УДК 004.432.2 **DOI 10.52928/2070-1624-2025-44-1-9-13**

СИНТЕЗ НЕДЕТЕРМИНИРОВАННЫХ КОНЕЧНЫХ АВТОМАТОВ ПО РЕГУЛЯРНЫМ ВЫРАЖЕНИЯМ АЛГОРИТМОМ ГЛУШКОВА В ФОРМАТЕ JFF

Д. С. ЛОБАРЁВ, Н. Д. ЛОБАРЁВ
(Псковский государственный университет, Россия)

D. Lobaryov ORCID https://orcid.org/0000-0002-1918-7990
N. Lobaryov ORCID https://orcid.org/0009-0006-2547-8467

Представлены результаты реализации программного средства синтеза недетерминированных конечных автоматов по регулярным выражениям в формате JFF. В качестве метода синтеза автоматов применялся алгоритм Глушкова. При разработке программы использовались интегрированная среда разработки для языка программирования Python – Visual Studio Code, программный пакет JFLAP для визуализации конечных автоматов, а также библиотеки Python xml.etree и pythonds.

Ключевые слова: *недетерминированные конечные автоматы, регулярные выражения, алгоритм Глушкова, JFLAP, Python, xml.etree, pythonds.*

Введение. При проектировании программных и аппаратных средств часто применяются математические модели в виде конечных автоматов. Так, например, с помощью автомата удобно описывать поведение пользователя при разработке интерфейса, управление процессами в операционных системах, моделирование аппаратных устройств в схемотехнике и алгоритмы управления робототехническими системами. Программирование конечных автоматов в настоящее время все шире используется в системах автоматизированного управления, в разработке компьютерных игр, текстовых интерпретаторах и онлайн-переводчиках.

Одним из распространенных способов синтеза конечных автоматов считается синтез по регулярному выражению. Конечные автоматы и регулярные выражения имеют стандартную форму [1]. Теоретической основой исследования стали работы зарубежных и отечественных ученых и практиков: В. М. Глушкова, Ю. Г. Карпова, К. Томпсона, А. А. Шалыто, Э. Ф. Мура, Дж. Мили, Д. Э. Хопкрофта, Р. Мотвани, Д. Ульмана и других[1] [1–3]. Методы исследования базируются на элементах теории множеств, теории алгоритмов и дискретной математике.

Конечные автоматы и регулярные выражения. Автомат – это любой объект, поведение которого может быть описано набором состояний и переходами между ними. Множество конечных автоматов можно разделить на два больших класса: автоматы-распознаватели и автоматы-преобразователи. Задача первых – определить, принадлежит ли входное слово к заданному множеству слов. Обычно они выдают ответ в виде «да» или «нет». Автоматы-распознаватели чаще всего используются в лексических анализаторах в качестве программных моделей. Аппаратная реализация конечного автомата обычно встречается для второго класса – автоматов-преобразователей. Такие автоматы преобразуют входное слово в некоторое выходное слово.

Автомат-распознаватель – это объект, который задается структурой $(Q, q_0, F, \Sigma, \delta)$, где Q – множество состояний; q_0 – начальное состояние; F – множество конечных состояний, т. е. все такие состояния, в которых завершение работы автомата считается корректным; Σ – множество входных символов или входной алфавит; δ – функция переходов. Пример графа автомата, распознающего слова *lite*, *lie* и *lit*, представлен на рисунке 1.

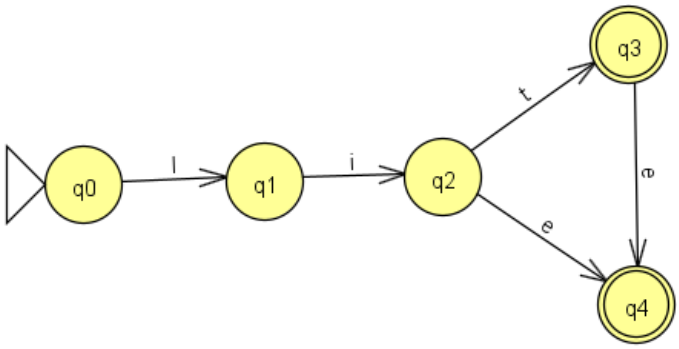


Рисунок 1. – Граф автомата, распознающего слова *lite*, *lie* и *lit*

[1] Thurston A. Ragel State Machine Compiler User Guide. URL: https://www.colm.net/files/ragel/ragel-guide-6.10.pdf.

Конечные автоматы делятся на детерминированные и недетерминированные. Детерминированный конечный автомат – это конечный автомат, принимающий или отклоняющий заданную строку символов путем прохождения через последовательность состояний, определенных строкой [1]. Недетерминированный конечный автомат – это детерминированный конечный автомат, который не выполняет следующих условий: любой его переход единственным образом определяется по текущему состоянию и входному символу; чтение входного символа требуется для каждого изменения состояния.

Регулярным языком, или языком, распознаваемым автоматом, называется множество таких слов, после прочтения которых автомат из начального состояния q_0 попадает в конечное из множества F [2].

Существуют следующие способы задания регулярного языка:

1. Перечень слов: представляет собой список слов. Например, для графа из рисунка 1 перечень слов, представляющий собой язык, распознаваемый автоматом, будет включать *lite*, *lie*, *lit*.
2. Регулярная грамматика, т. е. перечень правил, который определяет принадлежность слова к языку.
3. Регулярное выражение.

С помощью алгебры регулярных выражений описывают правила, которые можно применять к операциям, содержащимся в данном регулярном выражении. Алгебра регулярных выражений – это частный случай алгебры Клини – некоммуникативное идемпотентное полукольцо с нейтральным элементом [2].

Синтез недетерминированных конечных автоматов алгоритмом Глушкова. В работе рассматривается алгоритм построения графа методом Глушкова. Для получения множеств по регулярному выражению составляется синтаксическое дерево с помощью библиотеки pythonds языка программирования Python.

Алгоритм Глушкова был разработан советским ученым Виктором Михайловичем Глушковым. Суть алгоритма заключается в преобразовании регулярного выражения в эквивалентный ему недетерминированный конечный автомат без переходов по пустой строке [3]. Алгоритм Глушкова состоит из четырех этапов.

Первый этап – линеаризация выражения. Каждый символ алфавита, встречающийся в регулярном выражении e, переименовывается, так что в новом выражении e' каждый символ встречается не более одного раза. Конструкция Глушкова, по сути, основана на том факте, что e' представляет собой язык $L(e')$. Пусть A – старый алфавит, а B – новый.

Второй этап – вычисление множеств $P(e')$, $D(e')$, $F(e')$ и $\Lambda(e')$. Первое множество, $P(e')$, – это набор символов, который встречается как первый символ слова из языка $L(e')$. Второе множество, $D(e')$, – это набор символов, которыми может заканчиваться слово из языка $L(e')$. Третье множество, $F(e')$, представляет собой набор пар соседних символов, которые могут 26 раз последовательно встречаться в словах из языка $L(e')$. Эти наборы математически определяются формулами

$$P(e') = \{x \in B \mid xB^* \bigcap L(e') \neq \varnothing\};$$

$$D(e') = \{y \in B \mid B^* y \bigcap L(e') \neq \varnothing\};$$

$$F(e') = \{u \in B^2 \mid B^* uB^* \bigcap L(e') \neq \varnothing\}.$$

Они вычисляются методом индукции по структуре выражения, как описано ниже. Последнее множество $\Lambda(e')$ содержит пустое слово, если это слово принадлежит языку $L(e')$, и является пустым множеством в противном случае. Формально это $\Lambda(e') = \{\lambda\} \bigcap L(e')$, где λ обозначает пустое слово.

Третий этап – вычисление автомата, распознающего язык, определенный множествами $P(e')$, $D(e')$, $F(e')$ и $\Lambda(e')$. Язык, определяемый наборами P, D и F, по своей сути является набором слов, которые начинаются с символа множества P, заканчиваются символом множества D и пар соседних символов множества F, необязательно также включая пустое слово; то есть это язык $L' = (PB^* \bigcap B^* D) \setminus B^* (B^2 \setminus F) B^* \bigcup \Lambda(e')$, где \ означает разницу множеств, или относительное дополнение между множествами. Строго говоря, именно вычисление автомата для языка, обозначаемого этим линеаризованным выражением, является конструкцией Глушкова.

Четвертый этап – удаление линеаризации путем замены каждого индексированного символа алфавита B исходным символом алфавита A.

В роли входных данных выступает регулярное выражение типа данных str, в роли выходных данных – XML-дерево, содержащее информацию о недетерминированном конечном автомате. Предлагаемое программное средство состоит из 5 программных модулей, написанных на языке Python версии 3.11.0. Например, в [4] представлен опыт поиска оптимального решения в дифференциальной линейно-квадратичной задаче на языке программирования Python в облачной среде Google Colab.

Кратко опишем каждый модуль разрабатываемого программного средства:

1. init.py – модуль инициализации. Чтобы программа начала свою работу, необходимо запустить именно этот модуль. В данном файле производится импорт всех остальных модулей и соблюдается строгая последовательность выполнения функций каждого модуля.
2. grammar_check.py – модуль проверки введенного регулярного выражения. Представляет собой лексический и синтаксический анализатор.
3. transform.py – модуль преобразования регулярного выражения для правильного построения синтаксического дерева.
4. all_sets.py – модуль, реализующий 2-й этап алгоритма Глушкова. С помощью данного модуля по преобразованному регулярному выражению строится синтаксическое дерево, по которому формируются множества:
   - множество переходов из начального состояния (т. е. $P(e')$);
   - множество конечных состояний (т. е. $D(e')$);
   - множество всех возможных пар символов (т. е. $F(e')$);
   - множество, отвечающее за конечность начального состояния (т. е. $\Lambda(e')$).
5. create_xml_tree.py – модуль, реализующий 3-й и 4-й этапы алгоритма Глушкова. Также данный модуль создает xml-код, содержащий информацию о конечном автомате, визуализацию которого можно посмотреть в пакете JFLAP[2].

JFLAP (Java Formal Languages and Automata Package) – свободное ПО, созданное для образовательных целей в области теории автоматов. Основной особенностью JFLAP является широкий и регулярно пополняемый функционал, позволяющий интерактивно решать такие типовые задачи, как работа с регулярными языками (создание детерминированных и недетерминированных конечных автоматов, регулярных грамматик и выражений), преобразования (недетерминированного конечного автомата в детерминированный, недетерминированного конечного автомата в регулярное выражение или грамматику, минимизация детерминированного конечного автомата), создание и преобразование контекстно-свободных языков, работа с машинами Тьюринга, а также синтез конечных автоматов по регулярным выражениям хорошо известным алгоритмом Томпсона.

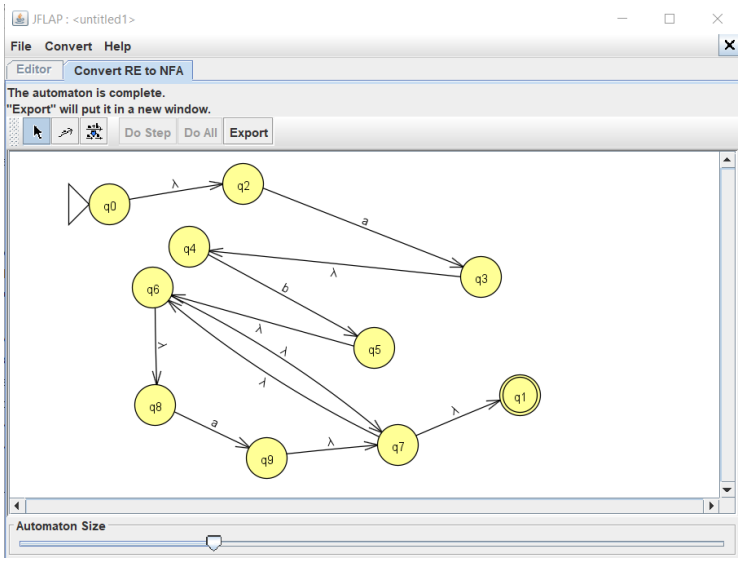


Рисунок 2. – Окно JFLAP с результатом синтеза недетерминированного конечного автомата по регулярному выражению aba*

Особенность данного исследования заключается в построении графа с помощью алгоритма Глушкова. В отличие от алгоритма Томпсона, он позволяет строить конечные автоматы без пустых переходов. Работа программного средства функционально состоит из четырех этапов:

1. Проверка регулярного выражения. На данном этапе осуществляется чтение введенного регулярного выражения. Программа должна оповестить пользователя о том, что он ввел регулярное выражение правильно или неправильно. Если выражение было указано неверно, то программа не допустит его дальнейшей обработки. Эту задачу выполняет модуль grammar_check.py.

[2] JFLAP: off. website. URL: www.jflap.org.

2. Преобразование регулярного выражения. После выполнения первого этапа необходимо преобразовать исходное регулярное выражение для правильного построения синтаксического дерева. Также после преобразования формируется новый алфавит B в виде списка, что является результатом первого шага алгоритма Глушкова. Эту задачу выполняет модуль transform.py.

3. Вычисление множеств. После выполнения второго этапа программа на основе преобразованного выражения создает синтаксическое дерево и с помощью него вычисляет множества начальных состояний $P(e')$, конечных состояний $D(e')$, всех возможных пар символов $F(e')$ и множество, определяющее конечность начального состояния $\Lambda(e')$. Результат вычисления этих множеств означает завершение второго шага алгоритма Глушкова. Каждое такое множество, за исключением последнего, также будет представлено в формате списков. На этом этапе задействованы следующие модули: init.py, all_sets.py.

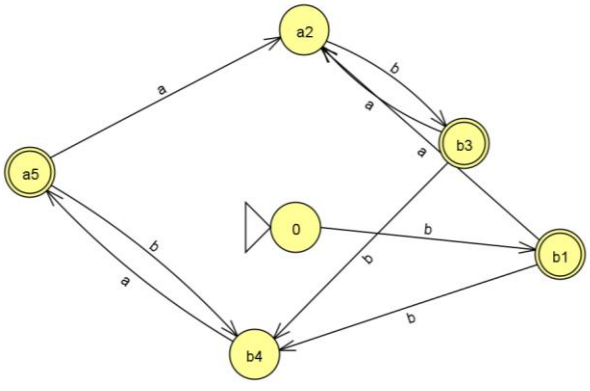
4. Формирование выходного файла. На основе списков, полученных на предыдущем этапе, программа формирует выходной файл в формате JFF, содержащий информацию о структуре недетерминированного конечного автомата. Эту задачу выполняет модуль create_xml_tree.py.

В качестве примера возьмем регулярное выражение $e = b(ab + ba)^*$. Запускаем модуль init.py и вводим регулярное выражение. Результат работы программного средства продемонстрирован на рисунке 3.

```
Введите регулярное выражение без пробелов: b(ab+ba)*
Поздравляем! Выражение составлено верно!
Исходное выражение: e = b.(a.b+b.a)*
Преобразованное выражение: e' = ( b1 . ( ( ( a2 . b3 ) + ( b4 . a5 ) ) * ) )
Новый алфавит: B = ['b1', 'a2', 'b3', 'b4', 'a5']
Множество переходов из начального состояния: P(e') = ['b1']
Множество переходов в конечные состояния: D(e') = ['b3', 'a5', 'b1']
Множество всех возможных пар символов: F(e') = [['b4', 'a5'], ['a2', 'b3'], ['b1', 'b4'], ['a5', 'a2'],
 ['a5', 'b4'], ['b1', 'a2'], ['b3', 'a2'], ['b3', 'b4']]
Определяем конечность начального состояния: Λ(e') = False
JFF-файл создан успешно!
```

Рисунок 3. – Ввод данных, проверка и преобразование регулярного выражения

Визуализация конечного автомата в программе JFLAP показана на рисунке 4. Модуль create_xml_tree.py формирует файл формата JFF, который представляет из себя XML-дерево. Для работы с XML-файлами необходимо использовать библиотеку xml.etree.



Input	Result
	Reject
b	Accept
bab	Accept
bba	Accept
ab	Reject
bb	Reject
babba	Accept
bababab	Accept
bbababa	Accept
babbaabba	Accept

Рисунок 4. – Построение графа конечного автомата по регулярному выражению

Заключение. В статье представлена авторская разработка специальной программы синтеза недетерминированных конечных автоматов по регулярным выражениям. Для синтеза используется алгоритм Глушкова, в результате чего получен недетерминированный конечный автомат без пустых переходов. Результаты тестирования программы указывают на необходимость введения дополнительных параметров.

Одной из таких модификаций может стать экранирование специальных символов для создания возможности их использования в регулярных выражениях.

Разработанное программное средство может быть использовано в образовательных целях, а также в качестве альтернативного метода синтеза недетерминированных конечных автоматов по регулярному выражению для программы JFLAP. Областями практического применения результатов разработки могут быть управление потоком задач и реализация синтеза цифровых устройств, описание алгоритмов, а также моделирование процессов в различных сферах, таких как лингвистика, химия, физика, биология, математика и др.

ЛИТЕРАТУРА

1. Хопкрофт Д. Э., Мотвани Р., Ульман Д. Введение в теорию автоматов, языков и вычислений. – М.: Вильямс, 2008. – 527 с.
2. Когабаев, Н. Т., Дискретная математика и теория алгоритмов. – Новосибирск: ИПЦ НГУ, 2023. – 126 с.
3. Глушков В. М. Абстрактная теория автоматов // Успехи математических наук. – 1961. – Т. 16, вып. 5(101). – С. 3–62. – URL: https://www.mathnet.ru/links/6e142a93807e67659b1200367fd3b613/rm6668.pdf (дата обращения: 12.01.2024).
4. Лобарёв Д. С., Толбухин Д. В. Поиск оптимального решения дифференциальной линейно-квадратичной задачи управления средствами Python в облачной среде Google Colab // Науч.-техн. вестн. Поволжья. – 2021. – № 12. – С. 208–213.

REFERENCES

1. Khopkroft, D. E., Motvani, R., & Ul'man, D. (2008). *Vvedenie v teoriyu avtomatov, yazykov i vychislenii.* Moscow: Vil'yams. (In Russ.).
2. Kogabaev, N. T., (2023). *Diskretnaya matematika i teoriya algoritmov.* Novosibirsk: IPTs NGU. (In Russ.).
3. Glushkov, V. M. The abstract theory of automata. (1961). *Russian Mathematical Surveys*, *16*(5), 1–53. DOI: 10.1070/RM1961v016n05ABEH004112.
4. Lobarev, D. S., & Tolbukhin, D. V. (2021). Poisk optimal'nogo resheniya differentsial'noi lineino-kvadratichnoi zadachi upravleniya sredstvami Python v oblachnoi srede Google Colab [Search for the optimal solution to the differential linear-quadratic Python management problem in the Google Colab cloud environment]. *Nauchno-tekhnicheskii vestnik Povolzh'ya*. (12), 208–213. (In Russ., abstr. in Engl.).

Поступила 17.01.2025

SYNTHESIS OF NONDETERMINISTIC FINITE AUTOMATON FROM REGULAR EXPRESSIONS BY GLUSHKOV'S ALGORITHM IN JFF FORMAT

D. LOBARYOV, N. LOBARYOV
(Pskov State University, Russia)

The paper presents the results of the implementation of a software tool for synthesizing nondeterministic finite automaton using regular expressions in JFF format. Glushkov's algorithm is used as a method for synthesizing automata. When developing the program, the integrated development environment for the Python – Visual Studio Code, the JFLAP software package for visualizing finite automaton, as well as the Python libraries xml.etree and pythonds were used.

Keywords: *nondeterministic finite automaton, regular expressions, Glushkov's algorithm, JFLAP, Python, xml.etree, pythonds.*