МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление 02.03.01 Математика и компьютерные науки

Отчет по дисциплине «Математическая логика и теория автоматов»

Лабораторная работа №2

«Проверка корректности года в разных форматах»

Вариант 19

Обучающийся: _____ Шклярова Ксения Алексеевна

Группа: 5130201/20102

Руководитель: _____ Востров Алексей Владимирович

«_____»_____20___г.

Санкт-Петербург, 2025

Содержание

Введение

В данной лабораторной работе по заданному варианту необходимо построить регулярное выражение, затем недетерминированный конечный автомат и детерминировать его. Реализовать программу, которая проверяет введенный текст через реализацию конечного автомата. Также необходимо реализовать функцию случайной генерации верной строки по полученному конечному автомату.

Задание для 19 варианта: год от -4000 до 2100 в формате до н.э и н.э. в русскоязычном и англоязычном форматах.

1 Математическое описание

1.1 Форматы представления года

В данной работе рассматриваются русский и английский форматы представления года в диапазоне от 4000 года до н.э. до 2100 года н.э. Важно отметить, что нулевой год в рамках данной реализации не поддерживается.

Отсутствие нулевого года в исторической хронологии обусловлено рядом факторов:

1. Система летоисчисления от Рождества Христова, разработанная Дионисием Малым в VI веке, не включала понятие нуля, так как использовала римские цифры.

2. Исторический отсчёт времени предполагает непосредственный переход от 1 года до н. э. к 1 году н. э., без промежуточного нулевого года: 1 год до н. э. $\rightarrow$ 1 год н. э.

1.1.1 Русский формат

В русском языке год обычно записывается арабскими цифрами. Для указания эры используются сокращения "н. э."(нашей эры) и "до н. э."(до нашей эры), которые ставятся после числа.

Допустимы два варианта записи:

• С "г."(год): "2024 г. н. э."(более формальный, характерен для документов и научных текстов).

• Без "г.": "2024 н. э."(более распространен в обычной письменной речи).

Примеры:

• "Октавиан Август умер в 14 г. н. э."или "Октавиан Август умер в 14 н. э."

• "Юлий Цезарь был убит в 44 г. до н. э."или "Юлий Цезарь был убит в 44 до н. э."

1.1.2 Английский формат

В английском языке используются латинские сокращения:

• AD (Anno Domini – "в год Господень") – обозначает годы нашей эры.

• BC (Before Christ – "до Рождества Христова") – обозначает годы до нашей эры.

Правила написания AD и BC:

• Пишутся без точек и пробелов (не A.D. или B.C.).

• AD (Anno Domini) обычно ставится перед годом, но в некоторых случаях допустимо написание после года: – В старых текстах (особенно до XX века) иногда встречается "1066 AD".

– В некоторых научных, исторических или религиозных работах авторы могут сознательно ставить AD после года для стилистического разнообразия.

– В разговорной речи или неформальном письме порядок менее строгий.

Примеры:

• "The Battle of Hastings took place in AD 1066."

- "Cleopatra died in 30 BC."
- "The Roman Empire fell in 476 AD."

1.2 Регулярное выражение

Регулярные выражения - формальный язык шаблонов для поиска и выполнения манипуляций с подстроками в тексте.

Регулярные множества - это (бесконечные) множества цепочек, построенных из символов конечного словаря по определённым правилам.

Регулярные множества, как множества цепочек, построенные над конечным словарём (по определённым правилам) - это языки. Их называют регулярными языками.

Класс регулярных выражений над конечным словарем Σ определяется так:

- $\emptyset$, ε и любой символ $a \in \Sigma$ — регулярные выражения;

Если R_1 и R_2 — регулярные выражения, то регулярными выражениями будут:

- их сумма $R_1 + R_2$;
- их произведение (конкатенация) $R_1 \cdot R_2$;
- итерация R_1^* и усеченная итерация R_1^+ (унарные операции);

Приоритеты:

- $*$ — итерация — наиболее приоритетная операция,
- $+$ — усеченная итерация;
- $\cdot$ произведение (конкатенация);
- $+$ (или $|$) — сложение.

Скобки ( ) — для группировки (задание порядка выполнения операций).

Ниже приведены правила построения регулярных множеств:

- Объединение: $L_1 \cup L_2 = \{\alpha | \alpha \in L_1 or \alpha \in L_2\}$.
- Конкатенация: $L_1 * L_2 = \{\alpha\beta | \alpha \in L_1 or \beta \in L_2\}$.
- Итерация: $L^* = L^0 \cup L^2 \cup L^3 ... = \cup_{k=0}^{\infty} L^k$.
- Усечённая итерация $L^+ = L^1 \cup ... = \cup_{k=1}^{\infty} L^k$

1.3 Конечный автомат

Математическая модель дискретного устройства, которая описывается набором $S = (S, X, Z, \delta, \lambda, a_0)$, где

- S = $\{s_0, \ldots, s_m, \ldots, s_M\}$ – множество состояний;
- X = $\{x_1, \ldots, x_f, \ldots, x_F\}$ – множество входных сигналов;
- Z = $\{z_1, \ldots, z_g, \ldots, z_G\}$ – множество выходных сигналов;
- Δ – функция переходов КА, которая $(s_m, x_f) \to s_s$, т.е. $s_s = \delta(s_m, x_f), s_s \in S$.
- Λ – функция выходов КА, которая $(s_m, x_f) \to z_g$, т.е. $z_g = \lambda(s_m, x_f), z_g \in Z$.
- s_0 – начальное состояние.

Выделяют недетерменированный и детерменированный конченый автоматы. Они различаются тем, что недетерменированный конечный автомат не выполняет условие, что любой его переход однозначно определяется по текущему состоянию и входному символу.

Из регулярного выражения можно получить недетерменированный автомат по следующим правилам:

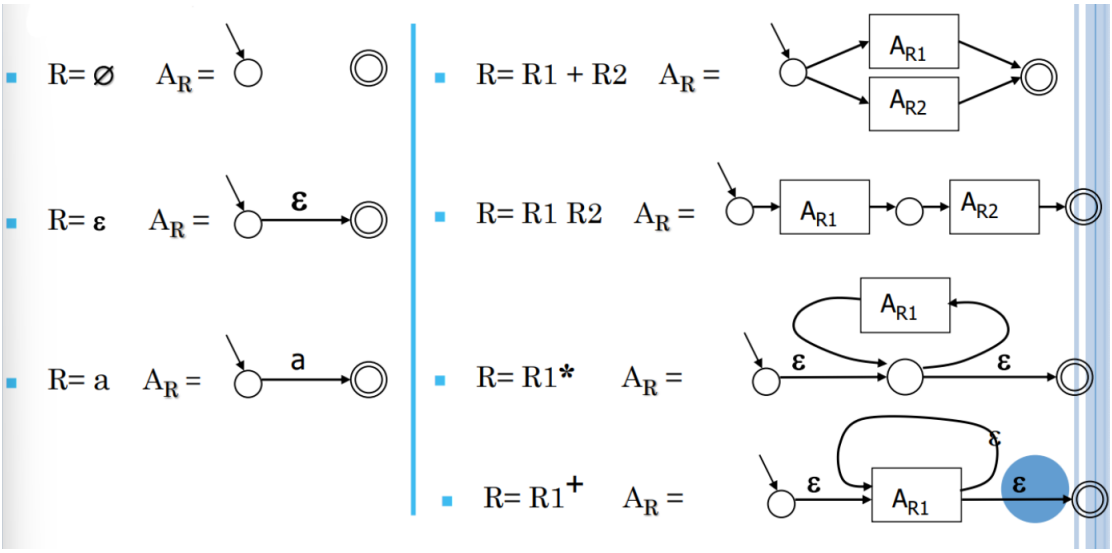

Рис. 1. Правила построения недетерменированного автомата по регулярному выражению.

После чего этот недетерменированный автомат можно свести к детерменированному и реализовать в программе.

1.4 Теорема Клини

Классы регулярных множеств $\mathcal{L}_R$ и автоматных языков $\mathcal{L}_A$ совпадают ($\mathcal{L}_R = \mathcal{L}_A$).

Это значит, что:

1) любой язык, распознаваемый КА, может быть задан формулой (рег. выр);

2) для любого регулярного языка существует задающий его КА.

Схема распознавания представлена на рис. 2:

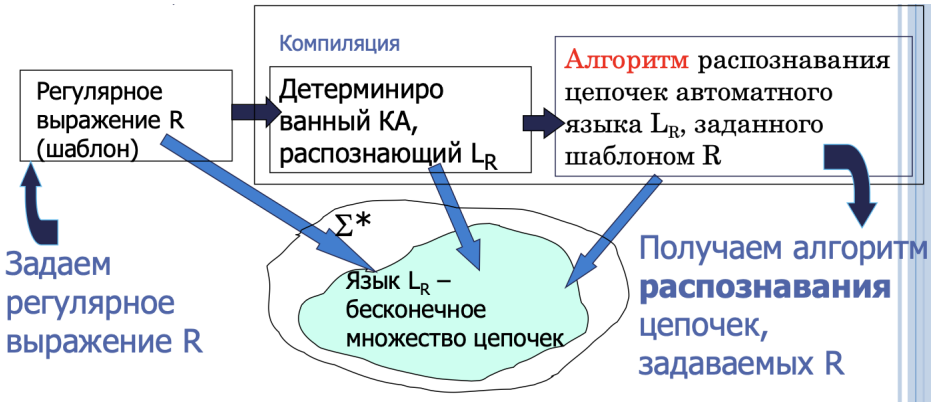

Рис. 2. Схема распознавания

1.5 Регулярное выражение для представления года

Для валидации и генерации представлений года приложение использует следующее составное регулярное выражение, состоящее из двух частей, разделенных оператором | (ИЛИ):

1. Регулярное выражение для лет до нашей эры (до н.э. / BC):

   (([1-3][0-9]{3}|4000|[1-9][0-9]{0,2})\s(до\sн\.\s*э\.|BC))

   [1-3][0-9]3: Сопоставляет годы от 1000 до 3999.

   4000: Сопоставляет год 4000.

   [1-9][0-9]0,2: Сопоставляет годы от 1 до 999.

   \s: Один пробельный символ.

   (до\sн\.\s*э\.|BC): Группа, захватывающая обозначение "до н.э."или "BC".

2. Регулярное выражение для лет нашей эры (н.э. / AD):

   ((1[0-9]{3}|20[0-9]{2}|2100|[1-9][0-9]{0,2})\s(н\.\s*э\.|AD))

   1[0-9]3: Сопоставляет годы от 1000 до 1999.

   20[0-9]2: Сопоставляет годы от 2000 до 2099.

   2100: Сопоставляет год 2100.

   [1-9][0-9]0,2: Сопоставляет годы от 1 до 999.

   \s: Один пробельный символ.

   (н\.*э|AD): Группа, захватывающая обозначение "н.э."или "AD".

Итоговое регулярное выражение выглядит следующим образом:

(([1-3][0-9]{3}|4000|[1-9][0-9]{0,2})\s(до\sн\.\s*э\.|BC))|((1[0-9]{3}|20[0-9]{2}|2100|[1-9][0-9]{0,2})\s

(н\.\s*э\.|AD))

Это регулярное выражение позволяет приложению корректно определять и генерировать год в указанном диапазоне (4000 до н.э. - 2100 н.э.) как в русскоязычном, так и в англоязычном форматах.

1.6 Недетерминированный конечный автомат

По регулярному выражению был построен недетерменированный автомат, который представлен на рис. 2.

По регулярному выражению, описывающему год в диапазонах от 4000 до н.э. до 2100 н.э., был построен недетерминированный конечный автомат (НКА). На рисунках 2–8 представлены НКА для общего формата даты и НКА для отдельных диапазонов годов с поддержкой обозначений эр.

Принцип работы автомата:

Переход по цифрам приводит к анализу диапазона (например, "1"→ проверка 1–999 н.э., "2"→ 2000–2100 н.э.). Состояния обрабатывают как русскоязычные ("н.э. "до н.э."), так и англоязычные ("AD "BC") варианты. Достижение цветного состояния означает корректное распознавание даты, автомат переходит в финальное состояние.

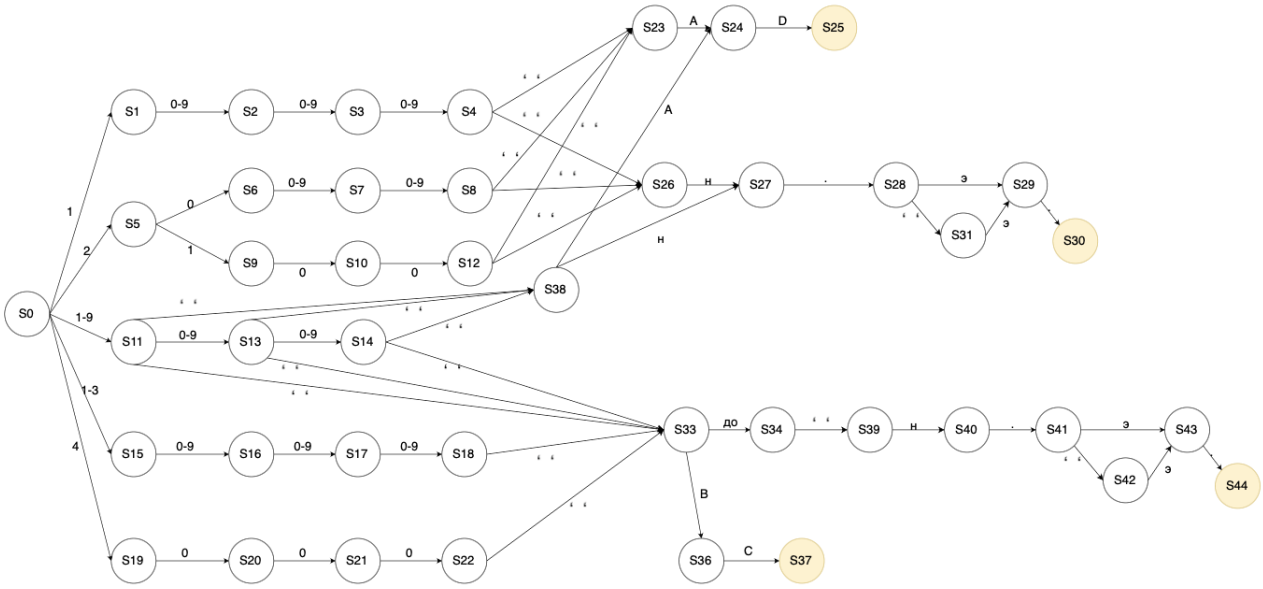

Рис. 3. Недетерминированный автомат для даты

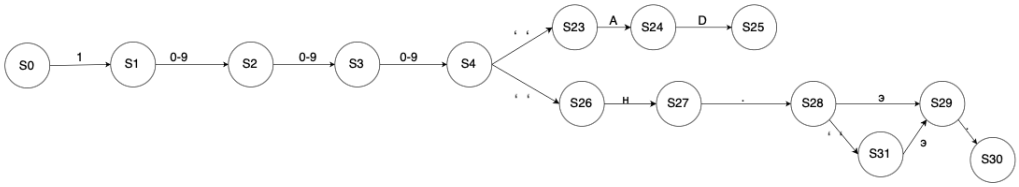

Рис. 4. Недетерминированный автомат для лет 1000-1999 нашей эры

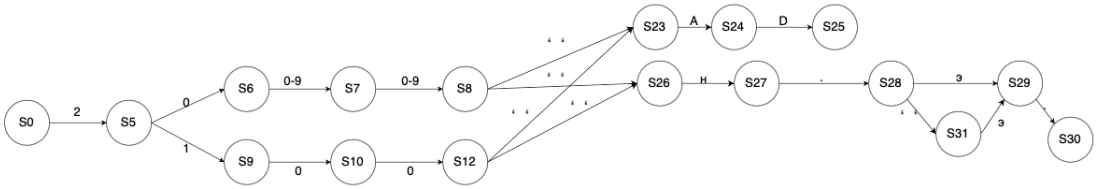

Рис. 5. Недетерминированный автомат для лет 2000-2100 нашей эры

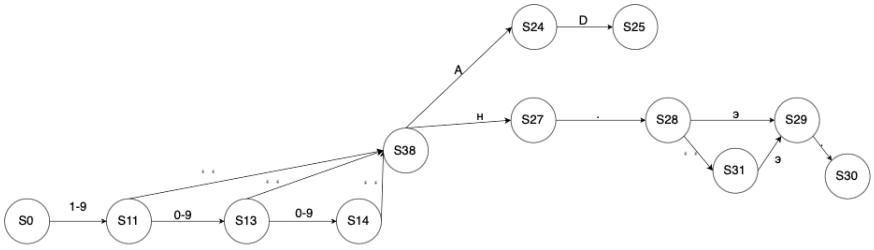

Рис. 6. Недетерминированный автомат для лет 1-999 нашей эры

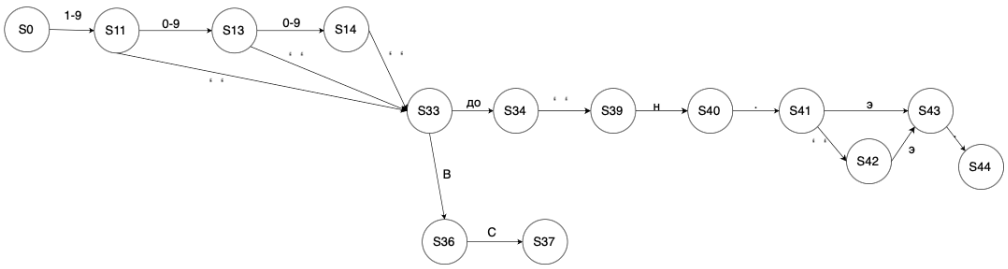

Рис. 7. Недетерминированный автомат для лет 1-999 до нашей эры

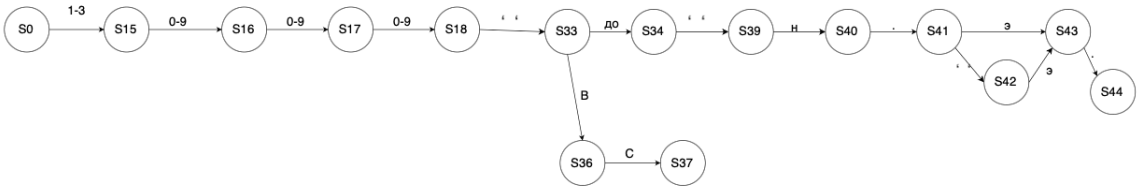

Рис. 8. Недетерминированный автомат для лет 1000-3999 до нашей эры

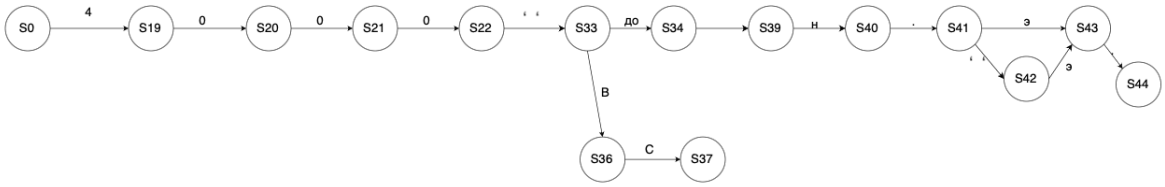

Рис. 9. Недетерминированный автомат для лет 4000 до нашей эры

1.7 Детерминированный конечный автомат

После применения алгоритма детерминизации к исходному недетерминированному автомату (НКА) был получен детерминированный конечный автомат (ДКА), который представлен на рис. 9.

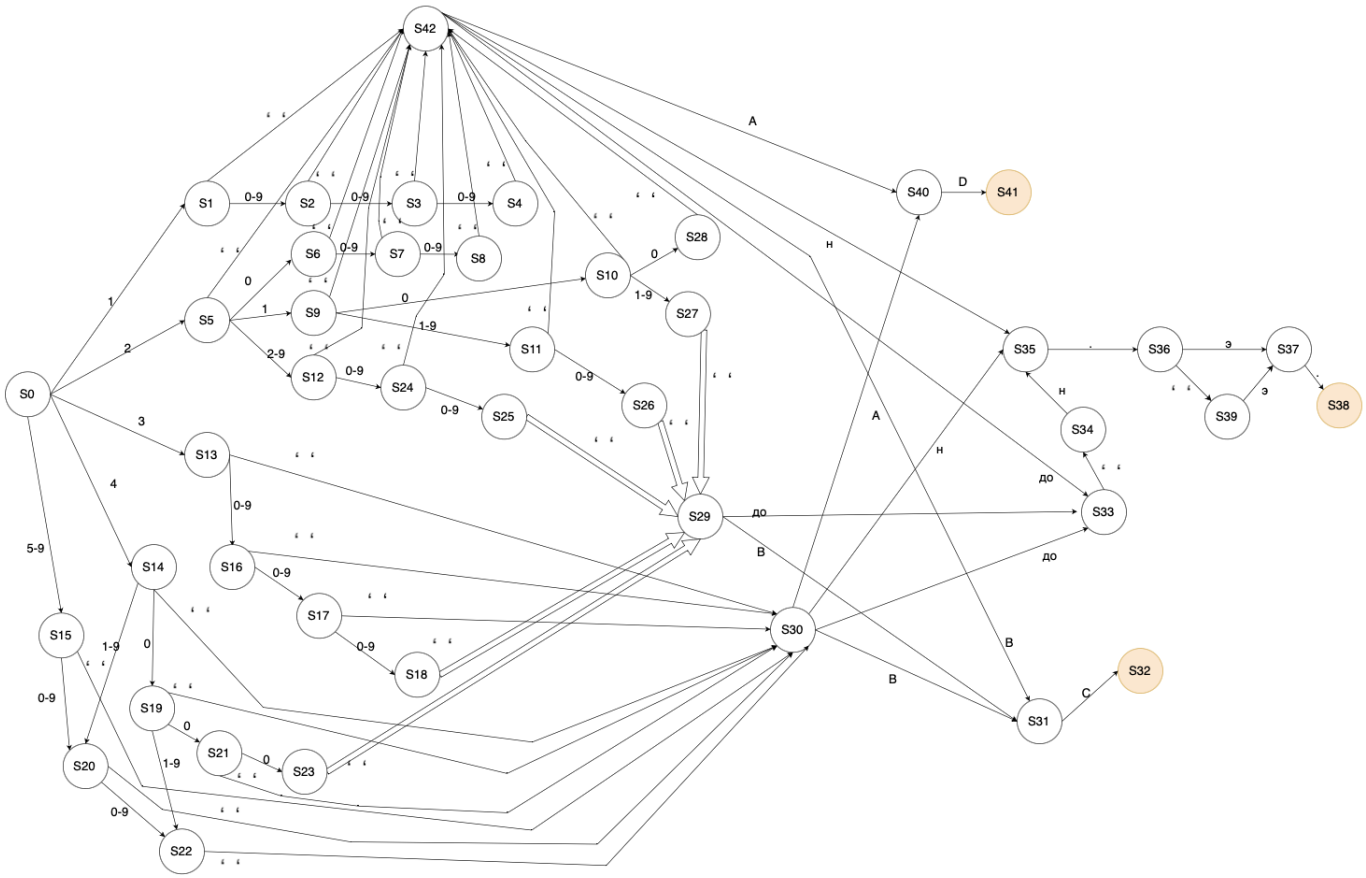

Рис. 10. Результат анализа пятого примера кода

Таблица 1. Таблица переходов детерминированного конечного автомата

Текущее состояние	Входящий символ	Следующее состояние
s0	1	s1
	2	s5
	3	s13
	4	s14
	5-9	s15
s1	0-9	s2
	’ ’	s42
s2	0-9	s3
	’ ’	s42
s3	0-9	s4
	’ ’	s42
s4	’ ’	s42
s5	0	s6
	1	s9
	2-9	s12
	’ ’	s42
s6	0-9	s7
	’ ’	s42
s7	0-9	s8
	’ ’	s42
s8	’ ’	s42
s9	0	s10
	1-9	s11
	’ ’	s42
s10	0	s28
	1-9	s27
	’ ’	s42
s28	’ ’	s42
s27	’ ’	s29
s11	0-9	s26
	’ ’	s42
s26	’ ’	s29

Текущее состояние	Входящий символ	Следующее состояние
s12	0-9	s24
	’ ’	s42
s24	0-9	s25
	’ ’	s42
s25	’ ’	s29
s13	0-9	s16
	’ ’	s30
s16	0-9	s17
	’ ’	s30
s17	0-9	s18
	’ ’	s30
s18	’ ’	s29
s14	0	s19
	1-9	s20
	’ ’	s30
s19	0	s21
	1-9	s22
	’ ’	s30
s21	0	s23
	’ ’	s30
s23	’ ’	s29
s15	0-9	s20
	’ ’	s30
s20	0-9	s22
	’ ’	s30
s22	’ ’	s30
s42	A	s40
	B	s31
	до	s33
	н	s35
s29	B	s31
	до	s33

Текущее состояние	Входящий символ	Следующее состояние
s30	A	s40
	B	s31
	до	s33
	н	s35
s40	D	s41 (конец)
S31	C	s32 (конец)
s33	’ ’	s34
s34	н	s35
s35	.	s36
s36	э	s37
	’ ’	s39
s39	э	s37
s37	.	s38 (конец)

2 Особенности реализации

2.1 Структуры данных

В реализации детерминированного конечного автомата (ДКА) проверки формата года с эрами используются следующие структуры данных и программные конструкции:

1. Перечисление (Enum) — State. Хранение всех возможных состояний автомата.

2. Словарь (Dict) — transition_table. Таблица переходов между состояниями. Ключ — текущее состояние, значение — функция-обработчик символа.

3. Списки (List) — parts в generate_valid_date(). Постепенное накопление символов при генерации случайной даты.

4. Множество (Set) — Проверка финальных состояний. Проверка, является ли текущее состояние допустимым конечным.

5. Строки (str). Хранение и обработка входных/выходных данных.

2.2 Класс для хранения состояний

Класс State определяет все возможные состояния детерминированного конечного автомата (ДКА) для проверки формата года с эрами ("до н.э."/ "н.э."или "BC"/ "AD").

Состояния группируются по логике работы:

START — начальное состояние.

S1, S2, ..., S5_9 — состояния для обработки цифр года.

WS1, WS2, WS3 — состояния для обработки пробелов перед суффиксом эры. before_d, before_o, our, era — состояния для разбора "до н.э."/ "н.э.".

A, AD, B, BC — состояния для разбора "AD"/ "BC".

ERROR — состояние ошибки.

Объявление класса см. в листинге 1.

Листинг 1. Хранение состояний автомата

```
class State(Enum):
    START = auto()  # Начальное состояние

    S1 = auto()
    S1_ = auto()
    S1__ = auto()
    S1___ = auto()

    S2 = auto()
    S20 = auto()
    S20_ = auto()
    S20__ = auto()
```

```
13      S21 = auto()
14      S210 = auto()
15      S2100 = auto()
16      S210_ = auto()
17      S21_ = auto()
18      S21__ = auto()
19      S2_ = auto() #вторая цифра 2-9
20      S2__ = auto()
21      S2___ = auto()
22
23      S3 = auto()
24      S3_ = auto()
25      S3__ = auto()
26      S3___ = auto()
27
28      S4 = auto()
29      S40 = auto()
30      S400 = auto()
31      S4000 = auto()
32      S40_ = auto()
33      S4_ = auto()
34      S4__ = auto()
35
36      S5_9 = auto()
37      S5_ = auto() #первая цифра 5-9
38      S5__ = auto()
39
40      WS1 = auto()
41      WS2 = auto()
42      WS3 = auto()
43
44      before_d = auto()
45      before_o = auto()
46      ws1 = auto()
47      our = auto()
48      dot1 = auto()
49      ws2 = auto()
50      era = auto()
51      dot2 = auto() #
52
53      B = auto()
54      BC = auto() #
55
56      A = auto()
```

```
57      AD = auto() #
58
59      ERROR = auto()  # Ошибка
```

2.3 Заполение таблицы состояний

Глобальная переменная transition_table хранит правила переходов между состояниями. Каждому состоянию сопоставлена лямбда-функция, которая принимает символ и возвращает новое состояние.

Ключ: текущее состояние (например, State.START).

Значение: функция, которая анализирует входной символ и решает, куда перейти.

Функция init_transition_table() инициализирует transition_table, которую и принимает, результатом её выполнения является заполнение всех возможных переходов между состояниями. Заполнение таблицы происходит следующим образом: для каждого состояния из State определяется лямбда-функция, которая: принимает текущий символ (symbol), проверяет его соответствие допустимым значениям (цифры, пробелы, буквы) и возвращает следующее состояние или State.ERROR, если символ некорректен.

Реализацию данной функции см. в листинге 2.

Листинг 2. Заполнение таблицы состояний

```
1  transition_table: Dict[State, Callable[[str], State]] = {}
2
3  def init_transition_table():
4      # Переходы для Start
5      transition_table[State.START] = lambda symbol: (
6          State.S1 if symbol == '1' else
7          State.S2 if symbol == '2' else
8          State.S3 if symbol == '3' else
9          State.S4 if symbol == '4' else
10         State.S5_9 if '5' <= symbol <= '9' else
11         State.ERROR
12     )
13     # Переходы для S1
14     transition_table[State.S1] = lambda symbol: (
15         State.S1_ if '0' <= symbol <= '9' else
16         State.WS1 if symbol == ' ' else
17         State.ERROR
18     )
19     transition_table[State.S1_] = lambda symbol: (
20         State.S1__ if '0' <= symbol <= '9' else
21         State.WS1 if symbol == ' ' else
```

```
22          State.ERROR
23      )
24      transition_table[State.S1__] = lambda symbol: (
25          State.S1___ if '0' <= symbol <= '9' else
26          State.WS1 if symbol == ' ' else
27          State.ERROR
28      )
29      transition_table[State.S1___] = lambda symbol: (
30          State.WS1 if symbol == ' ' else
31          State.ERROR
32      )
33      # Переходы для S2
34      transition_table[State.S2] = lambda symbol: (
35          State.S20 if symbol == '0' else
36          State.S21 if symbol == '1' else
37          State.S2_ if '2' <= symbol <= '9' else
38          State.WS1 if symbol == ' ' else
39          State.ERROR
40      )
41      transition_table[State.S20] = lambda symbol: (
42          State.S20_ if '0' <= symbol <= '9' else
43          State.WS1 if symbol == ' ' else
44          State.ERROR
45      )
46      transition_table[State.S20_] = lambda symbol: (
47          State.S20__ if '0' <= symbol <= '9' else
48          State.WS1 if symbol == ' ' else
49          State.ERROR
50      )
51      transition_table[State.S20__] = lambda symbol: (
52          State.WS1 if symbol == ' ' else
53          State.ERROR
54      )
55      transition_table[State.S21] = lambda symbol: (
56          State.S210 if symbol == '0' else
57          State.S21_ if '1' <= symbol <= '9' else
58          State.WS1 if symbol == ' ' else
59          State.ERROR
60      )
61      transition_table[State.S210] = lambda symbol: (
62          State.S2100 if symbol == '0' else
63          State.S210_ if '1' <= symbol <= '9' else
64          State.WS1 if symbol == ' ' else
65          State.ERROR
```

```
66      )
67      transition_table[State.S2100] = lambda symbol: (
68          State.WS1 if symbol == ' ' else
69          State.ERROR
70      )
71      transition_table[State.S210_] = lambda symbol: (
72          State.WS2 if symbol == ' ' else
73          State.ERROR
74      )
75      transition_table[State.S21_] = lambda symbol: (
76          State.S21__ if '0' <= symbol <= '9' else
77          State.WS1 if symbol == ' ' else
78          State.ERROR
79      )
80      transition_table[State.S21__] = lambda symbol: (
81          State.WS2 if symbol == ' ' else
82          State.ERROR
83      )
84      transition_table[State.S2_] = lambda symbol: (
85          State.S2__ if '0' <= symbol <= '9' else
86          State.WS1 if symbol == ' ' else
87          State.ERROR
88      )
89      transition_table[State.S2__] = lambda symbol: (
90          State.S2___ if '0' <= symbol <= '9' else
91          State.WS1 if symbol == ' ' else
92          State.ERROR
93      )
94      transition_table[State.S2___] = lambda symbol: (
95          State.WS2 if symbol == ' ' else
96          State.ERROR
97      )
98      # Переходы для S3
99      transition_table[State.S3] = lambda symbol: (
100         State.S3_ if '0' <= symbol <= '9' else
101         State.WS3 if symbol == ' ' else
102         State.ERROR
103     )
104     transition_table[State.S3_] = lambda symbol: (
105         State.S3__ if '0' <= symbol <= '9' else
106         State.WS3 if symbol == ' ' else
107         State.ERROR
108     )
109     transition_table[State.S3__] = lambda symbol: (
```

```
110         State.S3___ if '0' <= symbol <= '9' else
111         State.WS3 if symbol == ' ' else
112         State.ERROR
113     )
114     transition_table[State.S3___] = lambda symbol: (
115         State.WS2 if symbol == ' ' else
116         State.ERROR
117     )
118     # Переходы для S4
119     transition_table[State.S4] = lambda symbol: (
120         State.S40 if symbol == '0' else
121         State.S4_ if '1' <= symbol <= '9' else
122         State.WS3 if symbol == ' ' else
123         State.ERROR
124     )
125     transition_table[State.S40] = lambda symbol: (
126         State.S400 if symbol == '0' else
127         State.S40_ if '1' <= symbol <= '9' else
128         State.WS3 if symbol == ' ' else
129         State.ERROR
130     )
131     transition_table[State.S400] = lambda symbol: (
132         State.S4000 if symbol == '0' else
133         State.WS3 if symbol == ' ' else
134         State.ERROR
135     )
136     transition_table[State.S4000] = lambda symbol: (
137         State.WS2 if symbol == ' ' else
138         State.ERROR
139     )
140     transition_table[State.S40_] = lambda symbol: (
141         State.WS3 if symbol == ' ' else
142         State.ERROR
143     )
144     transition_table[State.S4_] = lambda symbol: (
145         State.S4__ if '0' <= symbol <= '9' else
146         State.WS3 if symbol == ' ' else
147         State.ERROR
148     )
149     transition_table[State.S4__] = lambda symbol: (
150         State.WS3 if symbol == ' ' else
151         State.ERROR
152     )
153     # Переходы для S5_9
```

```
154     transition_table[State.S5_9] = lambda symbol: (
155         State.S5_ if '0' <= symbol <= '9' else
156         State.WS3 if symbol == ' ' else
157         State.ERROR
158     )
159     transition_table[State.S5_] = lambda symbol: (
160         State.S5__ if '0' <= symbol <= '9' else
161         State.WS3 if symbol == ' ' else
162         State.ERROR
163     )
164     transition_table[State.S5__] = lambda symbol: (
165         State.WS3 if symbol == ' ' else
166         State.ERROR
167     )
168     # Переходы для WS
169     transition_table[State.WS1] = lambda symbol: (
170         State.A if symbol == 'A' else
171         State.our if symbol == 'н' else
172         State.B if symbol == 'B' else
173         State.before_d if symbol == 'д' else
174         State.ERROR
175     )
176     transition_table[State.WS2] = lambda symbol: (
177         State.B if symbol == 'B' else
178         State.before_d if symbol == 'д' else
179         State.ERROR
180     )
181     transition_table[State.WS3] = lambda symbol: (
182         State.A if symbol == 'A' else
183         State.our if symbol == 'н' else
184         State.B if symbol == 'B' else
185         State.before_d if symbol == 'д' else
186         State.ERROR
187     )
188     transition_table[State.before_d] = lambda symbol: (
189         State.before_o if symbol == 'о' else
190         State.ERROR
191     )
192     transition_table[State.before_o] = lambda symbol: (
193         State.ws1 if symbol == ' ' else
194         State.ERROR
195     )
196     transition_table[State.ws1] = lambda symbol: (
197         State.our if symbol == 'н' else
```

```
198             State.ERROR
199         )
200         transition_table[State.our] = lambda symbol: (
201             State.dot1 if symbol == '.' else
202             State.ERROR
203         )
204         transition_table[State.dot1] = lambda symbol: (
205             State.era if symbol == 'э' else
206             State.ws2 if symbol == ' ' else
207             State.ERROR
208         )
209         transition_table[State.ws2] = lambda symbol: (
210             State.era if symbol == 'э' else
211             State.ERROR
212         )
213         transition_table[State.era] = lambda symbol: (
214             State.dot2 if symbol == '.' else
215             State.ERROR
216         )
217         transition_table[State.A] = lambda symbol: (
218             State.AD if symbol == 'D' else
219             State.ERROR
220         )
221         transition_table[State.B] = lambda symbol: (
222             State.BC if symbol == 'C' else
223             State.ERROR
224         )
```

2.4 Генерация дат и проверка на корректность

Функция check_date_format(input_string: str) проверяет, соответствует ли входная строка (input_string) формату года с указанием эры ("до н.э. "н.э. "BC "AD") в заданном диапазоне (от 4000 до н.э. до 2100 н.э.). Входные параметры: input_string: str — строка, представляющая дату (например, "2025 н.э."или "1500 до н.э."). Возвращаемое значение: True — если строка соответствует формату и допустимому диапазону, False — если строка некорректна или выходит за границы диапазона. Реализация данной функции заключается в следующем: начиная с состояния State.START, для каждого символа в input_string вызывается соответствующая функция перехода из transition_table для текущего состояния. Если переход ведёт в State.ERROR — сразу возвращает False. После обработки всех символов проверяется, является ли текущее состояние финальным (например, State.AD). Возвращается True, если состояние финальное, иначе False.

Функция generate_valid_date() генерирует случайную строку с корректным форматом года

и указанием эры (например, "1500 до н.э."или "2025 AD"). Функция не принимает аргументов. Возвращает str — случайная дата в допустимом формате. Создается пустой список parts для накопления символов. Начиная с состояния State.START, в цикле генерируются случайные цифры, соответствующие текущему состоянию (например, для S2 — цифры от 2000 до 2100). Эти цифры добавляются в parts. Далее так же случайным образом выбирается один из суффиксов, который после добавляяся в parts. В результате parts объединяется в строку, которая и является результатом выполнения.

Реализацию данных функций см. в листинге 3.

Листинг 3. Генерация дат и проверка на корректность

```
def check_date_format(input_string: str) -> bool:
    current_state = State.START

    for symbol in input_string:
        if current_state == State.ERROR:
            return False

        try:
            current_state = transition_table[current_state](symbol)
        except KeyError:
            current_state = State.ERROR

    # Проверяем, находимся ли в конечном состоянии
    return current_state in {
        State.AD, State.BC, State.dot1, State.dot2,
        State.S1___, State.S20__, State.S2100, State.S210_,
        State.S21__, State.S2___, State.S3___, State.S4000,
        State.S40_, State.S4__, State.S5__
    }


def generate_valid_date() -> str:
    parts: List[str] = []
    state = State.START

    while True:
        # Проверка конечных состояний
        if state in {State.AD, State.BC, State.dot2}:
            break
        elif state == State.dot1:
            parts.append('.')
            break

```

```
        # Генерация числовой части
        if state == State.START:
            choice = random.choice(['1', '2', '3', '4', '5', '6', '7', '8', '9'])
            parts.append(choice)
            state = State[f'S{choice}'] if choice in ['1', '2', '3', '4'] else State.
                S5_9

        # Генерация для чисел, начинающихся с 1
        elif state == State.S1:
            choice = random.choice('0123456789 ')
            parts.append(choice)
            if choice == ' ':
                state = State.WS1
            else:
                state = State.S1_
        elif state == State.S1_:
            choice = random.choice('0123456789 ')
            parts.append(choice)
            if choice == ' ':
                state = State.WS1
            else:
                state = State.S1__
        elif state == State.S1__:
            choice = random.choice('0123456789 ')
            parts.append(choice)
            if choice == ' ':
                state = State.WS1
            else:
                state = State.S1___
        elif state == State.S1___:
            parts.append(' ')
            state = State.WS1

        # Генерация для чисел, начинающихся с 2
        elif state == State.S2:
            choice = random.choice('0123456789 ')
            parts.append(choice)
            if choice == ' ':
                state = State.WS1
            elif choice == '0':
                state = State.S20
            elif choice == '1':
                state = State.S21
            else:
```

```
77                  state = State.S2_
78          elif state == State.S20:
79              choice = random.choice('0123456789 ')
80              parts.append(choice)
81              if choice == ' ':
82                  state = State.WS1
83              else:
84                  state = State.S20_
85          elif state == State.S20_:
86              choice = random.choice('0123456789 ')
87              parts.append(choice)
88              if choice == ' ':
89                  state = State.WS1
90              else:
91                  state = State.S20__
92          elif state == State.S20__:
93              parts.append(' ')
94              state = State.WS1
95          elif state == State.S21:
96              choice = random.choice('123456789 ')
97              parts.append(choice)
98              if choice == ' ':
99                  state = State.WS1
100             elif choice == '0':
101                 state = State.S210
102             else:
103                 state = State.S21_
104         elif state == State.S210:
105             choice = random.choice('0123456789 ')
106             parts.append(choice)
107             if choice == ' ':
108                 state = State.WS1
109             elif choice == '0':
110                 state = State.S2100
111             else:
112                 state = State.S210_
113         elif state == State.S2100:
114             parts.append(' ')
115             state = State.WS1
116         elif state == State.S210_:
117             parts.append(' ')
118             state = State.WS2
119         elif state == State.S21_:
120             choice = random.choice('0123456789 ')
```

```
121             parts.append(choice)
122             if choice == ' ':
123                 state = State.WS1
124             else:
125                 state = State.S21__
126         elif state == State.S21__:
127             parts.append(' ')
128             state = State.WS2
129         elif state == State.S2_:
130             choice = random.choice('0123456789 ')
131             parts.append(choice)
132             if choice == ' ':
133                 state = State.WS1
134             else:
135                 state = State.S2__
136         elif state == State.S2__:
137             choice = random.choice('0123456789 ')
138             parts.append(choice)
139             if choice == ' ':
140                 state = State.WS1
141             else:
142                 state = State.S2___
143         elif state == State.S2___:
144             parts.append(' ')
145             state = State.WS2
146 
147         # Генерация для чисел, начинающихся с 3
148         elif state == State.S3:
149             choice = random.choice('0123456789 ')
150             parts.append(choice)
151             if choice == ' ':
152                 state = State.WS3
153             else:
154                 state = State.S3_
155         elif state == State.S3_:
156             choice = random.choice('0123456789 ')
157             parts.append(choice)
158             if choice == ' ':
159                 state = State.WS3
160             else:
161                 state = State.S3__
162         elif state == State.S3__:
163             choice = random.choice('0123456789 ')
164             parts.append(choice)
```

```
165             if choice == ' ':
166                 state = State.WS3
167             else:
168                 state = State.S3___
169         elif state == State.S3___:
170             parts.append(' ')
171             state = State.WS2
172 
173         # Генерация для чисел, начинающихся с 4
174         elif state == State.S4:
175             choice = random.choice('0123456789 ')
176             parts.append(choice)
177             if choice == ' ':
178                 state = State.WS3
179             elif choice == '0':
180                 state = State.S40
181             else:
182                 state = State.S4_
183         elif state == State.S40:
184             choice = random.choice('0123456789 ')
185             parts.append(choice)
186             if choice == ' ':
187                 state = State.WS3
188             elif choice == '0':
189                 state = State.S400
190             else:
191                 state = State.S40_
192         elif state == State.S400:
193             choice = random.choice('0 ')
194             parts.append(choice)
195             if choice == ' ':
196                 state = State.WS3
197             else:
198                 parts.append(' ')
199                 state = State.WS2
200         elif state == State.S4_:
201             choice = random.choice('0123456789 ')
202             parts.append(choice)
203             if choice == ' ':
204                 state = State.WS3
205             else:
206                 state = State.S4__
207         elif state == State.S4__:
208             parts.append(' ')
```

```
209             state = State.WS3
210
211         # Генерация для чисел, начинающихся с 5–9
212         elif state == State.S5_9:
213             choice = random.choice('0123456789 ')
214             parts.append(choice)
215             if choice == ' ':
216                 state = State.WS3
217             else:
218                 state = State.S5_
219         elif state == State.S5_:
220             choice = random.choice('0123456789 ')
221             parts.append(choice)
222             if choice == ' ':
223                 state = State.WS3
224             else:
225                 state = State.S5__
226         elif state == State.S5__:
227             parts.append(' ')
228             state = State.WS3
229
230         # Генерация суффиксов
231         elif state == State.WS1 or state == State.WS3:
232             choice = random.choice(['н.э.', 'AD', 'BC', 'до н.э.'])
233             if choice == 'AD':
234                 parts.append('AD')
235                 state = State.AD
236             elif choice == 'BC':
237                 parts.append('BC')
238                 state = State.BC
239             elif choice == 'н.э.':
240                 parts.append('н.э.')
241                 state = State.dot2
242             else:
243                 parts.append('до н.э.')
244                 state = State.dot2
245         elif state == State.WS2:
246             choice = random.choice(['BC', 'до н.э.'])
247             if choice == 'BC':
248                 parts.append('BC')
249                 state = State.BC
250             else:
251                 parts.append('до н.э.')
252                 state = State.dot2
```

```
253         else:
254             break
255 
256     return ''.join(parts)
```

2.5 Реализация меню программы

Функция get_user_input() -> str запрашивает у пользователя ввод даты вручную, проверяет его на корректность (не пустой ввод) и возвращает строку или None (если пользователь решил отменить ввод). На вход принимает введенную пользователем строку. Возвращаемое значение: str — введённая пользователем строка с датой (например, "2025 н.э."), None — если пользователь ввёл q для отмены.

Функция show_menu() отображает консольное меню и вызывает соответствующую функцию. На вход принимает введенное пользователем значение, результатом выполнения является вызов соответствующей функции.

Реализацию данных функций см. в листинге 4.

Листинг 4. Реализация меню программы

```
1  def get_user_input() -> str:
2      print("\nДоступные форматы дат:")
3      print("Год + суффикс (1000 AD, 2000 BC, 3000 н.э., 1500 до н.э.)")
4  
5      while True:
6          user_input = input("\nВведите дату или 'q' для отмены: ").strip()
7  
8          # Проверка на команду выхода
9          if user_input.lower() == 'q':
10             return None
11 
12         # Проверка пустого ввода
13         if not user_input:
14             print("Ошибка: Пустой ввод. Пожалуйста, введите дату.")
15             continue
16 
17         return user_input
18 
19 def show_menu():
20     while True:
21         print("\nМеню:")
22         print("1. Сгенерировать случайную дату")
23         print("2. Ввести дату вручную")
24         print("3. Выход")
```

```
25
26          choice = input("Выберите действие (1-3): ")
27
28          if choice == "1":
29              date = generate_valid_date()
30              print(f"\nСгенерированная дата: {date}")
31              print("Проверка корректности:", "OK" if check_date_format(date) else "ERROR
                    ")
32          elif choice == "2":
33              date = get_user_input()
34              if check_date_format(date):
35                  print(f"\nВведенная дата: {date}")
36                  print("Проверка корректности: OK")
37              else:
38                  print("Некорректный формат даты")
39          elif choice == "3":
40              print("Выход из программы.")
41              break
42          else:
43              print("Некорректный ввод.")
```

3 Результаты работы программы

На рис. 10-18 показаны результаты работы программы.

```
Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): аа
Некорректный ввод.
```

Рис. 11. Обработка неверного ввода пользователя при выборе пункта в меню

```
Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): 1

Сгенерированная дата: 1150 AD
Проверка корректности: OK

Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): 1

Сгенерированная дата: 2962 BC
Проверка корректности: OK

Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): 1

Сгенерированная дата: 913 BC
Проверка корректности: OK
```

Рис. 12. Генерация случайной даты №1-3

```
Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): 1

Сгенерированная дата: 2930 BC
Проверка корректности: OK

Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): 1

Сгенерированная дата: 1 до н.э.
Проверка корректности: OK

Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): 1

Сгенерированная дата: 771 BC
Проверка корректности: OK
```

Рис. 13. Генерация случайной даты №4-6

```
Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): 1

Сгенерированная дата: 434 н.э.
Проверка корректности: OK

Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): 1

Сгенерированная дата: 441 до н.э.
Проверка корректности: OK

Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): 1

Сгенерированная дата: 1 AD
Проверка корректности: OK
```

Рис. 14. Генерация случайной даты №7-9

На рис. 14 показан пример корректной и некорректной введенной даты. Вторая дата является ошибочной, так как «выходит за верхнюю границу».

```
Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): 2

Доступные форматы дат:
Год + суффикс (1000 AD, 2000 BC, 3000 н.э., 1500 до н.э.)

Введите дату или 'q' для отмены: 897 н.э.

Введенная дата: 897 н.э.
Проверка корректности: OK

Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): 2

Доступные форматы дат:
Год + суффикс (1000 AD, 2000 BC, 3000 н.э., 1500 до н.э.)

Введите дату или 'q' для отмены: 2567 н. э.
Некорректный формат даты
```

Рис. 15. Проверка даты, введенной пользователем

На рис. 15 показан пример корректной и некорректной введенной даты. Первая дата является ошибочной, так как «нулевого» года не существует.

```
Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): 2

Доступные форматы дат:
Год + суффикс (1000 AD, 2000 BC, 3000 н.э., 1500 до н.э.)

Введите дату или 'q' для отмены: 0 BC
Некорректный формат даты

Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): 2

Доступные форматы дат:
Год + суффикс (1000 AD, 2000 BC, 3000 н.э., 1500 до н.э.)

Введите дату или 'q' для отмены: 328 AD

Введенная дата: 328 AD
Проверка корректности: OK
```

Рис. 16. Проверка даты, введенной пользователем

```
Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): 2

Доступные форматы дат:
Год + суффикс (1000 AD, 2000 BC, 3000 н.э., 1500 до н.э.)

Введите дату или 'q' для отмены: 4000 BC

Введенная дата: 4000 BC
Проверка корректности: OK
```

Рис. 17. Проверка даты, введенной пользователем

На рис. 17 показан пример корректной и некорректной введенной даты. Первая дата является ошибочной, так как «выходит за нижнюю границу».

```
Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): 2

Доступные форматы дат:
Год + суффикс (1000 AD, 2000 BC, 3000 н.э., 1500 до н.э.)

Введите дату или 'q' для отмены: 5643 до н.э.
Некорректный формат даты

Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): 2

Доступные форматы дат:
Год + суффикс (1000 AD, 2000 BC, 3000 н.э., 1500 до н.э.)

Введите дату или 'q' для отмены: 567 до н. э.

Введенная дата: 567 до н. э.
Проверка корректности: OK
```

Рис. 18. Проверка даты, введенной пользователем

```
Меню:
1. Сгенерировать случайную дату
2. Ввести дату вручную
3. Выход
Выберите действие (1-3): 3
Выход из программы.
```

Рис. 19. Выход из программы

Заключение

В рамках лабораторной работы по заданному варианту (год от -4000 до 2100 в форматах "до н.э."и "н.э."на русском и английском языках) были построены регулярное выражение и конечный автомат, который был детерминирован.

Реализованная программа включает в себя следующие функции:

• Создание и заполнение таблицы состояний на основе построенного детерминированного конечного автомата.

• Проверка введенной строки на соответствие формату года (от -4000 до 2100 в форматах "до н.э."и "н.э. "BC"и "AD") с использованием полученного конечного автомата.

• Генерация корректной строки формата года на основе полученного конечного автомата.

Функциональность программы была протестирована на наборе примеров, включающем как корректные, так и некорректные входные данные. Результаты тестирования представлены в отчете.

Эта работа на практике подтверждает теорему Клини, демонстрируя эквивалентность регулярных выражений и конечных автоматов. Мы описываем язык (допустимые форматы года) регулярным выражением, а затем строим по нему НКА и ДКА. Это показывает, что любой регулярный язык (заданный выражением) распознаваем автоматом. Программа, проверяющая строки с помощью ДКА, подтверждает обратное: для каждого регулярного языка существует автомат, его распознающий.

Из плюсов программы можно выделить использование Enum для состояний и typing для аннотаций, что улучшает читаемость и поддерживаемость кода, упрощая добавление новой функциональности.

Недостатки: большое количество конструкций if - else в функциях состояний, что образует вложенность и замедляет работу программы; повторяющийся код, который можно было бы заменить использованием одной функции.

Масштабируемость: можно добавить поддержку альтернативных форматов эр, даты с месяцем и днем. Так же можно добавить поддержку других календарей (например, исламский или еврейски).

Работа выполнена на языке программирования Python в среде разработки PyCharm версии 2023.3.4.

Список литературы

[1] Вострov, А. В. Математическая логика

URL:https://tema.spbstu.ru/compiler (Дата обращения: 23.03.2025).

[2] Сети, Р.; Ахо, А. Компиляторы: принципы, технологии и инструменты / Р. Сети, А. Ахо. - М.: Издательство «Наука», 2006. - С. 104.