Государственное образовательное учреждение высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Рубежный контроль №2

по курсу «Анализ алгоритмов»

Конечные автоматы и регулярные выражения

Выполнил: Сорокин А.П., гр. ИУ7-52Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019 г.

Оглавление

Введение

Цель рубежного контроля: при помощи конечного автомата и регулярного выражения написать программу для поиска критических секций в коде.

1. Аналитическая часть

В данном разделе будут описаны конечные автоматы и регулярные выражения.

1.1 Конечные автоматы

Конечный автомат — это модель вычислений, основанная на гипотетической машине состояний. В один момент времени только одно состояние может быть активным. Следовательно, для выполнения каких-либо действий машина должна менять свое состояние. Конечные автоматы обычно используются для организации и представления потока выполнения чего-либо [1].

Конечный автомат можно представить в виде графа, вершины которого являются состояниями, а ребра — переходы между ними. Каждое ребро имеет метку, информирующую о том, когда должен произойти переход.

1.2 Регулярные выражения

Регулярные выражения — это формальный язык поиска подстроки или подстрок в тексте. Для поиска используется строка-образец (паттерн, или шаблон), состоящая из символов и метасимволов (специальные символы, которые обозначают набор символов).

Это довольно мощный инструмент, который может пригодиться во многих случая — поиск, проверка на корректность строки и т.д. Спектр его возможностей трудно уместить в одну статью [2].

2. Конструкторская часть

В данном разделе будет построены конечный автомат и регулярное выражение для решения поставленной задачи. Шаблон критической секции представлен в листинге 2.1.

Листинг 2.1: Шаблон критической секции

```
1 ::EnterCriticalSection(&obj);
2 // ...
3 ::LeaveCriticalSection(&obj);
```

2.1 Конечный автомат

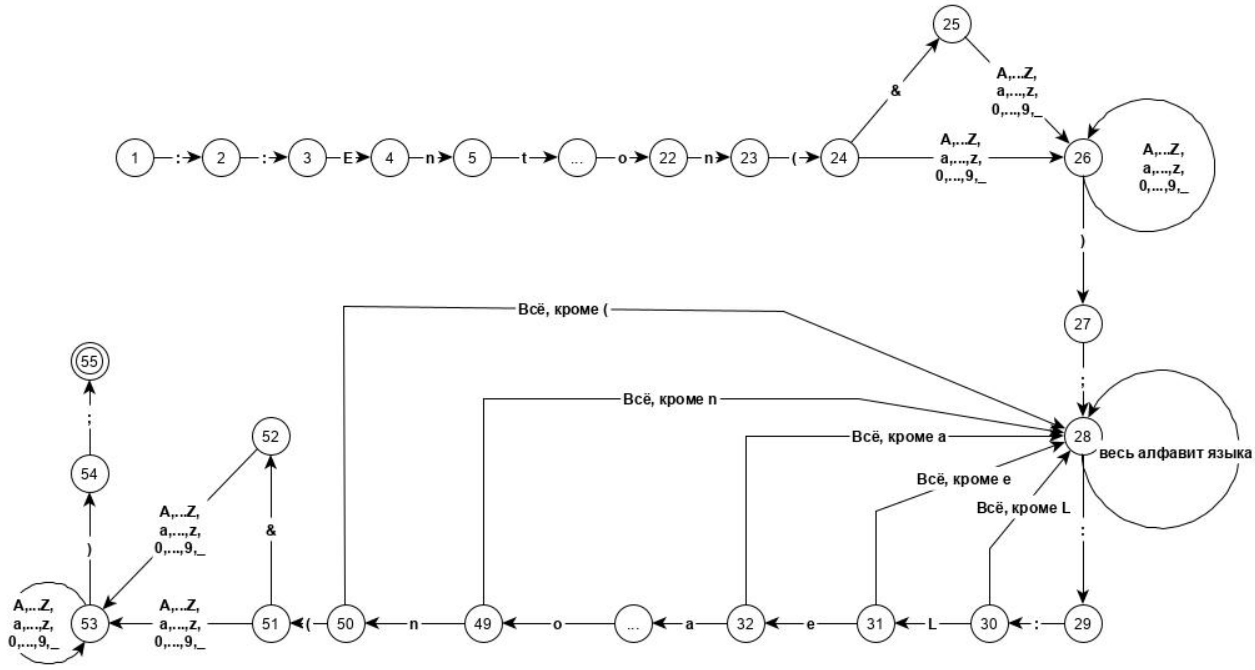

Рис. 2.1: Конечный автомат

2.2 Регулярное выражение

$:: EnterCriticalSection'('(\&|\lambda)(A+...+Z+a+...+z)^{+}')'; (...)^{*} :: LeaveCriticalSection'('(\&|\lambda)(A+...+Z+a+...+z)^{+}')';$

3. Технологическая часть

3.1 Средства реализации

Для реализации программы был использован язык Python [3] (версия интерпретатора 3.7). Для измерения времени была взята функция time.time() из библиотеки time. Данный язык обусловлен тем, что функции, необходимые для реализации регулярного выражения, находятся в встроенной библиотеке re.

3.2 Реализации алгоритмов

На листингах 3.1 и 3.2 представлены коды реализации поиска критических секций с помощью конечных автоматов и с помощью регулярного выражения.

Листинг 3.1: Поиск с помощью конечного автомата

```
1  enter_str = "::EnterCriticalSection"
2  leave_str = "::LeaveCriticalSection"
3
4  name_symbols = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvxyz_"
5  digits = "0123456789"
6
7
8  def find_crit(s):
9    state = 1
10   crit_section = ""
11   result = []
12
13 for value in s:
14   if state == 1:
15     if value == ':':
16       state = 2
17   elif 2 <= state <= 22:
18     if value == enter_str[state - 1]:
19       state += 1
20     else:
21       state = 1
22   elif state == 23:
23     if value == '(':
24       state = 24
25     else:
26       state = 1
27   elif state == 24:
28     if value == '&':
29       state = 25
```

```
    elif value in name_symbols:
      state = 26
    else:
      state = 1
  elif state == 25:
    if value in name_symbols + digits:
      state = 26
    else:
      state = 1
  elif state == 26:
    if value in name_symbols + digits:
      state = 26
    elif value == ')':
      state = 27
    else:
      state = 1
  elif state == 27:
    if value == ';':
      state = 28
    else:
      state = 1

  elif state == 28:
    if value == ':':
      state = 29

  elif 29 <= state <= 49:
    if value == leave_str[state - 28]:
      state += 1
    else:
      state = 28
  elif state == 50:
    if value == '(':
      state = 51
    else:
      state = 28
  elif state == 51:
    if value == '&':
      state = 52
    elif value in name_symbols:
      state = 53
    else:
      state = 28
  elif state == 52:
    if value in name_symbols + digits:
      state = 53
    else:
      state = 28
  elif state == 53:
    if value in name_symbols + digits:
      state = 53
    elif value == ')':
      state = 54
```

```
83      else:
84        state = 28
85    elif state == 54:
86      if value == ';':
87        state = 55
88    else:
89      state = 28
90
91    if state == 1:
92      crit_section = ""
93    else:
94      crit_section += value
95
96    if state == 55:
97      result.append(crit_section)
98      crit_section = ""
99      state = 1
100
101 return result
```

Листинг 3.2: Поиск с помощью регулярного выражения

```
1  regexp = r'::EnterCriticalSection\(([A-Za-z&]\w*)\);' +\
2    r'((?!::EnterCriticalSection\(\1\);)[\s\S])*' +\
3    r'::LeaveCriticalSection\(\1\);'
4
5  def find_crit(text):
6    tmp = text
7
8    found = False
9    match = ""
10
11   while match is not None:
12     match = re.search(regexp, tmp)
13     if match is not None:
14       found = True
15       print(match[0])
16       tmp = tmp[match.end():]
17       print("##############")
18
19   if not found:
20     print("None")
```

4. Экспериментальная часть

4.1 Примеры работы

На рисунке 4.1 представлен пример работы программы.

```
::EnterCriticalSection(&m_obj);
do_work(1);

for (int i = 0; i < 5; i++)
{
    do_work(5);
    do_work(4);
    if (boolvar)
        do_work(7);
    else
        do_work(6);
}
do_work(4);
::LeaveCriticalSection(&m_obj);
############
::EnterCriticalSection(&m_obj2);
do_work(1);
if (boolvar)
    do_work(7);
else
    do_work(6);
do_work(4);
::LeaveCriticalSection(&m_obj3);
############
::EnterCriticalSection(&m_obj2);
    do_work(1);
    if (boolvar)
        do_work(7);
    else
        do_work(6);
    do_work(4);
    ::LeaveCriticalSection(&m_obj2);
############
```

Рис. 4.1: Пример работы программы

4.2 Сравнительный анализ

В таблице 4.1 представлены результаты 20 измерений времени в секундах.

Таблица 4.1: Сравнение работы поиска с помощью конченых автоматов и регулярных выражений

Конечный автомат	Регулярные выражения
0.000221	0.003557
0.000278	0.001659
0.000255	0.001432
0.000256	0.001235
0.000365	0.001505
0.000260	0.001316
0.000248	0.001327
0.000249	0.001309
0.000257	0.001313
0.000277	0.001205
0.000292	0.001314
0.000238	0.001257
0.000241	0.001333
0.000265	0.001335
0.000249	0.001242
0.000247	0.001360
0.000241	0.001252
0.000238	0.001271
0.000268	0.001294
0.000254	0.001292

4.3 Вывод

В среднем поиск с помощью регулярных выражений выполняется в 5 раз дольше, чем поиск с помощью конечного автомата.

Заключение

В ходе выполнения рубежного контроля были реализованы программы поиска критических секций в коде с помощью конечного автомата и регулярного выражения. В результате сравнения было выяснено, что алгоритм поиска с помощью регулярных выражений работает дольше, чем алгоритм поиска с помощью конечного автомата.

Литература

[1] Конечные автоматы (finite-state machine) [Электронный ресурс]. https://habr.com/ru/post/358304/

[2] Регулярные выражения [Электронный ресурс]. https://habr.com/ru/company/badoo/blog/343

[3] Python [Электронный ресурс]. https://www.python.org/