**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФГБОУ ВО  
«БРЯНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

**Кафедра «Информатика и программное обеспечение»**

**Расчетно-графическая работа**

**по дисциплине «Теория автоматов и Формальных языков»**

направление подготовки 02.03.03 «Математическое обеспечение и администрирование информационных систем»

на тему:

**Основы разработки компилятора**

Выполнил студ. гр. О-19-МОА-тп-Б:

**Байдекин С.М.**

Проверил:

к.т.н., доц. **Трубаков Е. О.**

Брянск 2021

Содержание

[1. Введение 3](#_Toc89359263)

[2. Текстовое описание языка 4](#_Toc89359264)

[3. Описание языка на основе регулярных выражений 5](#_Toc89359265)

[4. Контекстно-свободная грамматика 6](#_Toc89359266)

[5. Автоматная грамматика 8](#_Toc89359267)

[6. Недетерминированный автомат 10](#_Toc89359268)

[7. Детерминированный автомат 12](#_Toc89359269)

[8. Минимальный детерминированный автомат 14](#_Toc89359270)

[9. Лексический анализатор 16](#_Toc89359271)

[10. Таблица компиляции 17](#_Toc89359272)

[11. Дерево разбора 19](#_Toc89359273)

[12. Генерация кода 21](#_Toc89359274)

[13. Оптимизация кода 22](#_Toc89359275)

[14. Список литературы 23](#_Toc89359276)

# Введение

Теория автоматов – это специальный раздел дискретной математики, изучающий математические модели обработки информации, так же называемые автоматами или по другому вычислительные машины.

Формальный язык – знаковая система, построена на строгих правилах грамматики и синтаксиса, предполагают однозначную запись сообщения (формулы, языки программирования и т.д.).

Цель данной работы заключается в том, чтобы научиться разрабатывать язык, а также получить понимание о работе компилятора.

Ход разработки языка выполняется по плану:

1. текстовое описание языка;
2. формальное описание языка;
3. построение грамматики;
4. создание автомата.

Этапы работы компилятора:

1. лексический анализатор;
2. построение таблиц компилятора;
3. синтаксический анализ;
4. генерация и оптимизация кода на ассемблере.

# Текстовое описание языка

Входной язык содержит арифметическое выражение разделенное символом – точка с запятой (;). Арифметические выражения состоят из идентификатора, чисел в виде 0 и 1, знак присваивания (=) и знаков операций (+, -, \*, /),а так же специальных операций такие как (XOR,OR,NOT,AND).

1. XOR – «Исключающее ИЛИ », это операция которая истинна, тогда и только тогда, когда один из аргументов истинен, а другой ложен.
2. OR – « ИЛИ », это операция которая истинна, когда один из 2 аргументов истинен и ложен, если оба аргумента ложны.
3. NOT – « НЕ », это операция которая возвращает обратное значение аргумента, если аргумент истинен, то будет ложен и наоборот.
4. AND – « И », это операция которая будет истинна, если оба аргумента истинна.

# Описание языка на основе регулярных выражений

Регулярные выражения – это один из способов определения формальных языков. Они помогают определять допустимые цепочки во входном языке. Регулярные выражения используются во многих системах.

Алгебра регулярных выражений довольно проста. Она состоит из констант (ε, Ǿ), переменных и трех операций: объединение (+ или |), конкатенации (.) и итерации (\*).

Далее представлен пример регулярных выражений для языка, описанного выше:

Ч = 0|1

П = [a..z][a..z]\*

O = +, -, \*, /,&,!^,~

ЗС = <, >, ==, !=

У = П | Ч (О П | Ч)\* ЗС П | Ч (О П | Ч)\*

B =П=Ч(ОЧ)\*;

В(В)\*

# Контекстно-свободная грамматика

Формальная грамматика – способ описания формального языка, то есть составление предложений некоторого языка.

Контекстно-свободная грамматика – частный случай формальной грамматики, у которого левые части всех продукций являются одиночные нетерминалами. Нетерминал – объект, обозначающий какую-либо сущность языка и не имеющий конкретного символьного значения.

Грамматика определяется следующими элементами:

* – множество терминальных символов, или алфавит терминальных символов;
* – множество нетерминальных символов, или алфавит нетерминальных символов;
* R – множество правил (продукций) грамматики, которые имеют вид a → в
* S – целевой (начальный) символ грамматики.

Далее представлен пример контекстно-свободной грамматики для языка, описанного выше:

G (, , R, S)

= {[0..1], [a..z], +, -, \*, /, <, >, =, }

= {<Ч>, <О>, <И>, <БУКВА>, <В>, <ВВ>}

О={&,|,^,~,!}

Ч={0,1}

R:

<И>→<БУКВА>|<И><БУКВА>

<БУКВА>→[a..z]

<В>→<И>=<Ч>|<Ч><ВВ>

<ВВ>→<О><Ч>|<ВВ><О><Ч>

# Автоматная грамматика

Автоматная грамматика – это бесконтекстная грамматика, каждое правило которой имеет вид А→аВ или А→а, где А и В – нетерминальные символы, а – терминальный символ.

Переход от контекстно-свободной грамматики к автоматной происходит в несколько этапов:

1. Удаление бесполезных символов.

Бесполезные символы возникают в следующих случаях:

* если из символа не может быть получена конечная терминальная цепочка;
* если символ не может быть получен при выводе.

1. Убираем все леворекурсивные правила

Правило вида А→Ва называется леворекурсивным, а правило вида А→аВ – праворекурсивным.

Грамматика, в которой есть только праворекурсивные правила, называется праволинейной.

1. Удаляем цепные правила.

Цепные правила – это правила вида А→В, где А и В – нетерминальные символы.

Из цепочки правил А→В→С→аХ мы оставляем только одно правило А→аХ.

1. Преобразовываем праволинейную грамматику в автоматную.

Из правила А→авсХ получаем 3 правила: А→аВ, В→вС, С→сХ.

Далее представлен пример автоматной грамматики для языка, описанного выше:

<Выражение>

<Выражение>→[a..z] <выр1>

<выр1>→[0,1] <выр2>

<выр2>→=<выр3>

<выр3>→+<выр1>|-<выр1>|\*<выр1>|/<выр1>

<выр3>→AND<выр1>| OR<выр1>| NOT<выр1>| XOR<выр1>

<выр3>→;

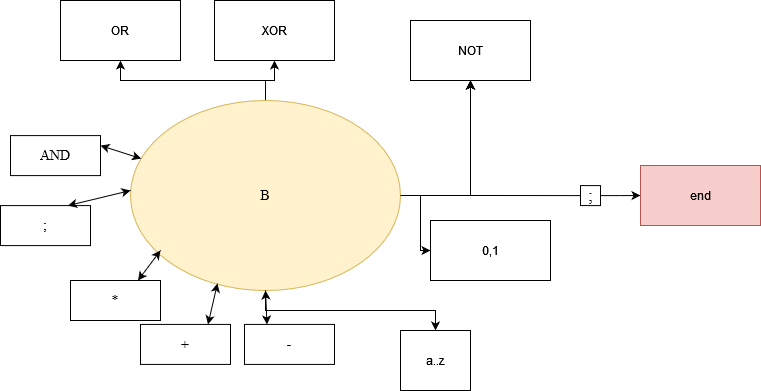
# Недетерминированный автомат

Недетерминированный автомат – автомат, который имеет несколько начальных состояний, а также присутствует неопределенность (из одного состояния по одному символу можно перейти в несколько различных состояний).

Недетерминированный автомат формально представляется как 5-кортеж *A*=(*Q, V, М, S, Z*), состоящий из:

* конечного множества состояний Q;
* входной алфавит V;
* функции переходов М;
* множества начальных состояний S;
* множества заключительных состояний Z.

На рис.1 представлен недетерминированный автомат, а табл.1 является таблицей перехода, в которой по вертикали отмечены состояния графа, а по горизонтали – символы перехода из одного состояния в другое.



*Рис. 1. Недетерминированный автомат*

*Таблица 1*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | XOR | OR | a..z | 0,1 | = | + | - | \* | / | ; | NOT | AND | ! | ( | ) |
| в | в | в | в | в | в | в | в | в | в | в end | в | в | в | в | в |
| end |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

# Детерминированный автомат

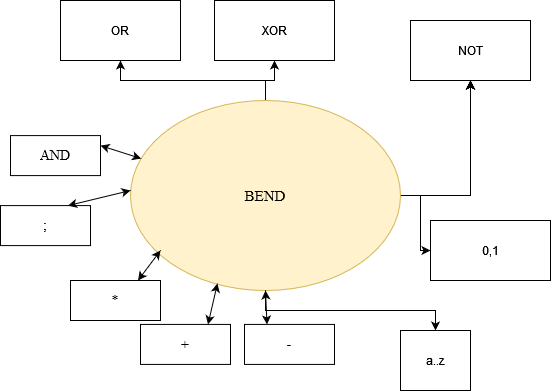
Детерминированный автомат – автомат, который имеет только одно начальное состояние, а также отсутствуют альтернативные состояний, в которые можно перейти при подаче одного символа.

Преобразование недетерминированного автомата в детерминированный происходит за счёт замены нескольких альтернативных состояний на одно, которое будет эквивалентно всем этим состояниям.

Детерминированный автомат формально представляется как 5-кортеж *A*=(*Q, V, М, S, Z*), состоящий из:

* конечного множества состояний Q;
* входной алфавит V;
* функции переходов М;
* начального состояния S;
* множества заключительных состояний Z.

На рис.2 представлен детерминированный автомат, а табл.2 является таблицей перехода, в которой по вертикали отмечены состояния графа, а по горизонтали – символы перехода из одного состояния в другое.



*Рис. 2 Детерминированный автомат*

*Таблица 2*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | XOR | OR | a..z | 0,1 | = | + | - | \* | / | ; | NOT | AND | ! | ( | ) |
| В end | В end | В end | В end | В end | В end | В end | В end | В end | В end | В end | В end | В end | В end | В end | в end |

# Минимальный детерминированный автомат

Минимизация конечного детерминированного автомата – это процедура построения конечного автомата, который будет эквивалентный данному, но имеющий наименьшее число состояний.

Минимизация проводится в два этапа:

1. Множество состояний автомата разбивается на классы эквивалентности;
2. Строится минимальный детерминированный автомат по классам эквивалентности.

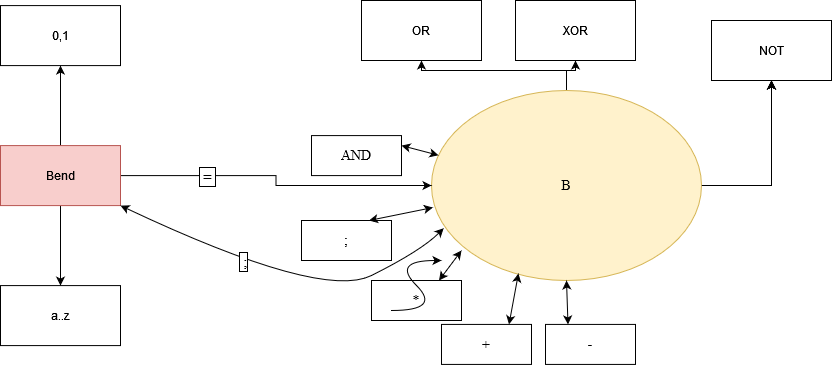
Для того, чтобы разбить множество состояний автомата на классы эквивалентности строится таблица (табл.3).

*Таблица 3*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | XOR | OR | a..z | 0,1 | = | + | - | \* | / | ; | NOT | AND | ! | ( | ) |
| В end |  | | В end | В end |  | | | | | В end |  | |  | | |
| Выр2 | Выр2 | | Выр2 | Выр2 | Выр2 | Выр2 | | | |  | Выр2 | | Выр2 | | |

По результатам табл.3 получились следующие классы эквивалентности: = {вend}, = {Выр2}

По данным классам эквивалентности строим автомат и таблицу перехода (рис.3 и табл.4).



*Рис. 3. Минимизированный детерминированный автомат*

*Таблица 4*

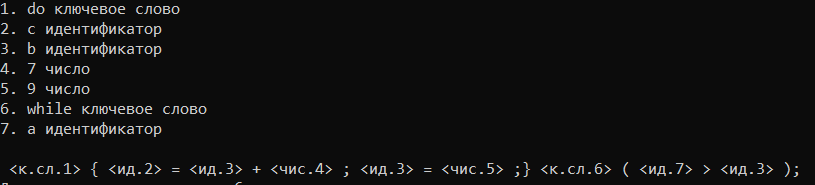
|  |  |
| --- | --- |
| Выр2 | X |
|  | *Вend* |

# Лексический анализатор

Лексический анализатор – это транслятор, который читает исходную программу и выделяет в ее тексте лексемы входного языка. Лексема – это строка символов, с которой связывается лексическая структура, состоящая из пары тип лексемы и некоторые данные.

На вход лексического анализатора поступает текст исходной программы, а выходом лексического анализатора служит последовательность лексем.

Для строки «*do { c=b+7; b=9;} while (a>b);»* лексический анализатор выводит следующую строку (рис.4).



*Рис. 4. Результат работы лексического анализатора.*

# Таблица компиляции

Таблица компиляции – это таблица, которая является результатом работы лексического анализатора.

Таблица компиляции, чаще всего, очень объёмная, поэтому в программе она хранится в виде хеш-таблицы, так как все три операции (поиск, вставка и удаление элементов) в среднем выполняется за время О(1).

Хеш-таблица –⎯ это обычный одномерный массив размером N с необычной адресацией, задаваемой хеш-функцией.

Хеш-функция –⎯ это функция, которая преобразует подаваемый ключ в индекс (число от 0 до N-1) по определенным правилам, которые заложены в данную функцию.

Коллизия –⎯ ситуация, при которой для двух различных ключей значение хеш-функции совпадают .

В данной работе хеш-функцией является метод деления, а методом решения коллизий является метод цепочек.

Метод деления: вычисление индекса для хеш-таблице посредством нахождения остатка от деления ключа k, который подается хеш-функции, на размер таблицы N. Данная хеш-функция выдает числа от 0 до N-1. В общем случае можно записать это следующей функцией:

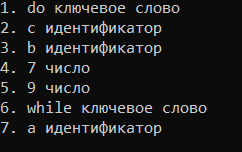
Метод цепочек: хеш-таблица рассматривается как массив связных списков. В данном случае, при возникновении коллизии, добавляем запись в список, который находится в ячейке, которую вычислила хеш-функция.

Добавление и поиск в хеш-таблице происходит примерно одинаково. Для начала вычисляется номер ячейки массива с помощью хеш-функции, а затем переходим в нужную ячейку и работаем со списком. При добавлении переходим в самый последний элемент списка и добавляем новый. При поиске проходим каждый элемент и сравниваем с текущим, если нашли выводим данный элемент на экран, в противном случае выводим, что данного элемента в таблице нет.

*Листинг 1. Структура хеш-таблицы.*

|  |
| --- |
| struct Para { // элемент  int number; // номер в таблице  string pole; // исходный элемент  string title\_lexsem; // название лекстемы  };  class Cell { //ячейка  Para para; //элемент  Cell\* Next; //указатель на следующий элемент  public:  Cell();  Cell(int num, string p, string t\_l);  void add(int num, string p, string t\_l); //добавить элемент  void poisk(string mas); //найти элемент  int poisk\_int(string mas); //номер элемента в таблице компиляции  void show();//показать ячейку  };  class Table { //таблица  Cell\* cell; //массив ячеек  int xesh(string mas); //хеш-функция  public:  void add(int num, string p, string t\_l); //добавить в таблицу  void poisk(string mas); //поиск элемента  int poisk\_int(string mas); //номер элемента в таблице компиляции  void show();//вывести всю таблицу на экран  }; |

Для строки *«do { c=b+7; b=9;} while (a>b);»* таблица компиляции выглядит следующим образом (рис.5).



*Рис. 5. Таблица компиляции.*

# Дерево разбора

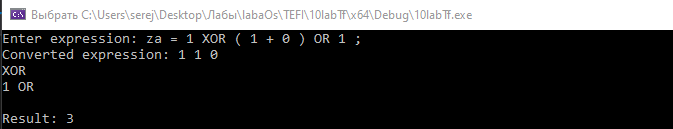
Последовательность лексем (выход работы лексического анализатора) поступает на вход синтаксического анализатора. Синтаксический анализатор, или разбор, – это процесс, в котором исследуется строка лексем и устанавливается, удовлетворяет ли она структурным условиям, явно сформулированным в определении синтаксиса языка.

Дерево разбора – это результат работы синтаксического анализатора.

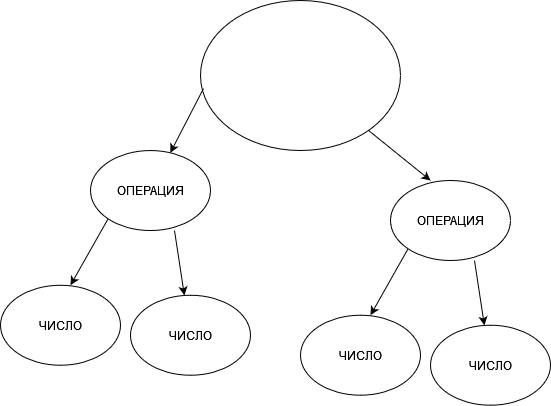
*Листинг 2. Структура дерева.*

|  |
| --- |
| static int code[256];  string operations[4] = { "XOR","AND","NOT","OR" };  struct Token  {  char type;  double number;  };  void InitCode();  double Execute(vector<Token>& exp);  void PrintVectorOfTokens(vector<Token>& vs);  vector<Token> ExpressionConverter(string& exp); |

Для строки *«za = 1 XOR (1 + 0) OR 1;»* дерево разбора выглядит следующим образом (рис.6 и рис.7).



*Рис. 6. Дерево разбора.*



*Рис. 7. Дерево разбора.*

# Генерация кода

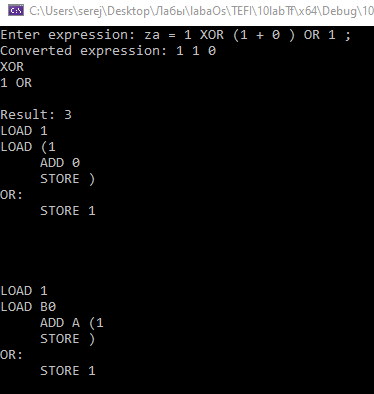
Дерево разбора (результат работы синтаксического анализатора) используется для перевода входного языка. Данный перевод может быть программой на машинном коде, но чаще всего – на промежуточном языке, к примеру, на ассемблере.

Генерация кода на ассемблере происходит при обходе дерева либо слева направо, либо справа налево.

Команды на языке ассемблера, которые используются для перевода языка, описанного выше:

* LOAD – загрузить данные;
* STORE – сохранить данные;
* ADD – сложение;

Для строки *«za = 1 XOR (1 + 0) OR 1;»* код на ассемблере выглядит следующим образом (рис.8).



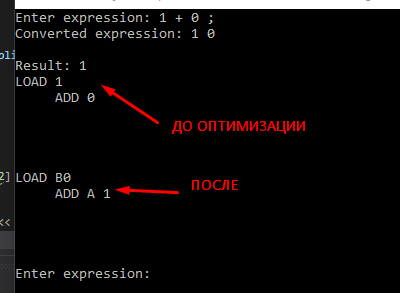
*Рис. 8. Код на ассемблере.*

# Оптимизация кода

Для того, чтобы предыдущий код стал более коротким, используется несколько преобразований:

1. Сложение – коммутативная операция, поэтому последовательность команд LOAD α; ADD β можно заменить последовательностью LOAD β; ADD α*.*
2. Умножение – коммутативная операция, поэтому LOAD α; MPY β можно заменить на LOAD β; MPY α*.*
3. Последовательность операторов вида STORE α; LOAD α можно удалить из программы при условии, что-либо ячейка α не будет далее использоваться, либо перед использованием α будет заполнена заново.
4. Последовательность LOAD α; STORE β можно удалить, если за ней следует другой оператор LOAD и нет перехода к оператору STORE β, а последующие вхождения β будут заменены на α вплоть до того места, где появляется другой оператор STORE β (но исключая его).

Для строки *«za = 1 + 1 ;»* оптимизированный код на ассемблере выглядит следующим образом (рис.9).



*Рис. 9. Оптимизированный код на ассемблере.*

# Список литературы

1. А. Ахо, Дж. Ульман. Теория синтаксического анализа, перевода и компиляции. Т. 1, 2. "Мир", М., 1978
2. А. Ахо, Р. Сети, Дж. Ульман. Компиляторы. Принципы, технологии, инструменты. "Вильямс", М. 2003
3. D. Grune. Parsing Techniques: A Practical Guide. 1st Edition, 1990.
4. Б. К. Мартыненко. Языки и трансляции. Учебное пособие, Изд. СПбГУ, СПб, 2008.
5. Гладкий А. В. Формальные грамматики и языки. – М.: Наука, 1973. — 368 с.
6. Хопкрофт Дж., Мотвани Р., Ульман Дж. Введение в теорию автоматов, языков и вычислений. — М.: Вильямс, 2002 (пер. издания Addison Wesley). — 528 с. — ISBN 5-8459-0261-4
7. Кревский И. Г., Селивёрстов М. Н., Григорьева К. В. Формальные языки, грамматики и основы построения трансляторов: Учебное пособие / Под ред. А. М. Бершадского. — Пенза: Изд-во Пенз. гос. ун-та, 2002. — 124 с.
8. Мартыненко Б. К. Языки и трансляции: Учебное пособие. — СПб.: Издательство С.-Петербургского университета, 2003. — 235 с.
9. Серебряков В. А., Галочкин М. П., Гончар Д. Р., Фуругян М. Г. Теория и реализация языков программирования — М.: МЗ-Пресс, 2006 г., 2-е изд. — ISBN 5-94073-094-9
10. Пентус А. Е., Пентус М. Р. Математическая теория формальных языков. — М.: Интернет-университет информационных технологий, Бином. Лаборатория знаний, 2006. — 248 с.
11. Фомичёв В. С. Формальные языки, грамматики и автоматы: Курс лекций — Интернет-публикация, 2006.
12. Формализованный язык / Б. В. Бирюков // Новая философская энциклопедия : в 4 т. / пред. науч.-ред. совета В. С. Стёпин. — 2-е изд., испр. и доп. — М. : Мысль, 2010. — 2816 с.