализатора;

5) результаты тестирования.

Методические рекомендации к лабораторной работе

1. Лексический анализатор из лабораторной работы №1 должен быть расширен обработкой появившихся в КС-грамматике новых слов и включен в виде подпрограммы или поля класса или метода класса в синтаксический анализатор.

2. Если грамматика леворекурсивная, то устранить левую рекурсию.

3. Оформить синтаксический анализатор в виде процедуры или функции или класса, которые при обращении обрабатывают весь исходный текст.

Варианты индивидуальных заданий

В таблице 1.1 представлены варианты заданий.

Таблица 1.1. Варианты заданий

№	Грамматика	№	Грамматика	№	Грамматика
1	S → S S + S → A	2	S → A | B A → A B + | B	3	O → A = A | (L) A → <1> | <2>

	A → S S * A → B B → <2> | <1> Внимание! Не является грамматикой класса LL(1).		B → B C * | C C → C - | <1> | <2>		L → true| false | U | not L | O U → <2>
4	П → П , O | O O → <2> = <1> <1> O → <2> = (O)	5	C → B | P B → [P P] | <1> P → (B B) | <2>	6	M → M + П | П П → П * A | A A → [ ] A → [S] S → <1>, S | <2> , S | <1> | <2> | [A]
7	П → П ; K | K K → O R , A O → <2> R → <1> A → <1> | <2>	8	D → D ∨ K | K K → K ∧ A | A A → ¬ A | (D) | O O → <1> = <2> | <2> = <1> | <2> = <2>	9	S → A | B A → <1> | ((A) [B]) B → <2> | [[B] (A)]
10	S → <2> S A S → A A → (S) A → ( ) A → <1>	11	S → S ; P | P P → A = A A → A + M | M M → M * T | T T → <1> | <2> | (A)	12	S → (D !: S ! S) S → <1> D → D ∨ K | K K → K ∧ A | A A → <2> | ¬ A | (D)
13	S → <1> A S S → <2> A → <1> A → <2> S A	14	S → B A A → + B A | B B → D C C → * D C | D D → (S) | <1> | <2>	15	S → <2> A <2> B S → <1> A <1> B A → <2> | <2> <1> B → <2> B B → <2>
16	S → A : B . A → <2> B → B ; C B → C C → C , D C → D D → <1> D → <2>	17	S → A ; S S → A A → A , B A → B B → <1> B → <2> : <1>	18	S → (A !: B ! B) S → (A !: B) A → <2> B → <1> B → S
19	S → + S S S → A A → * S S A → B B → <2> [<1>]	20	S → S + A S → A A → A * B A → B B → <2> (<1>)	21	S → S ; A S → A A → <2> := [B] B → <1> , B B → <1>
22	S → (A) A → (<2> B) A → <1> B → (C) C → C , <1> C → <1>	23	S → A :– B . A → <2> (<1>) B → B , A B → A	24	S → <2> A <1> B A → [A] A → <1> B → B <2> B → <2>
25	S → S A S → B B	26	S → <2> (A) :- B . A → <1> , A	27	S → [S] | A A → [<2> B]

	A → <2> A A A → <1> B → <2>		A → <1> B → B , <2> B → <2>		B → [C] C → C , <1> | <1>

Примечание. Через <1> и <2> обозначены слова из лабораторной работы №1.

Внимание! Символы <, > частью слов не являются. Для лучшего рассмотрения содержащихся в грамматике символов рекомендуется изучить свой вариант задания в увеличенном масштабе.

Контрольные вопросы:

1. Классификация языков, грамматик, автоматов по Хомскому.
2. Контекстно-свободные грамматики.
3. Синтаксические деревья.
4. Нисходящие алгоритмы синтаксического анализа.
5. Восходящие алгоритмы синтаксического анализа.
6. Метод рекурсивного спуска.
7. Документирование КС - синтаксиса. Форма Бэкуса-Наура.
8. Генераторы синтаксических анализаторов.

2. КОНТЕКСТНО-СВОБОДНЫЕ ГРАММАТИКИ

Контекстно-свободная грамматика (КС-грамматика, бесконтекстная грамматика) – частный случай формальной грамматики (тип 2 по иерархии Хомского), у которой левые части всех продукций являются одиночными нетерминалами. Смысл термина «контекстно-свободная» заключается в том, что возможность применить продукцию к нетерминалу, в отличие от общего случая неограниченной грамматики Хомского, не зависит от контекста этого нетерминала.

Язык, который может быть задан КС-грамматикой, называется контекстно-свободным языком или КС-языком.

КС-грамматики находят большое применение. Ими задаётся грамматическая структура большинства языков программирования, структурированных данных и т.д.

3. УСТРАНЕНИЕ ЛЕВОЙ РЕКУРСИИ ИЗ ГРАММАТИКИ

Грамматика является леворекурсивной, если в ней имеется нетерминал A, такой, что существует порождение $A \overset{+}{\Rightarrow} A\alpha$ для некоторой строки α. Методы нисходящего разбора, к

которым относится метод рекурсивного спуска, не в состоянии работать с леворекурсивными грамматиками, поэтому требуется преобразование грамматики, которое устранило бы из нее левую рекурсию.

Устранение непосредственной левой рекурсии (продукции вида $A \to A\alpha$) может быть осуществлено с помощью следующей технологии. Вначале группируются A-продукции:

$A \to A\alpha_1 | A\alpha_2 | \ldots | A\alpha_m | \beta_1 | \beta_2 | \ldots | \beta_n$

где β_i не начинаются с A. Затем эти A-продукции заменяются на

$A \to \beta_1 A' | \beta_2 A' | \ldots | \beta_n A'$

$A' \to \alpha_1 A' | \alpha_2 A' | \ldots | \alpha_m A' | \varepsilon$

где A' – новый нетерминальный символ.

Нетерминал A порождает те же строки, что и ранее, но без левой рекурсии. Эта процедура устраняет все непосредственные левые рекурсии из продукций для A и A' (при условии, что ни одна строка α_i не является ε), но не устраняет левую рекурсию, вызванную двумя или более шагами порождения.

Алгоритм, приведенный ниже, удаляет из грамматики левую рекурсию. Он гарантированно работает с грамматиками, не имеющими циклов (порождений типа $A \overset{+}{\Rightarrow} A$) и ε-продукций (продукций типа $A \to \varepsilon$). Из грамматики могут быть также удалены и циклы, и ε-продукции.

Алгоритм. Устранение левой рекурсии

Вход. Грамматика G без циклов и ε-продукций.

Выход. Эквивалентная грамматика без левой рекурсии.

Метод. Применить алгоритм, приведенный ниже. Обратите внимание, что результирующая грамматика без левых рекурсий может иметь ε-продукции.

1. Расположить нетерминалы в некотором порядке $A_1, A_2, \ldots, A_n$.

2. for i := 1 to n do begin

    for j := 1 to i–1 do begin

        Заменить каждую продукцию вида $A_i \to A_j \gamma$ продукциями $A_i \to \delta_1 \gamma \mid \delta_2 \gamma \mid \ldots \mid \delta_k \gamma$, где $A_j \to \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$ – все текущие A_j-продукции

    end

    Устранить непосредственную левую рекурсию среди A_i-продукций

end

4. МЕТОД РЕКУРСИВНОГО СПУСКА

Метод рекурсивного спуска – алгоритм синтаксического анализа, реализуемый путем взаимного вызова процедур разбора, соответствующих правилам контекстно-свободной грамматики. Применения правил последовательно, слева-направо поглощают токены, полученные от лексического анализатора. Это один из самых простых алгоритмов парсинга, подходящий для полностью ручной реализации.

Варианты реализации:

1. Предсказывающий парсер. Для парсеров этого типа нужна подходящая КС-грамматика, конкретно LL(k) грамматика, позволяющая по очередному токену или токенам однозначно выбрать (предсказать) один из альтернативных вариантов раскрытия каждого нетерминала. Такой парсер работает за линейное время. Вариантом является LL-парсер — реализация предсказывающего парсера с автоматическим построением «таблицы предсказания», определяющей по заданному нетерминалу и очередному токену подходящее правило для раскрытия нетерминала.

2. Парсер с возвратом. Вместо предсказания парсер просто пытается применить все альтернативные варианты правил по порядку, пока одна из попыток не увенчается успехом. Такой парсер может потребовать экспоненциального времени работы, и не всегда гарантирует завершение, в зависимости от грамматики.

Пример анализатора по методу предсказывающего рекурсивного спуска

Рассмотрим грамматику:

$S \rightarrow if\ E\ then\ S\ else\ S$

$S \rightarrow begin\ S\ L$

$S \rightarrow print\ E$

$L \rightarrow end$

$L \rightarrow ;\ S\ L$

$E \rightarrow num = num$

Напишем анализатор языка, порождаемого этой грамматикой, методом рекурсивного спуска. Для этого нам придется описать по одной процедуре для каждого нетерминала грамматики:

```
class SimpleParser {
   /* Лексические классы, т.е. терминалы */
   const int IF = 1;
   const int THEN = 2;
   const int ELSE = 3;
   const int BEGIN = 4;
   const int END = 5;
   const int PRINT = 6;
   const int SEMICOLON = 7;
```

```
   const int NUM = 8;
   const int EQ = 9;

   public static void nextStep(int lc)
   {
      if (lexical_class == lc)
         lexical_class = getLC();
      else
         error();
   }

   public static void S(void)
   {
      switch(getLC())
      {
         case IF:
            E(); nextStep(THEN); S(); nextStep(ELSE); S(); break;
         case BEGIN:
            S(); L(); break;
         case PRINT:
            E(); break;
         default:
            error(); break;
      }
   }

   public static void L(void)
   {
      switch (lexical_class)
      {
         case END:
            getLC(); break;
         case SEMICOLON:
            getLC(); S(); L(); break;
         default:
            error(); break;
      }
   }

   public static void E(void)
   {
      nextStep(NUM); nextStep(EQ); nextStep(NUM);
   }

   public static void main(void)
   {
      lexical_class = getLC();
      S();
   }
} // end of SimpleParser
```

5. ПРИМЕР ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В данном разделе приводятся результаты выполнения лабораторной работы при следующей КС-грамматике:

$S \rightarrow A :- B .$

$A \rightarrow <2> (<1>)$

B → B , A

B → A

После устранения левой рекурсии грамматика выглядит следующим образом:

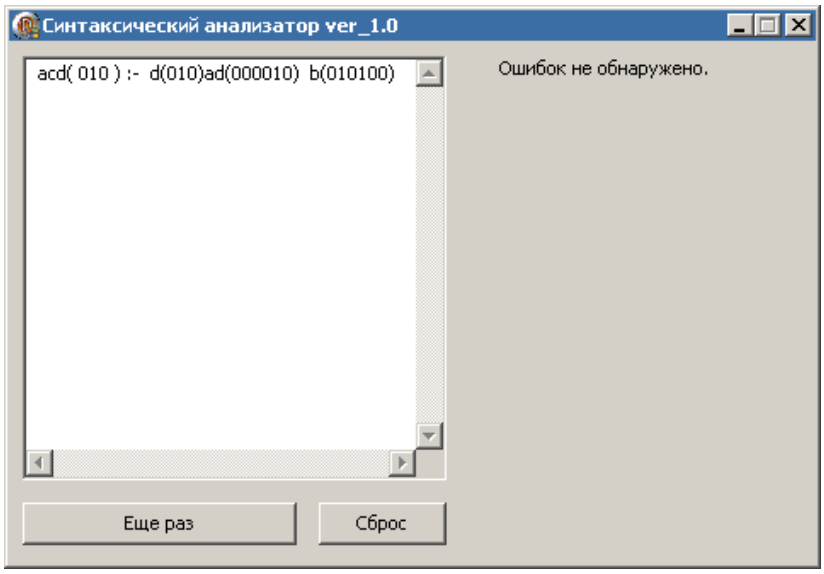
S → A :– B .

A → <2> (<1>)

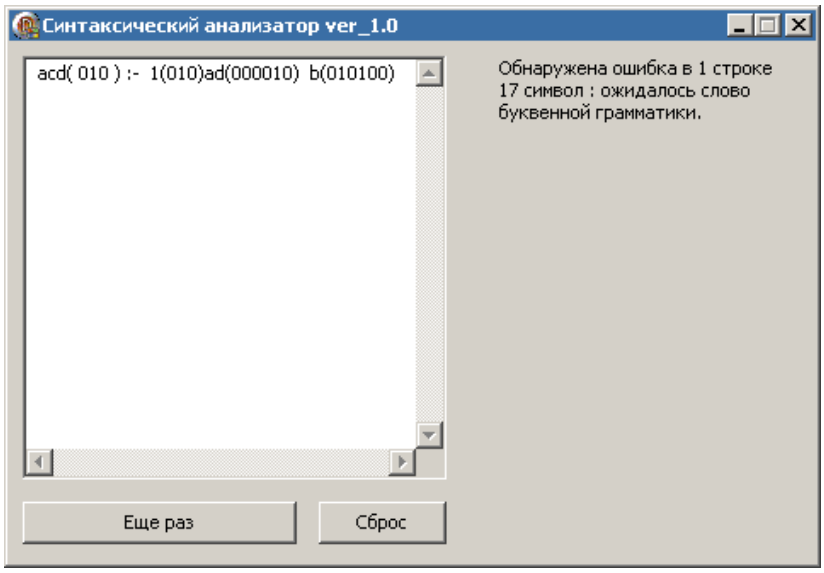
B → A B’

B’ → , A | ε

На рисунке 5.1 приведены примеры работы программы.



а)



б)

Рисунок 5.1. Примеры работы программы: а) при правильном тексте; б) при неправильном тексте

ЛИТЕРАТУРА

1. Дж. Хопкрофт, Р. Мотвани, Дж. Ульман. Введение в теорию автоматов, языков и вычислений. – М.: «Вильямс», 2002. – 528 с.
2. И.Г. Кревский, М.Н. Селиверстов, К.В. Григорьева. Формальные языки, грамматики и основы построения трансляторов. Учебное пособие (под ред. д.т.н., профессора А.М. Бершадского). – Пенза, 2003.
3. Свердлов С. З. Языки программирования и методы трансляции: Учебное пособие. — СПб.: Питер, 2007. — 638 с: ил.
4. Льюис Ф., Розенкранц Д., Стирнз Р. «Теоретические основы проектирования компиляторов». – М.: Мир, 1979.
5. Компаниец Р. И., Маньяков Е. В., Филатов Н. Е., «Системное программирование. Основы построения трансляторов». – СПб.: КОРОНА принт, 2000.
6. Мозговой М. В. «Классика программирования: алгоритмы, языки, автоматы, компиляторы. Практический подход». – СПб.: Наука и Техника, 2006.