

**LAPORAN TEORI BAHASA DAN AUTOMATA
SIMULATOR DFA, NFA, DAN E-NFA**



Disusun oleh:

- | | |
|------------------------------|------------|
| 1. Adidya Abimanyu | (L0122004) |
| 2. Alfath Roziq Widhayaka | (L0122012) |
| 3. Andrew Gustaya Kristiawan | (L0122020) |
| 4. Ayasha Anggun Anindya | (L0122028) |
| 5. Brama Prameswara Tarigan | (L0122036) |

**PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS DATA
UNIVERSITAS SEBELAS MARET**

2024

BAB I

LATAR BELAKANG

Dalam dunia komputasi, Teori Bahasa dan Automata memegang peranan penting dalam pengembangan dan analisis sistem-sistem komputasi. Teori ini memberikan fondasi matematis untuk memahami berbagai jenis bahasa formal dan model-model komputasi yang digunakan dalam pemrosesan bahasa alami, desain compiler, analisis algoritma, dan banyak lagi.

Salah satu konsep yang paling mendasar dalam Teori Bahasa dan Automata adalah penggunaan otomata untuk merepresentasikan bahasa dan proses komputasi. Otomata dapat digunakan untuk memodelkan perilaku sistem komputasi yang berbeda, mulai dari yang paling sederhana hingga yang paling kompleks. Di antara jenis-jenis otomata yang umum digunakan adalah Finite Automata (FA), Non-deterministic Finite Automata (NFA), Deterministic Finite Automata (DFA), dan Regular Expression (RegEx).

Dalam konteks pengembangan perangkat lunak, seringkali diperlukan alat bantu untuk memahami dan menganalisis bahasa formal serta otomata yang terkait. Oleh karena itu, kami membuat simulator sederhana ini sebagai alat bantu untuk memahami dan menguji konsep-konsep dasar Teori Bahasa dan Automata. Simulator ini memiliki beberapa fitur utama yang dirancang untuk membantu user dalam memahami dan menguji berbagai jenis otomata dan bahasa formal, antara lain:

- 1. Konversi dari NFA atau e-NFA ke DFA**
- 2. Pengkonversian e-NFA dari regular expression**
- 3. Minimisasi DFA**
- 4. Penentuan ekivalensi antara dua DFA**
- 5. Pengujian string pada DFA, NFA, E-NFA, dan RegEx**

BAB II

CARA KERJA PROGRAM MENYELESAIKAN SETIAP MASALAH YANG ADA

1. Nomor 1

a. Input NFA atau e-NFA

```
6 def index():
7     if request.method == 'POST':
8         # Ambil data input dari html
9         states = set(request.form['states'].split(","))
10        alphabet = set(request.form['alphabet'].split(","))
11        transitions = {}
12        transitions_input = request.form.getlist('transitions')
13        for transition in transitions_input:
14            source, symbol, destination = transition.split(",")
15            transitions[(symbol.strip(), source.strip())] = transitions.get((symbol.strip(), source.strip()), set()) | {destination.strip()}
16        start_state = request.form['start_state']
17        accept_states = set(request.form['accept_states'].split(","))
18        # Buat objek NFA dari data input
19        nfa = NFA(states, alphabet, transitions, start_state, accept_states)
20        # Konversi NFA menjadi DFA
21        dfa = nfa.nfa_to_dfa()
22        # Tampilkan tabel transisi DFA
23        dfa_table = dfa.display_transition_table()
24        return render_template('tba.html', dfa_table=dfa_table)
25    return render_template('tba.html', dfa_table=None)
```

Fungsi ini merupakan bagian dari program yang bertujuan untuk mengonversi sebuah automata NFA atau e-NFA menjadi automata DFA. Ketika user mengirimkan data dengan metode POST, program ini akan mengambil data seperti states, alfabet, transisi, state awal, dan accept state. Kemudian, dengan menggunakan data yang dikumpulkan, program akan membuat objek NFA dan mengonversinya menjadi DFA.

b. Representasi NFA dan e-NFA

```
26
27 class NFA:
28     def __init__(self, states, alphabet, transitions, start_state, accept_states):
29         self.states = states
30         self.alphabet = alphabet
31         self.transitions = transitions
32         self.start_state = start_state
33         self.accept_states = accept_states
34
35     def epsilon_closure(self, states):
36         closure = set()
37         stack = list(states)
38         while stack:
39             state = stack.pop()
40             closure.add(state)
41             print(self.transitions)
42             if ('ε', state) in self.transitions:
43                 for next_state in self.transitions[('ε', state)]:
44                     if next_state not in closure:
45                         stack.append(next_state)
46                         closure.add(next_state)
47         print(closure)
48         return frozenset(closure)
49
```

Kelas NFA ini menggambarkan sebuah automata nondeterministik. Ketika sebuah objek NFA dibuat, objek tersebut menyimpan informasi tentang states, alfabet, transisi, state awal, dan accept state. epsilon_closure digunakan untuk menghitung himpunan keadaan yang dapat dicapai dari satu atau lebih keadaan dengan melalui transisi epsilon (ϵ). Metode ini

bekerja dengan mengumpulkan semua keadaan yang dapat dicapai melalui transisi epsilon dari state awal yang diberikan, dengan menggunakan pendekatan depth-first search (DFS). Hasilnya adalah himpunan keadaan yang lengkap yang dapat dicapai dari state awal, termasuk state-state yang dapat dijangkau melalui transisi epsilon.

c. Logika konversi NFA atau e-NFA ke DFA

```
50 def move(self, states, symbol):
51     reachable_states = set()
52     for state in states:
53         if (symbol, state) in self.transitions:
54             reachable_states.update(self.transitions[(symbol, state)])
55     return frozenset(reachable_states)
56
57 def nfa_to_dfa(self):
58     dfa_states = set()
59     dfa_transitions = {}
60     dfa_start_state = self.epsilon_closure({self.start_state})
61     dfa_accept_states = set()
62     unmarked_states = [dfa_start_state]
63
64     while unmarked_states:
65         current_state_set = unmarked_states.pop(0)
66         dfa_states.add(current_state_set)
67
68         for symbol in self.alphabet:
69             next_state = self.move(current_state_set, symbol)
70             epsilon_closure_next = self.epsilon_closure(next_state)
71
72             if epsilon_closure_next:
73                 dfa_transitions[(current_state_set, symbol)] = epsilon_closure_next
74
75                 if epsilon_closure_next not in dfa_states:
76                     unmarked_states.append(epsilon_closure_next)
77
78             if any(state in self.accept_states for state in current_state_set):
79                 dfa_accept_states.add(current_state_set)
80
81     return DFA(dfa_states, self.alphabet, dfa_transitions, dfa_start_state, dfa_accept_states)
82
```

Metode `nfa_to_dfa` dalam kelas NFA bertujuan untuk mengkonversi sebuah automata NFA menjadi automata DFA. Prosesnya dimulai dengan menginisialisasi state, transisi, dan state awal dari DFA. Selanjutnya, state awal DFA dihitung dengan mengambil epsilon-closure dari state awal NFA. Kemudian, proses iteratif dimulai dengan mengunjungi setiap state yang belum ditandai dalam DFA. Untuk setiap state, transisi DFA diperbarui berdasarkan transisi NFA yang sesuai dengan simbol alfabet. Jika transisi baru memiliki keadaan yang belum ditandai, state tersebut ditambahkan ke daftar keadaan yang belum ditandai. Selama proses ini, state yang merupakan accept state dalam DFA ditambahkan ke himpunan state akhir DFA. Setelah semua keadaan NFA ditelusuri, DFA yang dihasilkan dikembalikan sebagai objek DFA baru. Dengan demikian, metode ini membantu dalam menghasilkan representasi deterministik dari otomata nondeterministik.

d. Representasi DFA

```
83 class DFA:
84     def __init__(self, states, alphabet, transitions, start_state, accept_states):
85         self.states = states
86         self.alphabet = alphabet
87         self.transitions = transitions
88         self.start_state = start_state
89         self.accept_states = accept_states
90
```

Kelas DFA ini menggambarkan sebuah automata deterministik. Saat membuat objek DFA, states, alfabet, transisi, state awal, dan accept_states disimpan dalam objek tersebut. Dengan menggunakan kelas DFA, hasilnya dapat memodelkan dan mengoperasikan automata yang sesuai dengan aturan deterministik.

e. Output DFA

```
90
91 def display_transition_table(self):
92     # Mengurutkan simbol alphabet sesuai input pengguna
93     sorted_alphabet = sorted(self.alphabet)
94
95     # Pisahkan initial state dan final state
96     initial_state = self.start_state
97     final_states = sorted(self.accept_states - {self.start_state})
98
99     # Sort state untuk menempatkan initial state di awal dan final state di akhir
100     sorted_states = [initial_state] + sorted(self.states - {initial_state} - self.accept_states) + final_states
101
102     dfa_table = []
103     dfa_table.append("Tabel Transisi DFA:")
104     dfa_table.append("-----")
105     dfa_table.append("| Keadaan | " + " | ".join(sorted_alphabet) + " |")
106     dfa_table.append("-----")
107
108     max_state_length = max(len(state) for state in sorted_states)
109     for state in sorted_states:
110         row = "|"
111         if state == initial_state:
112             row += " -> "
113         elif state in final_states:
114             row += " * "
115         else:
116             row += " "
117
118         state_str = ', '.join(sorted(state))
119         row += f"{state_str} ".ljust(max_state_length + 6) + "|"
120         for symbol in sorted_alphabet:
121             next_state = self.transitions.get((state, symbol), set())
122
123             next_state_str = ', '.join(sorted(next_state))
124             row += f"{next_state_str} ".ljust(len(symbol) + 6) + "|"
125         dfa_table.append(row)
126     dfa_table.append("-----")
127     return '\n'.join(dfa_table)
128
```

Metode `display_transition_table` digunakan untuk membuat tabel transisi yang menampilkan bagaimana DFA berpindah ke keadaan lain berdasarkan simbol dalam alfabet. Proses dimulai dengan mengurutkan alfabet untuk menampilkan transisi sesuai dengan urutan input user. Kemudian, state awal dan accept state dipisahkan untuk memudahkan penyesuaian tampilan tabel. Selanjutnya, state-state diurutkan untuk menempatkan state awal di awal dan accept state di akhir. Tabel tersebut kemudian dibuat dengan menyusun baris-baris yang mewakili setiap keadaan dan transisi yang terkait. State awal ditandai dengan tanda panah (->) dan accept state ditandai dengan bintang (*).

2. Nomor 2

a. Menginputkan Regular Expression

```
1 @app.route('/')
2 def index():
3     return render_template('index.html')
4
5 @app.route('/submit', methods=['POST'])
6 def submit():
7     postfix_exp = request.form['regex']
8     pr = postfix(postfix_exp)
9     et = constructTree(pr)
10    fa = evalRegex(et)
11
12    transitions = []
13    symbol_table = {fa[0]: 0}
14    printStateTransitions(fa[0], [], symbol_table)
15
16    for state in symbol_table:
17        current_state = "q" + str(symbol_table[state])
18        for symbol in state.next_state:
19            next_states = state.next_state[symbol]
20            for next_state in next_states:
21                next_state_index = "q" + str(symbol_table[next_state])
22                transitions.append({'state': current_state, 'symbol': symbol, 'next_state': next_state_index})
23
24    return render_template('index.html', pr=pr, transitions=transitions)
```

Pada saat user menginputkan regex pada web, input dari user yang berbentuk regular expression type data string akan diterima melalui web yang ada pada Flask. fungsi 'index():' yang digunakan untuk memanggil file index.html. Fungsi 'submit():' ini akan dipanggil ketika user menginput regex pada web dengan ('/submit', methods=['POST']).

b. Convert ke Postfix

```
1 def higherPrecedence(a, b):
2     p = ["+", "-", "*", "/"]
3     return p.index(a) > p.index(b)
4
5 def postfix(regexp):
6     temp = []
7     for i in range(len(regexp)):
8         if i != 0 and (regexp[i-1].isalpha() or regexp[i-1] == "(" or regexp[i-1] == "(") and (regexp[i].isalpha() or regexp[i] == "("):
9             temp.append(".")
10            temp.append(regexp[i])
11            regexp = temp
12
13            stack = []
14            output = ""
15
16            for c in regexp:
17                if c.isalpha():
18                    output = output + c
19                    continue
20
21                if c == "(":
22                    while len(stack) != 0 and stack[-1] != "(":
23                        output = output + stack.pop()
24                    stack.pop()
25                elif c == "(":
26                    stack.append(c)
27                elif c == "*":
28                    output = output + c
29                elif len(stack) == 0 or stack[-1] == "(" or higherPrecedence(c, stack[-1]):
30                    stack.append(c)
31                else:
32                    while len(stack) != 0 and stack[-1] != "(" and not higherPrecedence(c, stack[-1]):
33                        output = output + stack.pop()
34                    stack.append(c)
35
36            while len(stack) != 0:
37                output = output + stack.pop()
38
39            return output
```

Setelah diterima, input dari akan di convert terlebih dulu ke postfix. Postfix itu sendiri contohnya seperti dari $(a+b)*$ menjadi $ab+*$. Alasan mengubah nya menjadi postfix terlebih dulu adalah ini akan memudahkan komputer untuk melakukan komputasi. Pada 2 fungsi ini menggunakan stack dan aturan prioritas. Contoh pada saat menemukan tanda kurung buka '(', karakter tersebut dimasukkan ke dalam stack, menandakan awal dari suatu grup operasi. Ketika menemukan tanda kurung tutup), operator-operator yang ada dalam stack diproses hingga ditemukan tanda kurung buka yang sesuai, sehingga urutan operasi dalam tanda kurung dapat dijaga dengan benar. Operasi-operator seperti '*', '!', '+' ditangani dengan membandingkan prioritasnya dengan operator-operator dalam stack yang lain.

c. Convert ke ExpressionTree

```
1 class Type:
2     SYMBOL = 1
3     CONCAT = 2
4     UNION = 3
5     KLEENE = 4
6
7 class ExpressionTree:
8     def __init__(self, _type, value=None):
9         self._type = _type
10        self.value = value
11        self.left = None
12        self.right = None
13
14    def constructTree(regex):
15        stack = []
16        for c in regex:
17            if c.isalpha():
18                stack.append(ExpressionTree(Type.SYMBOL, c))
19            else:
20                if c == "+":
21                    z = ExpressionTree(Type.UNION)
22                    z.right = stack.pop()
23                    z.left = stack.pop()
24                elif c == ".":
25                    z = ExpressionTree(Type.CONCAT)
26                    z.right = stack.pop()
27                    z.left = stack.pop()
28                elif c == "*":
29                    z = ExpressionTree(Type.KLEENE)
30                    z.left = stack.pop()
31                    stack.append(z)
32        return stack[0]
33
34    def inorder(et):
35        if et._type == Type.SYMBOL:
36            return et.value
37        elif et._type == Type.CONCAT:
38            return inorder(et.left) + "." + inorder(et.right)
39        elif et._type == Type.UNION:
40            return inorder(et.left) + "+" + inorder(et.right)
41        elif et._type == Type.KLEENE:
42            return inorder(et.left) + "*"
43
```

Setelah diconvert menjadi postfix, maka akan diconvert menjadi expression tree. Kegunaan tree ini adalah untuk agar tetap terstruktur termasuk operator dan operand, dan sebagainya. Pada fungsi 'constructTree (regex):' akan melakukan pengconversion dari postfix ke expression tree. Jika operator karakter atau menerima seperti ('.', '+', atau '*'), maka operasi spesifik dilakukan tergantung pada jenis operator tersebut. Jika karakter atau menerima operator '.', maka pop dua node dari stack sebagai child kiri dan kanan. Kemudian, node baru dengan tipe 'Concat' dan child kiri dan kanannya sesuai dengan hasil pop, lalu masukkan node tersebut ke dalam stack. Jika menerima operator '+', maka pop dua node dari stack sebagai child kiri dan kanan. Kemudian, buat node baru dengan tipe 'Union' dan child kiri dan kanannya sesuai dengan hasil pop, lalu masukkan node tersebut ke dalam stack. Jika menerima operator '*', maka pop satu node dari stack sebagai child. Kemudian, buat node baru dengan tipe Kleene dan child nya sesuai dengan hasil pop, lalu masukkan node tersebut ke dalam stack. Jika karakter adalah simbol lain, maka buat node baru dengan tipe Symbol dan nilai sesuai dengan karakter tersebut, lalu masukkan node tersebut ke dalam stack.

Fungsi 'inorder(et)' digunakan untuk penelusuran. Yang dimaksud adalah jika tipe node adalah SYMBOL, maka nilai dari node tersebut dikembalikan. Jika tipe node adalah CONCAT, UNION, atau KLEENE, maka fungsi akan melakukan penelusuran inorder secara rekursif ke anak kiri dan kanan, dan mengembalikan string yang merupakan regex dari node tersebut dan anak-anaknya.

d. Convert ke E-NFA

```

1  class FiniteAutomataState:
2      def __init__(self):
3          self.next_state = {}
4
5  def evalRegex(et):
6      if et._type == Type.SYMBOL:
7          return evalRegexSymbol(et)
8      elif et._type == Type.CONCAT:
9          return evalRegexConcat(et)
10     elif et._type == Type.UNION:
11         return evalRegexUnion(et)
12     elif et._type == Type.KLEENE:
13         return evalRegexKleene(et)
14
15 def evalRegexSymbol(et):
16     start_state = FiniteAutomataState()
17     end_state = FiniteAutomataState()
18
19     start_state.next_state[et.value] = [end_state]
20     return start_state, end_state
21
22 def evalRegexConcat(et):
23     left_nfa = evalRegex(et.left)
24     right_nfa = evalRegex(et.right)
25
26     left_nfa[1].next_state['epsilon'] = [right_nfa[0]]
27     return left_nfa[0], right_nfa[1]
28
29 def evalRegexUnion(et):
30     start_state = FiniteAutomataState()
31     end_state = FiniteAutomataState()
32
33     up_nfa = evalRegex(et.left)
34     down_nfa = evalRegex(et.right)
35
36     start_state.next_state['epsilon'] = [up_nfa[0], down_nfa[0]]
37     up_nfa[1].next_state['epsilon'] = [end_state]
38     down_nfa[1].next_state['epsilon'] = [end_state]
39
40     return start_state, end_state
41
42 def evalRegexKleene(et):
43     start_state = FiniteAutomataState()
44     end_state = FiniteAutomataState()
45
46     sub_nfa = evalRegex(et.left)
47
48     start_state.next_state['epsilon'] = [sub_nfa[0], end_state]
49     sub_nfa[1].next_state['epsilon'] = [sub_nfa[0], end_state]
50
51     return start_state, end_state

```

Setelah menjadi expression tree, diconvert menjadi e-NFA. Setiap node dalam pohon ekspresi ('ExpressionTree') mewakili sebuah operasi atau simbol dalam ekspresi reguler. Fungsi 'evalRegex' memulai konversi dari pohon ekspresi menjadi e-NFA dengan menginisialisasi NFA dan menetapkan state awal dan akhir berdasarkan hasil dari pemrosesan pohon ekspresi. Bergantung pada jenis node ('_type'), fungsi ini memanggil fungsi yang sesuai untuk mengevaluasi bagian pohon ekspresi. Fungsi-fungsi 'evalRegexSymbol', 'evalRegexConcat', 'evalRegexUnion', dan 'evalRegexKleene' digunakan untuk memproses setiap node dalam pohon, tergantung pada jenis karakternya (seperti concat, union, kleene(closure), atau symbol). Masing-masing fungsi ini mengimplementasikan logika konversi untuk jenis karakter yang sesuai, dengan menambahkan state dan transisi yang diperlukan ke dalam NFA.

Transisi epsilon juga ditambahkan sesuai aturan NFA yang memungkinkan perpindahan tanpa konsumsi simbol input.

e. Mencetak hasil

```
1 def printStateTransitions(state, states_done, symbol_table):
2     if state in states_done:
3         return
4
5     states_done.append(state)
6
7     for symbol in list(state.next_state):
8         current_state_index = symbol_table[state]
9         next_states = state.next_state[symbol]
10
11         for next_state in next_states:
12             if next_state not in symbol_table:
13                 symbol_table[next_state] = len(symbol_table)
14                 next_state_index = "q" + str(symbol_table[next_state])
15
16         for next_state in next_states:
17             printStateTransitions(next_state, states_done, symbol_table)
```

Fungsi 'printStateTransitions' digunakan untuk mencetak transisi dari setiap state dalam finite automata. Ini adalah rekursif, menerima tiga parameter: state (state yang sedang diproses), states_done (daftar state yang sudah diproses untuk menghindari siklus), dan symbol_table (tabel yang digunakan untuk mengindeks state).

3. Nomor 3

Minimalisasi DFA adalah proses mengurangi jumlah state dalam sebuah DFA tanpa mengubah bahasa yang diterima oleh DFA tersebut. Tujuan dari minimalisasi DFA adalah untuk menciptakan DFA yang lebih sederhana dan efisien dalam memproses string-string input, dengan tetap mempertahankan sifat kekekalan terhadap bahasa yang diterima. Berikut merupakan cara kerja program dalam menyelesaikan setiap permasalahan yang ada :

a. Inisialisasi DFA, Transisi, dan Ekuivalen

```
1 # Membuat kelas untuk menginisialisasi objek DFA dengan masing-masing parameter
2 class DFA:
3     def __init__(self, states, addSimbol, transitions, state_awal, state_final):
4         self.states = states
5         self.addSimbol = addSimbol
6         self.transitions = transitions
7         self.state_awal = state_awal
8         self.state_final = state_final
9
10 # Fungsi pada state untuk melakukan transisi dari state saat ini dengan simbol input yang diberikan
11 def nextState(stateCurrent, simbol, transitions):
12     return transitions.get(stateCurrent, {}).get(simbol)
13
14 # Fungsi pada DFA untuk memeriksa apakah state tersebut ekuivalen atau tidak
15 def stateEkuivalen(state1, state2, classEkuivalen):
16     return classEkuivalen[state1][state2]
```

Cara kerja program yang pertama adalah program dimulai dengan mendefinisikan kelas DFA yang pada masing-masing parameter seperti **states**, simbol (**addSimbol**), **transitions**, **state_awal**, dan **state_final**. Selain itu, program juga mendefinisikan fungsi-fungsi yang berkaitan dengan operasi-operasi DFA seperti dibawah ini :

- **Fungsi nextState** yang berperan penting dalam proses pemrosesan transisi antar state dalam DFA. Dengan menerima input berupa state saat ini **stateCurrent**, **simbol**, dan informasi tentang transisi-transisi antar state dalam bentuk dictionary, fungsi ini memberikan kemampuan untuk mencari dan mengembalikan state tujuan setelah melakukan transisi dari **stateCurrent** dengan simbol input yang diberikan.
- **Fungsi stateEkuivalen** untuk pengecekan ekuivalensi antara dua state dalam DFA dengan menerima input berupa dua state (**state1** dan **state2**) yang hendak dibandingkan, serta struktur data classEkuivalen yang merepresentasikan matriks kebenaran dari ekuivalensi antar state dalam DFA, fungsi ini mengembalikan nilai kebenaran yang menunjukkan apakah kedua state tersebut ekuivalen atau tidak. Melalui fungsi ini, simulator dapat memvalidasi kelas ekuivalen antar state dalam proses minimalisasi DFA, memastikan bahwa setiap pasangan state diproses dengan benar untuk mempertahankan bahasa yang sama yang diterima oleh DFA sebelum dan setelah minimalisasi.

b. Pengecekan Input String User

```
1 # Fungsi untuk memeriksa DFA menerima string yang diinputkan user pada DFA sebelum
2 def inputanString(dfa, input_string):
3     stateCurrent = dfa.state_awal
4     for simbol in input_string:
5         stateLanjutan = nextState(stateCurrent, simbol, dfa.transitions)
6         if stateLanjutan is None:
7             return False
8         stateCurrent = stateLanjutan
9     return stateCurrent in dfa.state_final
```

Fungsi inputanString berfungsi untuk menguji apakah DFA menerima string input yang diberikan oleh user atau tidak, baik pada DFA sebelum atau setelah minimalisasi. Pertama fungsi ini dimulai dari pengecekan state awal **dfa.state_awal** dan setiap simbol dalam **input_string** dengan menggunakan fungsi **nextState** untuk melakukan transisi antar state berdasarkan simbol yang diinputkan user. Jika tidak ada transisi yang ditemukan untuk salah satu simbol, maka fungsi ini mengembalikan False, menandakan bahwa DFA tidak menerima string input tersebut. Namun, jika proses berlanjut hingga selesai membaca seluruh simbol dalam string, dan DFA berakhir di salah satu state final **dfa.state_final**, maka fungsi ini mengembalikan True, menandakan bahwa DFA menerima string input tersebut.

c. Menghapus State yang Tidak Berguna

```
1 # Fungsi untuk menghapus state-state yang tidak berguna
2 def hapusStateUnreach(objek, start, getIn=None):
3     if getIn is None:
4         getIn = set()
5         getIn.add(start)
6
7     for sideState in objek.transitions[start]:
8         if objek.transitions[start][sideState] not in getIn:
9             hapusStateUnreach(objek, objek.transitions[start][sideState], getIn)
10
11     return list(sorted(getIn))
```

Fungsi **hapusStateUnreach** berfungsi untuk menghapus state-state yang tidak berguna dalam DFA. Fungsi ini melakukan pencarian rekursif melalui **for sideState in objek.transitions[start]** untuk mencapai semua state yang dapat dicapai dari state awal. Ketika mencapai state baru, fungsi akan menambahkannya ke dalam **getIn** untuk menandai bahwa state tersebut telah dikunjungi. Proses ini terus berlanjut hingga tidak ada state baru yang dapat dicapai, dan pada akhirnya, fungsi mengembalikan daftar state-state yang telah dikunjungi secara terurut. Dengan demikian, **fungsi hapusStateUnreach** memungkinkan penghapusan state-state yang tidak dapat dicapai dari state awal.

d. Melakukan Proses Minimalisasi DFA (Langkah 1 – 3)

```
1 # Fungsi untuk melakukan minimalisasi DFA
2 def minimalisasiDFA(dfa):
3     # Langkah 1 menghapus state-state yang tidak berguna dari DFA yang diinputkan user
4     stateReach = hapusStateUnreach(dfa, dfa.state_awal)
5
6     # Langkah 2 melakukan Inisialisasi kelas ekuivalen untuk setiap pasangan state
7     classEkuivalen = {}
8     for state1 in stateReach:
9         classEkuivalen[state1] = {}
10        for state2 in stateReach:
11            classEkuivalen[state1][state2] = (state1 in dfa.state_final) == (state2 in dfa.state_final)
12
13    # Langkah 3 memeriksa kelas ekuivalen untuk setiap pasangan state berdasarkan transisi yang diberikan user
14    for state1 in stateReach:
15        for state2 in stateReach:
16            for simbol in sorted(dfa.addSimbol):
17                stateAfter1 = nextState(state1, simbol, dfa.transitions)
18                stateAfter2 = nextState(state2, simbol, dfa.transitions)
19                if (stateAfter1 is None and stateAfter2 is not None) or (stateAfter1 is not None and stateAfter2 is None):
20                    classEkuivalen[state1][state2] = False
```

Fungsi **minimalisasiDFA** memiliki fungsi untuk melakukan proses minimalisasi DFA dengan langkah-langkah pada kode.

- Langkah 1

Pertama menggunakan **fungsi hapusStateUnreach** untuk mengidentifikasi state-state yang tidak berguna dalam DFA. Fungsi ini menelusuri seluruh transisi dari state awal DFA dan menandai semua state yang dapat dicapai dari state awal. State-state yang tidak dapat dicapai dari state awal akan dihapus.

- Langkah 2

Setelah memiliki kumpulan state-state yang dapat dicapai dari state awal, maka inisialisasi kelas ekuivalen untuk setiap pasangan state dalam DFA. Ini dilakukan

dengan membuat **matriks classEkuivalen** yang menyimpan informasi tentang apakah setiap pasangan state adalah ekuivalen atau tidak.

- Langkah 3

Selanjutnya melakukan iterasi untuk setiap pasangan state dalam **stateReach**, Selanjutnya, melakukan iterasi untuk setiap simbol input dalam urutan yang sudah disortir dari **dfa.addSimbol**. Di dalam iterasi ini didapatkan state setelah transisi dari state1 dan state2 berdasarkan simbol input yang sedang diproses. Jika salah satu dari kedua state tersebut tidak memiliki transisi sedangkan yang lainnya memiliki transisi, maka kedua state tersebut dinyatakan tidak ekuivalen dan nilai False ditetapkan dalam **matriks classEkuivalen** untuk pasangan state tersebut.

e. Melakukan Proses Minimalisasi DFA (Langkah 4 – 5)

```
1 # Langkah 4 melakukan iterasi untuk memperbarui kelas ekuivalen hingga tidak ada perubahan
2 while True:
3     ubahState = False
4     for state1 in stateReach:
5         for state2 in stateReach:
6             if not classEkuivalen[state1][state2]:
7                 continue
8             for simbol in sorted(dfa.addSimbol):
9                 stateAfter1 = nextState(state1, simbol, dfa.transitions)
10                stateAfter2 = nextState(state2, simbol, dfa.transitions)
11                if not stateEkuivalen(stateAfter1, stateAfter2, classEkuivalen):
12                    classEkuivalen[state1][state2] = False
13                    ubahState = True
14                    break
15            if ubahState:
16                break
17        if ubahState:
18            break
19    if not ubahState:
20        break
21
22 # Langkah 5 mengelompokkan state-state ekuivalen menjadi satu grup DFA yang sudah
23 groupEkuivalen = {}
24 groupAddState = 0
25 for state1 in stateReach:
26     if state1 not in groupEkuivalen.keys():
27         groupAddState += 1
28         groupEkuivalen[state1] = state1
29     for state2 in stateReach:
30         if state1 != state2 and classEkuivalen[state1][state2]:
31             groupEkuivalen[state2] = groupEkuivalen[state1]
```

- Langkah 4

Pada langkah ini berfungsi untuk memperbarui kelas ekuivalen hingga mencapai kestabilan. Iterasi dilakukan menggunakan **loop while True**, dimana setiap iterasi memeriksa setiap pasangan state yang sebelumnya dinyatakan ekuivalen namun kemudian ditemukan tidak ekuivalen berdasarkan transisi yang diberikan dalam DFA. Dalam iterasi tersebut, dilakukan nested loop untuk mengiterasi setiap pasangan state dalam **stateReach**, kemudian mengiterasi setiap simbol input dari **dfa.addSimbol**. Setelah itu, memeriksa transisi dari kedua state setelah simbol input diterapkan, dan jika terdapat perbedaan dalam transisi antara kedua state, maka kelas ekuivalen dari pasangan state tersebut diubah menjadi False. Jika terjadi perubahan dalam iterasi tersebut, variabel **ubahState** akan diatur menjadi True. Iterasi berlanjut hingga tidak ada lagi perubahan yang terdeteksi, menandakan bahwa telah mencapai kelas ekuivalen yang stabil.

- Langkah 5

Setelah kelas ekuivalen stabil, langkah selanjutnya adalah mengelompokkan state-state ekuivalen ke dalam satu grup DFA yang sudah diminimalisasi. Dalam langkah ini yaitu membuat dictionary **groupEkuivalen** yang akan menyimpan informasi tentang grup-grup state ekuivalen yang baru. Setiap state dalam **stateReach** akan dihubungkan dengan grup ekuivalen yang sesuai, sehingga membentuk grup-grup DFA yang sudah diminimalisasi.

f. Melakukan Proses Minimalisasi DFA (Langkah 6 – 8)

```

1 # Langkah 6 memberi label/parameter baru untuk setiap state dalam grup ekuivalen yang b
2 stateBaru = {}
3 for state in stateReach:
4     stateBaru[state] = str(groupEkuivalen[state])
5
6 # Langkah 7 memperbarui state, state final, dan transisi sesuai dengan DFA yang
7 hasilStateBaru = []
8 stateFinalBaru = []
9 transisiBaru = []
10 for state in stateReach:
11     hasilStateBaru.append(stateBaru[state])
12     if state in dfa.state_final:
13         stateFinalBaru.append(stateBaru[state])
14
15
16     for state in stateReach:
17         for simbol in sorted(dfa.addSimbol):
18             stateLanjutan = dfa.transitions[state].get(simbol)
19             if stateLanjutan:
20                 stateLanjutanBaru = stateBaru[stateLanjutan]
21                 transisi = (stateBaru[state], simbol, stateLanjutanBaru)
22                 transisiBaru.append(transisi)
23
24 # Langkah 8 mengubah format transisi menjadi sesuai dengan DFA yang diminimalis
25 transisiModifikasi = {}
26 for transisi in transisiBaru:
27     stateStart, simbol, stateLanjutan = transisi
28     if stateStart not in transisiModifikasi:
29         transisiModifikasi[stateStart] = {}
30     transisiModifikasi[stateStart][simbol] = stateLanjutan

```

- Langkah 6

Selanjutnya memberikan label baru untuk setiap state dalam grup ekuivalen yang baru. Hal ini dilakukan dengan membuat sebuah dictionary baru bernama **stateBaru**. Dictionary ini akan menyimpan informasi label baru untuk setiap state dalam **stateReach**. Prosesnya adalah iterasi melalui setiap state dalam **stateReach**, dan menghubungkan setiap state dengan label yang sesuai dengan grup ekuivalennya dalam **groupEkuivalen**.

- Langkah 7

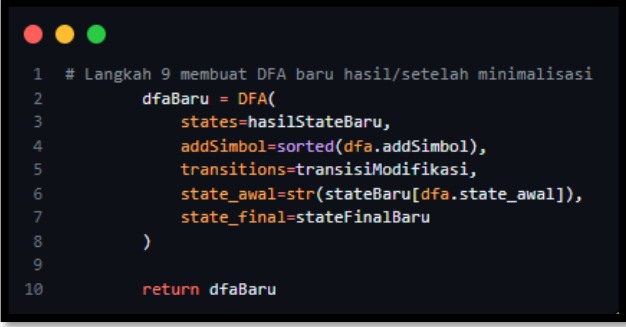
Setelah label baru diberikan pada setiap state, langkah selanjutnya adalah memperbarui informasi tentang state, state final, dan transisi sesuai dengan DFA yang telah diminimalisasi. Dalam langkah ini, dibuat list baru **hasilStateBaru** untuk menyimpan label baru dari setiap state dalam **stateReach**. State final juga diperbarui dengan mengganti state-state final dengan label baru yang sesuai dari **stateBaru**. Selanjutnya, melakukan iterasi kembali melalui setiap state dalam **stateReach** dan setiap simbol input dari **dfa.addSimbol**. Transisi antar state juga

diperbarui dengan menggunakan label baru dari **stateBaru**. Informasi transisi baru ini disimpan dalam list **transisiBaru**.

- Langkah 8

Selanjutnya melakukan iterasi kembali melalui setiap **stateReach** dan mengubah format transisi menjadi DFA yang sesuai diminimalisasi. **transisiModifikasi** akan menyimpan informasi transisi baru antar state.

g. Melakukan Proses Minimalisasi DFA (Langkah 9)



```
1 # Langkah 9 membuat DFA baru hasil/setelah minimalisasi
2 dfaBaru = DFA(
3     states=hasilStateBaru,
4     addSimbol=sorted(dfa.addSimbol),
5     transitions=transisiModifikasi,
6     state_awal=str(stateBaru[dfa.state_awal]),
7     state_final=stateFinalBaru
8 )
9
10 return dfaBaru
```

Langkah terakhir membuat DFA baru yang merupakan hasil dari minimalisasi. Pada langkah ini, sebuah objek DFA baru dibuat dengan menggunakan parameter-parameter baru yang telah diperbarui sebelumnya. Objek DFA baru ini akan memiliki state baru, simbol input yang sama dengan DFA sebelumnya, transisi baru sesuai dengan DFA yang telah diminimalisasi, state awal baru, dan state final baru.

Parameter **states** diambil dari **hasilStateBaru**, yang berisi label baru untuk setiap state dalam **stateReach**. Parameter **addSimbol** tetap sama seperti DFA sebelumnya. Parameter **transitions** diambil dari **transisiModifikasi**, yang berisi informasi transisi baru antar state berdasarkan label baru dari **stateBaru**. Parameter **state_awal** adalah label baru dari state awal DFA sebelumnya, yang diambil dari **stateBaru**. Parameter **state_final** diambil dari **stateFinalBaru**, yang berisi label baru untuk setiap state final dalam **stateReach**.

h. Membuat representasi DFA menggunakan Graphviz

```
1 # Fungsi untuk membuat grafik / representasi DFA menggunakan DOT lan
2 def cetakDFA(dfa):
3     # Membuat objek Digraph dari grapviz
4     dot = Digraph()
5     dot.attr(rankdir='LR')
6
7     # Menambahkan node untuk setiap state dalam DFA
8     for state in dfa.states:
9         if state in dfa.state_final:
10             dot.node(state, shape='doublecircle') # State Final
11         else:
12             dot.node(state)
13
14     # Menambahkan transisi antar state dalam DFA
15     for start_state, transitions in dfa.transitions.items():
16         for simbol, stateLanjutan in transitions.items():
17             dot.edge(start_state, stateLanjutan, label=simbol)
18
19     # Menambahkan tanda state awal dengan label start
20     dot.attr('node', shape='none', label='start')
21     dot.node('')
22     dot.edge('', dfa.state_awal)
23
24     return dot.pipe(format='svg').decode('utf-8')
```

Fungsi **cetakDFA** bertujuan untuk membuat representasi grafis dari DFA menggunakan DOT language, yang kemudian akan diubah menjadi format SVG untuk ditampilkan di web sebagai gambar vektor. Fungsi ini menggunakan library Graphviz untuk membuat objek Digraph dan mengatur arah tata letak grafik agar horizontal dari kiri ke kanan (`rankdir='LR'`). Selanjutnya, fungsi ini menambahkan node untuk setiap state dalam DFA, dengan state final diberi bentuk lingkaran ganda. Kemudian, transisi antar state ditambahkan sebagai transisi antara state, dengan label sesuai simbol transisi. Terakhir, sebuah node kosong ditambahkan sebagai tanda state awal dengan label **start**, dan transisi dibuat dari node kosong tersebut ke state awal DFA.

i. Membuat DFA melalui input form user yang ada di HTML dan membuat objek DFA dari data yang dimasukkan user

```
1 # Kode untuk membuat DFA dari input form yang ada di HTML menggunakan data yang dim
2 # Input form user ini nantinya akan disimpan dalam masing-masing dfa
3 dfa = {}
4 dfa['states'] = request.form['states'].split()
5 dfa['addSimbol'] = request.form['inputSimbol'].split()
6 dfa['state_awal'] = request.form['stateAwal']
7 dfa['state_final'] = request.form['stateFinal'].split()
8 dfa['transitions'] = {}
9 for state in dfa['states']:
10     dfa['transitions'][state] = {}
11     for simbol in dfa['addSimbol']:
12         stateLanjutan = request.form.get(f'transitions_{state}_{simbol}')
13         dfa['transitions'][state][simbol] = stateLanjutan
14
15 # Kode untuk membuat objek DFA dari data yang dimasukkan user pada input form
16 inputDFA = DFA(
17     states=set(dfa['states']),
18     addSimbol=set(dfa['addSimbol']),
19     transitions=dfa['transitions'],
20     state_awal=dfa['state_awal'],
21     state_final=set(dfa['state_final'])
22 )
23
```

Kode diatas berfungsi untuk membuat DFA berdasarkan data yang dimasukkan oleh user melalui formulir HTML. Data dari formulir HTML disimpan dalam **dictionary** **dfa**, yang terdiri dari **states**, **addSimbol**, **state_awal**, **state_final** dan **transitions**.

Selanjutnya, objek DFA baru atau **inputDFA** dibuat menggunakan data yang telah dikumpulkan dari formulir HTML, dengan menggunakan konstruktor DFA dan parameter yang sesuai seperti **states**, **addSimbol**, **transitions**, **state_awal**, dan **state_final**.

- j. Menampilkan hasil sebelum minimalisasi dan sesudah minimalisasi DFA

```
1 # Uji DFA sebelum minimalisasi
2 input_string = request.form['inputString']
3 hasilStringSebelum = "DFA menerima string yang diuji" if inputString(inputDFA, input_string) else "DFA tidak menerima string yang diuji"
4
5 # Minimalisasi DFA
6 minimalDFA = minimalisasiDFA(inputDFA)
7
8 # Uji DFA setelah minimalisasi
9 hasilStringSesudah = "DFA menerima string yang diuji" if inputString(minimalDFA, input_string) else "DFA tidak menerima string yang diuji"
10
11 # Ambil nilai yang ingin ditampilkan
12 hasilState = ", ".join(minimalDFA.states)
13 hasilSimbol = ", ".join(minimalDFA.addSimbol)
14 hasilTransisi = []
15 for start_state, transitions in minimalDFA.transitions.items():
16     for simbol, statelanjutan in transitions.items():
17         hasilTransisi.append([start_state, simbol, statelanjutan])
18 hasilStateAwal = minimalDFA.state_awal
19 hasilStateFinal = ", ".join(minimalDFA.state_final)
20
21 # Kode untuk membuat grafik / representasi DFA sebelum dan sesudah minimalisasi
22 graphSebelumMinim = cetakDFA(inputDFA)
23 graphSesudahMinim = cetakDFA(minimalDFA)
24
25 # Kode untuk mengembalikan hasil DFA minimalisasi ke halaman HTML untuk ditampilkan
26 return render_template('result3.html', hasilState=hasilState, hasilSimbol=hasilSimbol,
27     hasilTransisi=hasilTransisi, hasilStateAwal=hasilStateAwal,
28     hasilStateFinal=hasilStateFinal, hasilStringSebelum=hasilStringSebelum,
29     hasilStringSesudah=hasilStringSesudah, graphSebelumMinim=graphSebelumMinim, graphSesudahMinim=graphSesudahMinim)
30
```

Kode di atas melakukan uji DFA sebelum minimalisasi dengan menggunakan string input yang diberikan oleh user melalui formulir HTML. Hasilnya disimpan dalam variabel **hasilStringSebelum**, yang menunjukkan apakah DFA menerima atau tidak menerima string yang diuji. Selanjutnya, DFA akan diminimalisasi menggunakan fungsi **minimalisasiDFA**, dan hasil minimalisasi disimpan dalam variabel **minimalDFA**. Kemudian, DFA setelah minimalisasi diuji kembali dengan string input yang sama, dan hasilnya disimpan dalam variabel **hasilStringSesudah**. Selanjutnya, nilai-nilai yang ingin ditampilkan seperti **state**, **simbol**, **transisi**, **state awal**, dan **state final** diambil dari DFA minimalisasi. Grafik representasi DFA sebelum dan setelah minimalisasi juga dibuat menggunakan fungsi **cetakDFA**.

4. Nomor 4

a. Input DFA

```
1  dfa1 = {}
2  dfa1['states'] = request.form['states1'].split()
3  dfa1['symbols'] = request.form['symbol1'].split()
4  dfa1['initial_state'] = request.form['initialState1']
5  dfa1['final_states'] = request.form['finalStates1'].split()
6  dfa1['transitions'] = {}
7  for state in dfa1['states']:
8      dfa1['transitions'][state] = {}
9      for symbol in dfa1['symbols']:
10         next_state = request.form.get(f'transitions1_{state}_{symbol}')
11         dfa1['transitions'][state][symbol] = next_state
12
13  dfa2 = {}
14  dfa2['states'] = request.form['states2'].split()
15  dfa2['symbols'] = request.form['symbol2'].split()
16  dfa2['initial_state'] = request.form['initialState2']
17  dfa2['final_states'] = request.form['finalStates2'].split()
18  dfa2['transitions'] = {}
19  for state in dfa2['states']:
20      dfa2['transitions'][state] = {}
21      for symbol in dfa2['symbols']:
22         next_state = request.form.get(f'transitions2_{state}_{symbol}')
23         dfa2['transitions'][state][symbol] = next_state
24
25  graph1_dfa04 = VisualizeDFA(dfa1)
26  graph2_dfa04 = VisualizeDFA(dfa2)
27
28  result = equivalenceDFA(dfa1, dfa2)
29
30  if result:
31      result_message = "Kedua DFA Ekuivalen"
32  else:
33      result_message = "Kedua DFA Tidak Ekuivalen"
```

User diminta untuk memasukkan spesifikasi dua DFA melalui sebuah formulir web. Spesifikasi DFA mencakup:

- states: Himpunan states DFA.
- symbols: Himpunan simbol input yang diterima oleh state DFA.
- initial_state: state awal DFA.
- final_states: state akhir DFA.
- transitions: transisi antar keadaan state DFA berdasarkan simbol input.

b. Visualisasi DFA

```
1  def VisualizeDFA(dfa):
2      dot = Digraph()
3      dot.attr(rankdir='LR')
4
5      # Tambahkan node
6      for state in dfa['states']:
7          if state in dfa['final_states']:
8              dot.node(state, shape='doublecircle')
9          else:
10             dot.node(state)
11
12     # Tambahkan transisi
13     for start_state, transitions in dfa['transitions'].items():
14         for symbol, next_state in transitions.items():
15             dot.edge(start_state, next_state, label=symbol)
16
17     dot.attr('node', shape='none', label='start')
18     dot.node('')
19     dot.edge('', dfa['initial_state'])
20
21     return dot.pipe(format='svg').decode('utf-8')
```

Setelah user memasukkan spesifikasi DFA, fungsi `visualizedfa` akan memvisualisasikan DFA tersebut dalam bentuk grafik menggunakan library `Graphviz`. DFA direpresentasikan sebagai grafik dengan keadaan (states) sebagai simpul (node) dan transisi antara keadaan sebagai tepi (edge) di antara simpul-simpul tersebut. DFA ditampilkan dalam format SVG.

c. Pemeriksaan Equivalen

Setelah DFA divisualisasikan, program akan memeriksa ekuivalensi kedua DFA tersebut menggunakan fungsi `equivalent`. Algoritma yang digunakan adalah algoritma pencarian ekuivalensi DFA. Langkah-langkahnya meliputi:

- **Inisialisasi Tabel Ekuivalensi:** Tabel ekuivalensi diinisialisasi dengan membandingkan status keadaan akhir dari kedua DFA. Jika kedua DFA memiliki keadaan akhir yang sama, maka keadaan tersebut dianggap ekuivalen.

```
1 def equivalenceDFA(dfa1, dfa2):
2     def initialize_equivalence_table(dfa1, dfa2):
3         equivalent_table = {}
4         for state1 in dfa1['states']:
5             equivalent_table[state1] = {}
6             for state2 in dfa2['states']:
7                 equivalent_table[state1][state2] = (state1 in dfa1['final_states']) == (state2 in dfa2['final_states'])
8         return equivalent_table
9
10    equivalent_table = initialize_equivalence_table(dfa1, dfa2)
11
12    if not equivalent_table[dfa1['initial_state']][dfa2['initial_state']]:
13        return False
```

- **Pemeriksaan Transisi:** Setelah inisialisasi, program memeriksa transisi dari setiap keadaan dengan inputan yang diterima. Jika transisi dari dua keadaan yang ekuivalen mengarah ke keadaan yang tidak ekuivalen, maka keadaan tersebut dianggap tidak ekuivalen.

```
1 for state1 in dfa1['states']:
2     for state2 in dfa2['states']:
3         for symbol in dfa1['symbols']:
4             next_state1 = get_next_state(state1, symbol, dfa1['transitions'])
5             next_state2 = get_next_state(state2, symbol, dfa2['transitions'])
6             if (next_state1 is None and next_state2 is not None) or (next_state1 is not None and next_state2 is None):
7                 equivalent_table[state1][state2] = False
```

- **Iterasi:** Proses di atas diulangi hingga tidak ada perubahan dalam tabel ekuivalensi.

```

1 while True:
2     changed = False
3     for state1 in dfa1['states']:
4         for state2 in dfa2['states']:
5             if not equivalent_table[state1][state2]:
6                 continue
7             for symbol in dfa1['symbols']:
8                 next_state1 = get_next_state(state1, symbol, dfa1['transitions'])
9                 next_state2 = get_next_state(state2, symbol, dfa2['transitions'])
10                if not are_states_equivalent(next_state1, next_state2, dfa1, dfa2, equivalent_table):
11                    equivalent_table[state1][state2] = False
12                    changed = True
13                    break
14            if changed:
15                break
16        if not changed:
17            break

```

- Pemeriksaan Akhir: Terakhir, program memeriksa apakah keadaan-keadaan akhir dari kedua DFA ekuivalen. Jika tidak, maka DFA tidak ekuivalen.

```

1 for state1 in dfa1['states']:
2     for state2 in dfa2['states']:
3         if are_states_equivalent(state1, state2, dfa1, dfa2, equivalent_table):
4             for symbol in dfa1['symbols']:
5                 next_state1 = get_next_state(state1, symbol, dfa1['transitions'])
6                 next_state2 = get_next_state(state2, symbol, dfa2['transitions'])
7                 if not are_states_equivalent(next_state1, next_state2, dfa1, dfa2, equivalent_table):
8                     return False
9             else:
10                if state1 in dfa1['final_states'] != state2 in dfa2['final_states']:
11                    return False
12 return True

```

d. Output

Setelah proses pemeriksaan selesai, hasilnya ditampilkan kepada user dalam halaman web. Pesan yang menyatakan apakah kedua DFA ekuivalen atau tidak, ditampilkan bersama dengan visualisasi grafik kedua DFA.

5. Nomor 5

a. DFA

Program meminta user untuk menginput states, simbol, state final, transition state masing-masing dan string. Ini adalah bagian inisialisasi DFA berdasarkan input user. Jika string memiliki simbol yang tidak ada dalam DFA, maka program akan menampilkan error. Setelah semua bagian diinput, program akan mengecek jika string diterima karena sesuai dengan state dan simbol yang telah diinput.

```

def soal5dfasubmit():
    def convert(p, q):
        if q in t:
            li.append(k[s.index(p)][t.index(q)])
            return k[s.index(p)][t.index(q)]
        else:
            return None

    s = request.form['states'].split()
    t = request.form['symbol'].split()
    last = request.form['finalStates']

    k = [[0 for _ in range(len(t))] for _ in range(len(s))]
    for i in range(len(s)):
        for j in range(len(t)):
            k[i][j] = request.form.get(f'transitions_{s[i]}_{t[j]}')

    li = []
    start = s[0]
    q = request.form['string']
    for i in q:
        start = convert(start, i)
        if start is None:
            result = "String tidak berisi yang ditentukan di Finite Automata"
            break
    if start == last:
        result = "String diterima"
    else:
        result = "String tidak diterima"

    return render_template('result5dfa.html', result=result)

```

b. ϵ -NFA

Program meminta user untuk menginput states, simbol, state final, transition state masing-masing (termasuk epsilon) dan string. Ini adalah bagian inisialisasi ϵ -NFA berdasarkan input user. Jika string kosong atau memiliki simbol yang tidak ada dalam ϵ -NFA, maka program akan menampilkan error. Setelah semua bagian diinput, program akan mengecek jika string diterima karena sesuai dengan state dan simbol yang telah diinput.

```

def soal5enfsubmit():
    def epsilon_closure(states):
        closure = set()
        stack = list(states)
        while stack:
            current_state = stack.pop()
            closure.add(current_state)
            epsilon_transitions = k[s.index(current_state)][-1]
            stack.extend(epsilon_transitions - closure)
        return closure

    def convert(states, symbol):
        next_states = set()
        for state in states:
            transitions = k[s.index(state)][t.index(symbol)]
            next_states.update(transitions)
        epsilon_transitions = epsilon_closure(next_states)
        next_states.update(epsilon_transitions)
        return next_states

    s = request.form['states'].split()
    t = request.form['symbol'].split()
    last = request.form['finalStates']

    k = [[set() for _ in range(len(t) + 1)] for _ in range(len(s))]
    for i in range(len(s)):
        for j in range(len(t) + 1):
            if j == len(t):
                k[i][j] = set(request.form.get(f'transitions_{s[i]}_ε', "").split())
            else:
                k[i][j] = set(request.form.get(f'transitions_{s[i]}_{t[j]}', "").split())

```

```

start = epsilon_closure({s[0]})
regex_pattern = request.form['string']

pattern = re.compile(regex_pattern)

if pattern.fullmatch(''):
    result = "String kosong tidak diterima"
elif pattern.fullmatch('ε'):
    start = epsilon_closure(start)
elif not pattern.fullmatch('ε'):
    for symbol in pattern.pattern:
        start = convert(start, symbol)
        if not start:
            result = "String tidak berisi yang ditentukan di Finite Automata"
            break

if any(state in last for state in start):
    result = "String diterima"
else:
    result = "String tidak diterima"
return render_template('resultSenfa.html', result=result)

```

c. NFA

Program meminta user untuk menginput states, simbol, state final, transition state masing-masing dan string. Ini adalah bagian inisialisasi NFA berdasarkan input user. Jika string kosong atau memiliki simbol yang tidak ada dalam NFA, maka program akan menampilkan error. Setelah semua bagian diinput, program akan mengecek jika string diterima karena sesuai dengan state dan simbol yang telah diinput.

```

def soal5nfasubmit():
    def convert(p, q):
        next_states = set()
        for state in p:
            transitions = k[state].get(q, set())
            next_states.update(transitions)
        return next_states

    s = request.form['states'].split()
    t = request.form['symbol'].split()
    last = set(request.form['finalStates'].split())

    k = {}
    for state in s:
        k[state] = {}
        for symbol in t:
            transitions = set(request.form.get(f'transitions_{state}_{symbol}', '').split())
            k[state][symbol] = transitions

    start = {s[0]}
    q = request.form['string']
    for i in q:
        start = convert(start, i)
        if not start:
            result = "String tidak berisi yang ditentukan di Finite Automata"
            break
    else:
        if any(state in last for state in start):
            result = "String diterima"
        else:
            result = "String tidak diterima"

    return render_template('result5nfa.html', result=result)

```

d. Regular Expression

Program meminta user untuk menginput regular expression dan string. Ini adalah bagian inisialisasi Regular Expression berdasarkan input user. Setelah semua bagian diinput, program akan mengecek jika string diterima karena sesuai dengan Regular Expression yang telah diinput.

```
def regexsubmit():
    regex_pattern = request.form['regex']
    string_to_check = request.form['string']
    if re.fullmatch(regex_pattern, string_to_check):
        result = "String diterima"
    else:
        result = "String tidak diterima"

    return render_template('result5regex.html', result = result)
```

BAB III

CARA MENGGUNAKAN SISTEM

Cara menjalankan / run program ini adalah membuka folder **"PROJECT TBA KELOMPOK 4"** melalui VSCode, lalu menjalankan / run file bernama **"app.py"** untuk menampilkan interface melalui website, maka akan menampilkan **"index.html"**. Selanjutnya user dapat memilih nomor mana yang akan digunakan pada simulator. Untuk melihat kode html pada masing-masing nomor dapat dilihat melalui folder **"templates"**.

1. Nomor 1

Untuk cara menggunakan sistem pada case nomor 1 ini terdapat langkah-langkah yaitu:

- User memasukan states pada kolom states dipisahkan dengan koma.
- Setelah itu masukan Alphabet atau dikenal simbol dipisahkan dengan koma juga.
- Masukan transisinya, jika kurang bisa memencet tombol "Tambah Transisi".
- Setelah itu masukan initial statenya.
- Setelah itu masukan Final Statenya atau accept state, dipisahkan dengan koma juga jika lebih dari 1.
- Jika sudah selesai menginput semua kolom, pencet submit dan program akan memunculkan tabelnya.

CONTOH SOAL :

NFA ke DFA

MENGUBAH NFA ATAUPUN e-NFA MENJADI DFA

States (Pisahkan dengan koma):

Alphabet (Pisahkan dengan koma):

Masukkan transisi:

Initial State:

Final States (Pisahkan dengan koma):

Belum ada tabel transisi DFA yang dihasilkan.

+

HASIL :

Submit

Tabel Transisi DFA:

	Kadaan	0	1
->	q0	q0, q1	q0
	q0, q1	q0, q1	q0, q2
*	q0, q2	q0, q1	q0

CONTOH SOAL :
e-NFA ke DFA

MENGUBAH NFA ATAUPUN e-NFA MENJADI DFA

States (Pisahkan dengan koma):

q0,q1,q2,q3

Alphabet (Pisahkan dengan koma):

0,1

Masukkan transisi:

q0,ε,q0

q0,ε,q1

q1,0,q1

q1,0,q2

q1,1,q1

q1,ε,q1

q2,1,q3

q2,ε,q2

q3,ε,q3

Tambah Transisi

Initial State:

q0

Final States (Pisahkan dengan koma):

q0,q3

Tabel Transisi DFA:

Keadaan	0	1
-> q0, q1	q1, q2	q1
q1	q1, q2	q1
q1, q2	q1, q2	q1, q3
* q1, q3	q1, q2	q1

2. Nomor 2

Untuk cara menggunakan sistem pada case nomor 2 ini terdapat langkah-langkah yaitu:

- a. Pada website akan diminta untuk menginput regular expression, maka user dapat mengisi pada tempat yang sudah tersedia. Aturan dalam inputan:
 - Untuk concat menggunakan symbol (.)
 - Untuk union menggunakan symbol (+)
 - Untuk closure menggunakan symbol (*)
 - Jika ingin menginputkan harus diperjelas seperti a.b.(a+b)*
- b. Kemudian klik submit, untuk menampilkan hasil dari regex yang berupa tabel yang berisikan hasil dari generate dari regular expression ke e-NFA dari state, symbol, next state, dan bentuk postfix.

Regular Expression Evaluation

Masukkan Regular Expression:

(a+b)*

Table Transisi

State	Simbol	State Selanjutnya
Regular Expression dalam bentuk postfix:		

Sesudah Reguler Expression diinputkan oleh user.

Regular Expression Evaluation

Masukkan Regular Expression:

Table Transisi

State	Simbol	State Selanjutnya
q0	epsilon	q1
q0	epsilon	q2
q1	epsilon	q3
q1	epsilon	q4
q3	a	q5
q4	b	q7
q5	epsilon	q6
q6	epsilon	q1
q6	epsilon	q2
q7	epsilon	q6

Regular Expression dalam bentuk postfix: ab+*

Sesudah diklik submit oleh user.

3. Nomor 3

Untuk cara menggunakan sistem pada case nomor 3 ini terdapat langkah-langkah sebagai berikut :

- a. Pertama user memasukkan state yang diinginkan. Untuk memasukkan beberapa state, user **diharuskan** menggunakan spasi untuk memisahkan antar statenya.
- b. Selanjutnya user memasukkan simbol yang diinginkan. Untuk memasukkan beberapa simbol, user **diharuskan** menggunakan spasi untuk memisahkan antar simbolnya.
- c. Kemudian user memasukkan state awal dan state final yang diinginkan. Untuk memasukkan beberapa state final user **diharuskan** menggunakan spasi untuk memisahkan antar state final.
- d. Setelah itu jangan lupa untuk mengisi string yang diinginkan user, contoh string dapat berupa **001, 00100, aabbab, aabba, aaab** dan lain-lain.
- e. Tabel transisi dinamis akan otomatis terbentuk ketika user telah mengisi states dan simbolnya. Pada tabel transisi tersebut user **diharuskan** mengisi transisi sesuai yang diinginkan untuk semua state. Setiap transisi tidak boleh kosong maka harus diisi semua.
- f. Setelah sudah mengisi semua tabel transisi, user mengklik tombol **Hitung Minimal DFA**.

- g. Setelah informasi DFA dimasukkan dan diproses, sistem akan melakukan minimalisasi DFA yang diberikan dan melakukan uji terhadap DFA tersebut dengan string input yang diisi user.
- h. Untuk hasil akhirnya tampilan akan menampilkan hasil dari DFA sebelum dan setelah minimalisasi, beserta dengan tabel transisi, state awal, state final dan grafiknya.

CONTOH SOAL :

Sebelum submit (memasukkan input) :

The screenshot shows a web browser window with the address bar displaying "127.0.0.1:5000/soal3?". The page title is "No 3 Minimal DFA". The main content area is titled "Minimal DFA" and contains several input fields and a table.

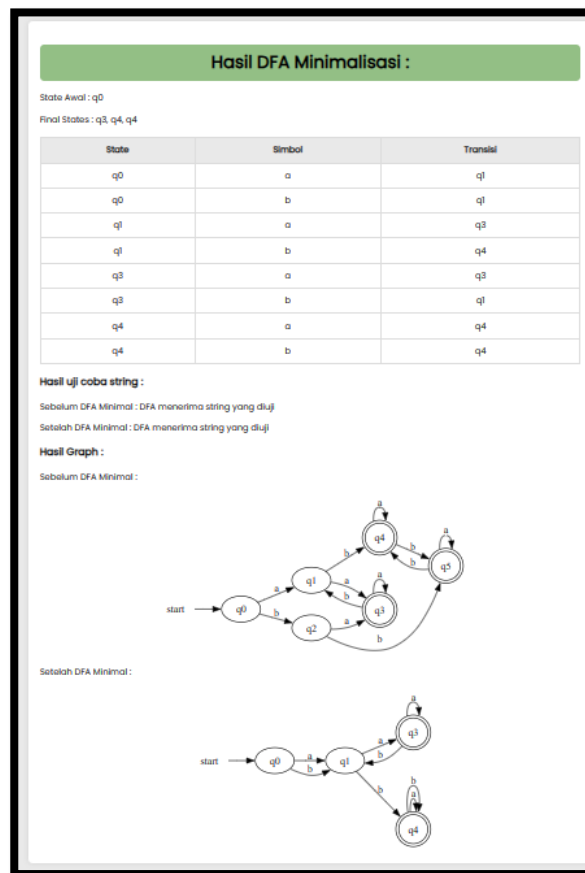
Input fields:

- State : q0 q1 q2 q3 q4 q5
- Simbol : a b
- State Awal : q0
- State Final : q3 q4 q5
- Uji String sebelum minimalisasi : abab
- Uji String sesudah minimalisasi : abab

Table:

States	a	b
q0	q1	q2
q1	q3	q4
q2	q5	q5
q3	q3	q1
q4	q4	q5
q5	q5	q4

Sesudah submit (hasil minimalisasi DFA) :



4. Nomor 4

Untuk cara menggunakan sistem pada case nomor 4 ini terdapat langkah-langkahnya:

1. User memasukkan state terlebih dahulu, gunakan pemisah berupa spasi (" ") jika user ingin memasukkan lebih dari 1 state.
2. Setelah itu, user dapat memasukkan inputan simbol, sama seperti memasukkan state, user diharuskan untuk menggunakan pemisah berupa spasi (" ") jika user ingin memasukkan lebih dari 1 inputan.
3. Kemudian, user dapat menentukan state awal dan final state, final state bisa lebih dari satu dengan syarat menggunakan pemisah berupa spasi (" ").
4. Terakhir, user dapat memasukkan transisi, transisi sudah otomatis di generate menggunakan logika JavaScript.

DFA 1

States :

q1 q2 q3

Simbol :

c d

State Awal :

q1

State Final :

q1

State: q1

Transition from state q1 with symbol c:

q1

Transition from state q1 with symbol d:

q2

State: q2

Transition from state q2 with symbol c:

q3

Transition from state q2 with symbol d:

q1

State: q3

Transition from state q3 with symbol c:

q2

Transition from state q3 with symbol d:

q3

Sesudah semua form DFA pertama diinput oleh user.

DFA 2

States :

Simbol :

State Awal :

State Final :

State: q4

Transition from state q4 with symbol c:

Transition from state q4 with symbol d:

State: q5

Transition from state q5 with symbol c:

Transition from state q5 with symbol d:

State: q6

Transition from state q6 with symbol c:

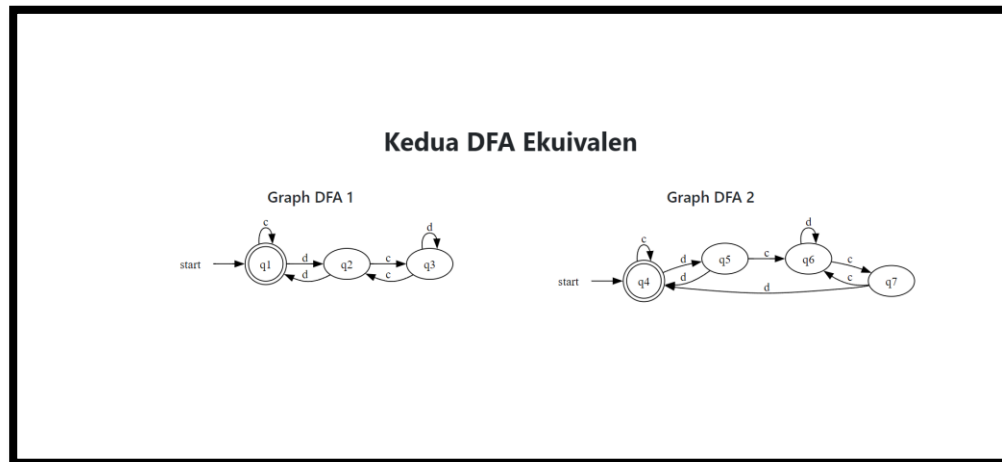
Transition from state q6 with symbol d:

State: q7

Transition from state q7 with symbol c:

Transition from state q7 with symbol d:

Sesudah semua form DFA kedua diinputkan oleh user.



Hasilnya ketika sudah diklik submit.

5. Nomor 5

Untuk cara menggunakan sistem pada case nomor 5 ini terdapat langkah-langkahnya yaitu :

a. DFA

Pertama, user memilih opsi DFA. Kemudian, user menginput states (q1 q2 q3), simbol (0 1), state final (q3), transition state masing-masing dan string (101). Setelah semua bagian diinput, user menekan tombol submit dan program akan mengecek string dan DFA. Program akan menampilkan kalimat “String diterima” atau “String tidak diterima”.

b. ϵ -NFA

Pertama, user memilih opsi DFA. Kemudian, user menginput states (q1 q2 q3), simbol (0 1), state final (q3), transition state masing-masing (termasuk untuk ϵ) dan string (101). Setelah semua bagian diinput, user menekan tombol submit dan program akan mengecek string dan ϵ -NFA. Program akan menampilkan kalimat “String diterima” atau “String tidak diterima”.

c. NFA

Pertama, user memilih opsi NFA. Kemudian, user menginput states (q1 q2 q3), simbol (0 1), state final (q3), transition state masing-masing dan string (101). Setelah semua bagian diinput, user menekan tombol submit dan program akan mengecek string dan NFA. Program akan menampilkan kalimat “String diterima” atau “String tidak diterima”.

d. Regular Expression

Pertama, user memilih opsi Regular Expression. Kemudian, user menginput regular expression (a|bc) dan string (a). Setelah semua bagian diinput, user menekan tombol submit dan program akan mengecek string dan Regular Expression. Program akan menampilkan kalimat “String diterima” atau “String tidak diterima”.

BAB IV

BEBAN KERJA

Anggota Kelompok :

1. Adidya Abimanyu (L0122004)
 - Mengerjakan nomor 4
2. Alfath Roziq Widhayaka (L0122012)
 - Mengerjakan nomor 3
3. Andrew Gustaya Kristiawan (L0122020)
 - Mengerjakan nomor 5
4. Ayasha Anggun Anindya (L0122028)
 - Mengerjakan nomor 2
5. Brama Prameswara Tarigan (L0122036)
 - Mengerjakan nomor 1

BAB V

KESIMPULAN

Laporan ini memberikan gambaran tentang implementasi dan fungsi simulator Teori Bahasa dan Automata. Dengan menyediakan langkah-langkah dan logika pada setiap nomor yang diperintahkan, laporan ini memberikan pemahaman yang mendalam tentang bagaimana program ini dapat mengonversi, mengelola, dan menganalisis finite automata. Dari pembahasan mengenai konversi NFA atau e-NFA ke DFA, pengkonversian e-NFA dari regular expression, proses minimalisasi DFA, ekuivalensi dua buah DFA, dan uji coba string pada DFA, NFA, e-NFA, dan Regular Expression. Laporan ini memberikan panduan yang jelas tentang bagaimana setiap langkah diimplementasikan dalam program. Selain itu, laporan ini juga membahas penggunaan input dari pengguna, baik melalui input user pada HTML, serta bagaimana program memproses informasi yang diberikan pengguna untuk menghasilkan output yang sesuai.