МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра ИиСП

КУРСОВАЯ РАБОТА

По дисциплине: «Операционные системы»

НАПРАВЛЕНИЯ ПОДГОТОВКИ

09.03.04 - ПРОГРАММНАЯ ИНЖЕНЕРИЯ

Выполнил: студент 3 курса

специальности «ПС»

группы «ПС-31»

Садовин Алексей Альбертович

Подпись:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:

Нехорошкова Людмила Георгиевна

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Йошкар-Ола

2022 год

Содержание

[Введение 3](#_Toc101353810)

[Основная часть 4](#_Toc101353811)

[Минимизация автоматов Мили и Мура 5](#_Toc101353812)

[Детерминирование конечного автомата 7](#_Toc101353813)

[Перевод регулярного выражения в детерминированный конечный автомат 8](#_Toc101353814)

[Лексический анализатор 9](#_Toc101353815)

[Сквозной пример 11](#_Toc101353816)

[Заключение 12](#_Toc101353817)

# Введение

Целью данной курсовой работы является изучение теории автоматов и формальных языков, алгоритмов преобразования автоматов, а также лексического и синтаксического анализа. Данная теория широко используется при разработке языков программирования и компиляторов, которые являются неотъемлемой частью современных операционных систем.

Теория автоматов лежит в основе всех цифровых технологий и программного обеспечения, так, например, компьютер является частным случаем практической реализации конечного автомата.

# Основная часть

Конечный автомат (КА) — математическая абстракция, модель дискретного устройства, имеющего один вход, один выход и в каждый момент времени находящегося в одном состоянии из множества возможных.

Формально КА определяется как A = (S, X, Y, δ, λ), где:

S – конечное множество состояний автомата;

X, Y – конечные входной и выходной алфавиты соответственно, из которых формируются

строки, считываемые и выдаваемые автоматом;

δ: S × X → S – функция переходов;

λ: S × X → Y – функция выходов.

Различают два класса КА:

* **Автомат Мура** - КА, у которых выходной сигнал зависит только от внутреннего состояния
* **Автомат Милли** - выходной сигнал зависит как от внутреннего состояния, так и от состояния входа.

Конечные автоматы подразделяются на детерминированные и недетерминированные.

* **Детерминированным конечным автоматом** (ДКА) называется такой автомат, в котором нет дуг с меткой *ε* (предложение, не содержащее ни одного символа), и из любого состояния по любому символу возможен переход не более, чем в одно состояние.
* **Недетерминированный конечный автомат** (НКА) является обобщением детерминированного. Недетерминированность автоматов может достигаться двумя способами: либо могут существовать переходы из состояния в состояние, вызываемые пустой цепочкой символов, то есть самопроизвольные переходы без внешних воздействий, либо из одного состояния КА может переходить в разные состояния под воздействием одного и того же символа.

Для изучения и закрепления материала данного курса был выполнены следующие лабораторные работы на языке программирования С++:

* Минимизация автоматов Мили и Мура
* Детерминирование конечного автомата
* Перевод регулярного выражения в детерминированный конечный автомат
* Лексический анализатор

## Минимизация автоматов Мили и Мура

**Минимальный автомат** — это автомат, имеющего наименьшее число состояний.

**Минимизация** – построение автомата эквивалентного исходному и имеющего наименьшее число состояний.

Структура автомата состоит из элементов, которые содержат группу, номер состояния и выходной символ.

Алгоритм минимизации:

1. Считываем тип автомата из файла
2. В зависимости от типа (Милли и Мура) заполняем структуру автомата нужным образом.
3. Запоминание кол-во текущих состояний.
4. По переданному автомату и кол-во текущих состояниям разбиваем множества состояний на классы эквивалентности
5. Если кол-во состояний не изменилось, то переходим к п.6, иначе возвращаемся к п.3
6. Выводим автомат в файл в нужном формате.

Примеры входных данных:

|  |  |
| --- | --- |
| Автомат Мили | Автомат Мура |
| Mi  8  3  2  S4 S5 S3 S5 S7 S1 S5 S3 Y1 Y2 Y2 Y1 Y1 Y1 Y1 Y2 S2 S1 S5 S8 S2 S3 S3 S5 Y2 Y1 Y1 Y2 Y2 Y2 Y2 Y1 S5 S4 S4 S4 S1 S4 S7 S6 Y1 Y2 Y2 Y2 Y1 Y2 Y2 Y2 | Mr  5  2  3  Y1 Y1 Y3 Y2 Y3  S2 S5 S5 S3 S3 S4 S2 S2 S1 S1 |

Структура входных данных построчно:

1. Тип автомата
2. Количество начальных состояний
3. Количество входных символов
4. Количество выходных символов

Примеры выходных данных:

|  |  |
| --- | --- |
| Автомат Мили | Автомат Мура |
| S1 S2 S3 S4  X1 [S4, Y1] [S1, Y2] [S3, Y2] [S1, Y1]  X2 [S2, Y2] [S1, Y1] [S1, Y1] [S3, Y2]  X3 [S1, Y1] [S4, Y2] [S4, Y2] [S4, Y2] | Y1 Y1 Y3 Y3 Y2  S1 S2 S3 S5 S4  X1 [S2] [S5] [S5] [S3] [S3]  X2 [S4] [S2] [S2] [S1] [S1] |

## Детерминирование конечного автомата

**Детерминирование конечного автомата** — перевод недетерминированного конечного автомата в детерминированный**.**

**Формальная грамматика** или просто грамматика в теории формальных языков – способ описания формального языка, то есть выделения некоторого подмножества из множества всех слов некоторого конечного алфавита.

**Грамматика** определяется следующими характеристиками:

* Σ – набор (алфавит) терминальных символов;
* N – набор (алфавит) нетерминальных символов;
* P – набор правил вида: «левая часть» → «правая часть», где:
  + «левая часть» – непустая последовательность терминалов и нетерминалов, содержащая хотя бы один нетерминал;
  + «правая часть» – любая последовательность терминалов и нетерминалов;
* S – стартовый (или начальный) символ грамматики из набора нетерминалов.

**Регулярные грамматики** – более простые, определяют в точности все регулярные языки, и поэтому эквивалентны конечным автоматам и регулярным выражениям.

Регулярная грамматика может быть задана набором правил как левая или правая регулярная грамматика.

**Правая регулярная грамматика**, или **праволинейная грамматика,** – все правила могут быть в одной из следующих форм:

* A → a
* A → aB
* A → ε

**Левая регулярная грамматика**, или **леволинейная грамматика,** – все правила могут быть в одной из следующих форм:

* A → a
* A → Ba
* A → ε

Обозначения:

* заглавные буквы (A, B) – нетерминалы из множества N
* строчные буквы (a, b) – терминалы из множества Σ
* ε – пустая строка, то есть строка длины 0.

Программа может детерминировать как леволинейную, так и праволинейную грамматику. Для распознавания грамматики первой строкой в файле указывается тип грамматики, а второй строкой идет количество состояний.

В программе данные хранятся в виде словаря, который в качестве ключа использует состоние, а значение – список путей из этого состояния с выходными символами.

Алгоритм детерминизации:

1. Считываем тип грамматики из файла
2. В зависимости от типа (леволинейной и праволинейной) заполняем структуру данными, где разница между праволинейной и леволинейной грамматиками только в том, что в первом случае сначала стоит символ–состоние из которого исходят пути, а во втором тот, в которых входят.
3. Выбираем символ от которого начинать детерминизировать автомат. В праволинейной - S, леволинейной – H.
4. Записываем в стек все существующие состояния.
5. Берет из стека следующее множество состоний.
6. Для каждого входного символа формируем множество состояний, в которые можно перейти из какого-либо состояния множества.
7. Непустым из полученных множеств присваиваются идентификаторы, они добавляются в стек;
8. Если стек не пустой то, переходим в состояние 5.
9. Выводим автомат в файл.

Пример входного файла:

|  |  |
| --- | --- |
| Праволинейная грамматика | Леволинейная грамматика |
| * R S:aS|aB|cS|c B:bB|aB|a | * L S:Ab|b A:Sc|c|Aa |

Пример входного файла:

H -> bS | cA

S -> cA

A -> aA | bS

## Перевод регулярного выражения в детерминированный конечный автомат

**Регулярные выражения** — используемый в компьютерных программах, работающих с текстом, формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов. Для поиска используется строка-образец, состоящая из символов и метасимволов и задающая правило поиска. Для манипуляций с текстом дополнительно задаётся строка замены, которая также может содержать в себе специальные символы.

Чтобы преобразовать регулярное выражение в ДКА, нужно:

* Преобразовать регулярное выражение в НКА с ε-переходами
* Устранить ε-переходы
* Построить по НКА эквивалентный ДКА

Алгоритм перевода регулярного выражения в ДКА:

1. Считываем регулярное выражение из файла.
2. Создаем класс “RegexMachine”, куда в конструктор передаем регулярное выражение.
3. В конструкторе класса происходит дробление регулярного выражения
4. Сохраняем начальное кол-во путей.
5. Преобразуем конкатенацию путей автомата(РВ).
6. Преобразуем объединения путей автомата(РВ).
7. Преобразуем пути с символами + и -.
8. Если кол-вол путей не изменилось тогда переходим к пункту 9, иначе возвращаемся ко 4.

Пример входных данных:

(xy\*|ab|(x|a\*))(x|y\*)

Пример выходного файла для этого регулярного выражения:

0 -> a1 | x2 | y3

1 -> a4 | b5 | x6 | y3

2 -> x6 | y7

3 -> y3

4 -> a4 | x6 | y3

5 -> x6 | y3

7 -> x6 | y7

## Лексический анализатор

**Лексический анализ** — процесс аналитического разбора входной последовательности символов на распознанные группы — лексемы — с целью получения на выходе идентифицированных последовательностей, называемых «токенами».

Анализатор считывает программу на языке С++ с файла и выводит списки идентификаторов и токенов.

Входные данные – текст, содержащий:

* Пробельные символы;
* Все лексемы языка С++ (ключевые слова, идентификаторы, операторы, целые, десятичные числа);
* Дробные числа;
* Строковые литералы;
* Блочные и строчные комментарии;

Для работы анализатора объявляем класс, который будет анализировать данных. В него добавлены токены языка C++ в формате словаря, где хранятся пары токен – значение. Программа считывает посимвольно входные данные и при нахождение какого-либо токена запишет это в файл указывает его позицию в файле. При нахождении блочного, строчного комментария, а так же строки анализатор перейдет в соответствующее состояние, корректно его обработав.

Пример выходного файла:

TOKEN VALUE LINE POS

INT int 0 0

IDENTIFIER n 0 4

OPEN\_PARENTHESIS ( 0 8

CLOSE\_PARENTHESIS ) 0 9

OPEN\_CURLY\_BRACKET { 0 10

FLOAT float 1 4

IDENTIFIER f 1 10

ASSIGN = 1 12

DOT . 1 14

NUMBER 0f 1 15

## Сквозной пример

Ориентируясь на выполненные лабораторные работы, можно привести сквозной пример преобразование регулярного выражения в детерминированный автомат.

Регулярное выражение:

(xy\*|ab|(x|a\*))(x|y\*)

Преобразуем его в детерминированный автомат:

0 -> a1 | x2 | y3

1 -> a4 | b5 | x6 | y3

2 -> x6 | y7

3 -> y3

4 -> a4 | x6 | y3

5 -> x6 | y3

7 -> x6 | y7

# Заключение

Целью данной курсовой работы являлось изучение теории автоматов и формальных языков, а также алгоритмов преобразования автоматов, лексического и синтаксического анализа.

В процессе курсовой работы были написаны программы для взаимодействия и преобразования разных типов грамматик, конечных автоматов, использование регулярных выражений. Изучены такие темы как конечные автоматы, недетерминированные и детерминированные автоматы, леволинейная и праволинейная грамматика, перевод автомата Мили в Мура и наоборот, минимизация автоматов, перевод недетерминированного автомата в детерминированный, перевод регулярного выражения в детерминированный конечный автомат, лексический анализатор для языка С++.

Данная практическая работа оказалась очень ценной и необходимой для развития навыков программирования.

Поставленная цель достигнута.