CPSC-406 Report

Ryan Benner
Chapman University

April 8, 2025

Abstract

Contents

1 Introduction

2 Week by Week

2.1 Week 1

1. Automata

- machine model which has variety of computations without memory - traversal (trees, graphs, nodes) - Turing machine - traffic light

- parking machine - mechanical calculators - streaming - regular expressions (regex): such as NLP, parsing, coding theory, file management

automata: implement algorithms, admit algorithmic instructions.

2. Computability And complexity

- universal language to analyze algorithms/problems/computations for: - efficiency/(runtime, spacetime, etc) resource/scaling - comparison between complexity, hardness of problems - complexity classes - big-o-notation - reducing problems - P vs NP, sat problems

3. Graph Algorithms

- network theory (data, mechanical, chemical, operations)

2/6

1. Automata theory

states: positions: (head) $\overrightarrow{l}$ () $\overrightarrow{t}$ (end)

ex.1 parking machine: charge/hr = 25c accepted: coins, no change, no 1, 2c coins

states: - value paid so far (0) 0 -> [5,10,25]

ex.2 valid var names: 1. index 0: letter? 2. index n $>$ 0: letter or digit or symbol

2.2 Week 2

Notes

Chapter 2.1 of ITALC discusses the basic model of computation that demonstrates the structure and components of finite automata. It explains that a finite automaton is a simple computational machine which is composed of a set of states, including a start state, and one or more end states. As the automaton reads an input, it transitions between states based on a set of rules, the transition functions. The section uses examples with simple intuition, such as an on/off switch, to show how a system can 'remember' essential information despite its simplicity. Also, it talks about the difference between deterministic and non-deterministic models. A deterministic model is one in which each state has a unique transition for each input, and a non-deterministic model is one where multiple transitions may be possible for one input. Although these two models are very different in practice, they recognize the same set of basic regular language.

Homework 1

Introduction to Automata Theory:

Homework: Characterize all the accepted words (i.e., describe exactly those words that get recognized).
Answer: [5, 5, 5, 5, 5], [5, 5, 5, 10], [5, 5, 10, 5], [5, 10, 5, 5], [5, 10, 10], [10, 5, 5, 5], [10, 5, 10], [10, 10, 5]

Homework: Characterize all the accepted words. Can you describe them via a regular expression?
Answer: Any word which ends in 'pay' will result in end state being unlocked

Deterministic and Non-Deterministic Finite Automata:

Homework: Determine for the following words, if they are contained in L_1, L_2, or L_3.
Answer:

Homework: Consider the DFA from above: Consider the paths corresponding to the words $w_1 = 0010$, $w_2 = 1101$, $w_3 = 1100$.
Answer: w_1 and w_2 both achieve the end state

Comments and Questions

Because finite automata have limited memory, their ability to recognize more complex language patterns is restricted. Will the range of what is categorized as a finite automaton broaden over time? How could we modify automata to recognize more complex language?

word	A_1	A_2
aaa	no	yes
aab	yes	no
aba	no	no
abb	no	no
baa	no	yes
bab	no	no
bba	no	no
bbb	no	no

2.3 Week 3

Notes

Homework 2

Exercise 2, 1-3:

The file **dfa.py** implements a class representing a deterministic finite automaton(DFA) The automaton is defined by a set of states, inputs, transition functions, an initial state, and accepting states, all of which are initialized in the constructor function. The repr method offers a formatted string representation of the DFA for easy inspection. The run method processes an input word by iterating through its characters and transitions between states according to the defined rules. If an undefined transition is encountered or an invalid symbol is detected, the word is rejected. Finally, the refuse method generates a complementary DFA by changing the set of accepting states to accept words that the original DFA rejects.
The file **dfa − ex01.py** implements two DFAs, A1 and A2, and runs each through dfa.py to test them repsectively.

```
# dfa.py
class DFA :

    # init the DFA
    def __init__(self, Q, Sigma, delta, q0, F) :
        self.Q = Q            # set of states
        self.Sigma = Sigma    # set of symbols
        self.delta = delta    # transition function
        self.q0 = q0          # initial state
        self.F = F            # final (accepting) states

    # print the data of the DFA
    def __repr__(self) :
        return f"DFA({self.Q},\n\t{self.Sigma},\n\t{self.delta},\n\t{self.q0},\n\t{self.F})"

    # run the DFA on the word w
    # return if the word is accepted or not
    def run(self, w) :
        current_state = self.q0
        for symbol in w:
            if symbol not in self.Sigma:
                raise ValueError(f"Symbol {symbol} not in the DFA alphabet {self.Sigma}.")
            # look up transition state, if none is defined the word is rejected
            if (current_state, symbol) not in self.delta:
                return False
            current_state = self.delta[(current_state, symbol)]
```

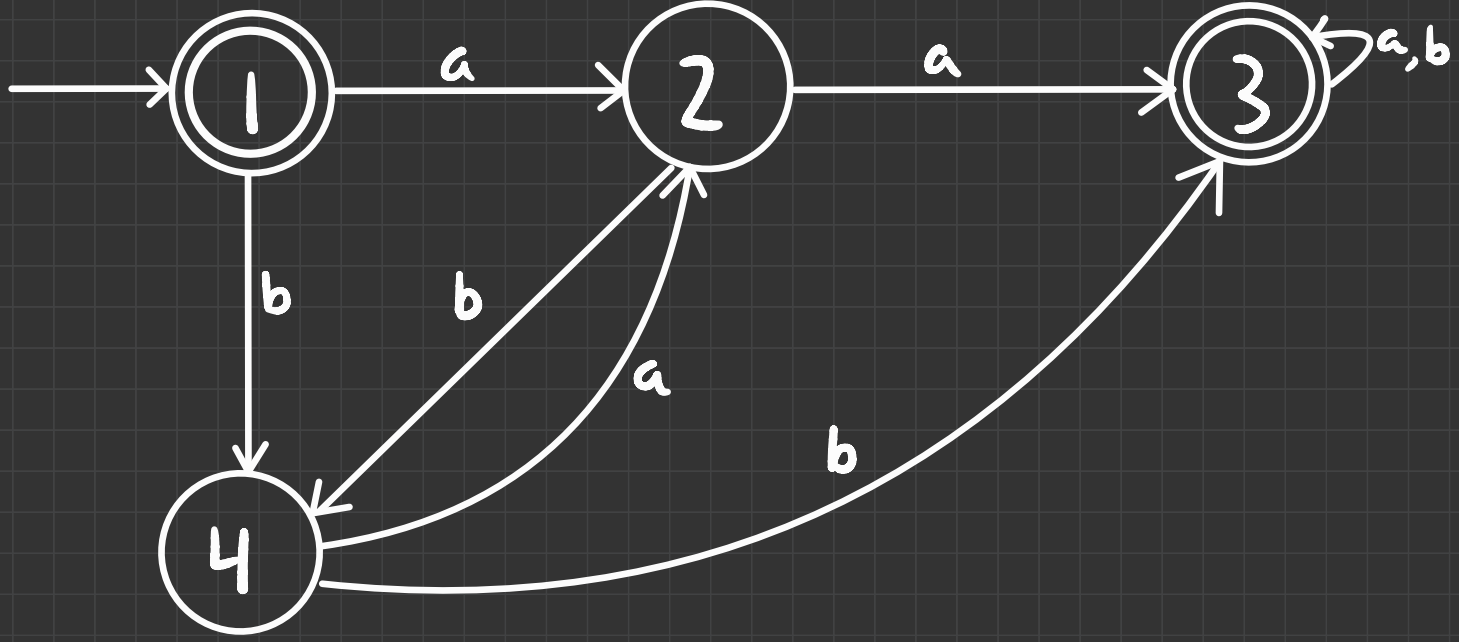
```
        return current_state in self.F

    # returns a new DFA that accepts the words that the current DFA refuses and vice versa
    def refuse(self):
        new_F = self.Q - set(self.F)
        return DFA(self.Q, self.Sigma, self.delta, self.q0, new_F)

    def to_NFA(self):
        # make new transition function where each DFA transition becomes a set
        from nfa import NFA
        nfa_delta = {}
        for (state, symbol), next_state in self.delta.items():
            nfa_delta[(state, symbol)] = {next_state}
        return NFA(self.Q, self.Sigma, nfa_delta, self.q0, self.F)
```

```
# dfa_ex01.py
import dfa as dfa

# generate words for testing
def generate_words():
    words = []
    alphabet = ['a', 'b']
    for first in alphabet:
        for second in alphabet:
            for third in alphabet:
                words.append(first + second + third)
    return words

def __main__() :
    Q1 = {'1', '2', '3', '4'}
    Sigma1 = {'a','b'}
    delta1 = {
        ('1', 'a'): '2',
        ('1', 'b'): '4',
        ('2', 'a'): '2',
        ('2', 'b'): '3',
        ('3', 'a'): '2',
        ('3', 'b'): '2',
        ('4', 'a'): '4',
        ('4', 'b'): '4'
    }
    q01 = '1'
    F1 = {'3'}

    Q2 = {'1', '2', '3'}
    Sigma2 = {'a','b'}
    delta2 = {
        ('1', 'a'): '2',
        ('1', 'b'): '1',
        ('2', 'a'): '3',
        ('2', 'b'): '1',
        ('3', 'a'): '3',
        ('3', 'b'): '1',
    }
    q02 = '1'
```

```
    F2 = {'3'}

    A1 = dfa.DFA(Q1, Sigma1, delta1, q01, F1)
    A2 = dfa.DFA(Q2, Sigma2, delta2, q02, F2)

    words = generate_words()
    automata = [A1, A2]

    for X in automata: # test words
         print(f"{X.__repr__()}")
         for w in words:
            print(f"{w}: {X.run(w)}")
         print("\n")

__main__()
```

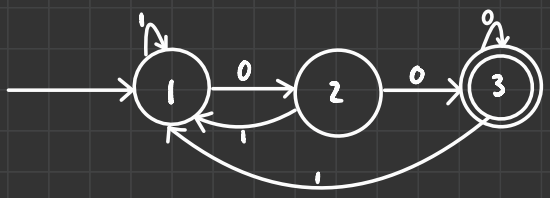
Drawings for Lab Exercise 4, ITALC Exercise 2.2.4

Lab Exercise 4:

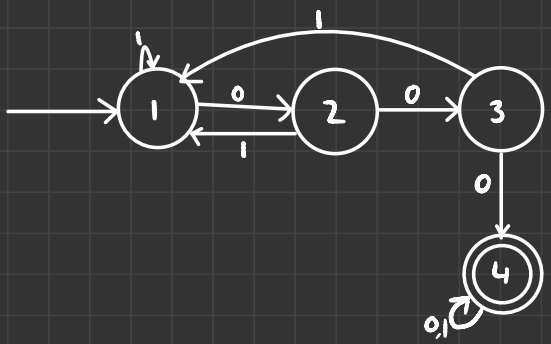


ITALC Exercise 2.2.4:

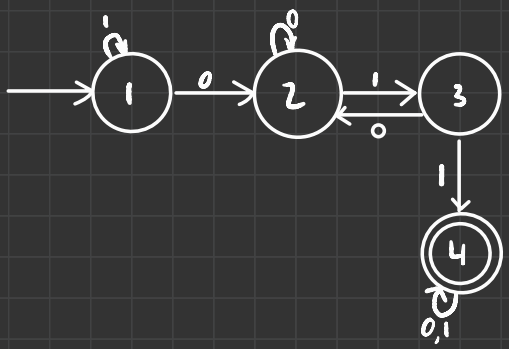
a) String ending in 00



b) all strings w/ 3 consecutive 0s



c) set of strings having substring 011



Comments and Questions

I occasionally find myself struggling to recognize the pattern when first looking at a new DFA. Is there an insightful method or trick to recognizing these patterns more easily?

2.4 Week 4

Notes

Homework 3

Homework 1

1. The language of the automata A2 can be described as starting with an a and having an odd length. Any b's that may occur must do so from state 2. This can be described as a regular expression $a((a|b)a)*$.

2. The extended transition functions can be evaluated as follows:

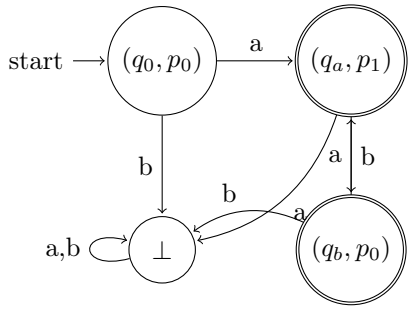
A1:

- $\delta_1(1, a) = 2$
- $\delta_1(2, b) = 4$
- $\delta_1(4, a) = 2$
- $\delta_1(2, a) = 3$

A2:

- $\delta_2(1, a) = 2$
- $\delta_2(2, b) = 1$
- $\delta_2(1, b) = 3$
- $\delta_2(3, a) = 3$

Homework 2



2. When constructing the product automaton, a word w is processed by both components simultaneously. The word is said to be accepted iff the computation ends in a state (q, p) where q is in F(1) and p is in F(2). This means that w is accepted by both A(1) and A(2), so L(A) can be the intersection between the two.

3. In the product automaton constrction, the set of states is always $Q_1 * Q_2$ and the transitions are the same. To find an automaton which is the union of the two languages, we change the accepting states. So, a word w is accepted by A' iff it is accepted by at least one of the original automata, which gives L(A') to be the union of L(A1) and L(A2).

To summarize:

$$
\begin{aligned}
Q &= Q^{(1)} \times Q^{(2)}, \\
\delta((q,p),x) &= (\delta^{(1)}(q,x),\, \delta^{(2)}(p,x)), \\
q_0 &= (q_0^{(1)}, q_0^{(2)}), \\
F' &= \left\{(q,p) \in Q^{(1)} \times Q^{(2)} : q \in F^{(1)} \text{ or } p \in F^{(2)}\right\}.
\end{aligned}
$$

Exercise 2.2.7 - ITALC

Theorem. Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let $q \in Q$ be such that

$$\delta(q,a) = q \quad \text{for all } a \in \Sigma.$$

Then for all $w \in \Sigma^*$, $\hat{\delta}(q,w) = q$.

Proof. We prove by induction on the length of w.

Base Case: $w = \varepsilon$ (the empty string). By definition of the extended transition function,

$$\hat{\delta}(q,\varepsilon) = q.$$

Inductive Step: Assume that for some $w \in \Sigma^*$, $\hat{\delta}(q,w) = q$. Let $a \in \Sigma$. Then,

$$\hat{\delta}(q,wa) = \delta(\hat{\delta}(q,w),a) = \delta(q,a) = q.$$

Thus, by the principle of induction, for all $w \in \Sigma^*$, $\hat{\delta}(q,w) = q$.

Comments and Questions

Do automata with some form of union or intersection have real world applications relating to natural language processing? Could they be used to improve parsing in conversational AI?

2.5 Week 5

Notes

Chapter 3.1-3.2 of ITALC explores regular expressions as a powerful way to more accurately describe formal languages, specifically being useful in various text processing. Regular expressions are built using basic operations such as union, concatenation,and Kleene closure, which combines simple patterns in order to create more complex expressions. These operations and the way they are used is defined through standard set operations. Union is able to combine languages by joining their elements, concatenation makes new strings by combining parts from each language, and Kleene closure allows for the generation of all potential strings. Also, Kleene's algorithm is shown as a structured method in order to change DFAs into regular expressions, usually relying on induction that builds expressions incrementally that represent state transitions. With larger automata, this algorithm can become very computationally expensive. Finally, the transition between regular expressions and automata is bidirectional, meaning you can go from one to the other and back again.

Homework 4

Homework 1:

Let $\mathcal{A} = (Q, \Sigma, \delta : Q \times \Sigma \to Q, q_0, F)$ be a DFA.

1. **Viewing $\mathcal{A}$ as an NFA:**

   A DFA is a specific case of an NFA where the transition function returns only one next state for each input symbol and state. To show the DFA as an NFA, we make a new transition function $\delta' : Q \times \Sigma \to \mathcal{P}(Q)$ sucsoh that:

   $$\delta'(q, a) = \{\delta(q, a)\}$$

   This shows that each deterministic transition becomes a singleton set in the NFA transition function. All other components of the DFA remain unchanged:

   - $Q' = Q$
   - Σ remains the same
   - $q_0' = q_0$
   - $F' = F$

2. **General Construction:**

   Given a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, we define an equivalent NFA $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$ as follows:

   $$\begin{aligned} Q' &= Q \\ \Sigma' &= \Sigma \\ q_0' &= q_0 \\ F' &= F \\ \delta'(q, a) &= \{\delta(q, a)\} \quad \text{for all } q \in Q, a \in \Sigma \end{aligned}$$

3. **Justification:**

   This design ensures that $\mathcal{A}'$ accepts the same language as $\mathcal{A}$. Because each transition in the NFA δ' corresponds to the DFA's transition , the NFA has exactly one computational path for any given input string, mirroring the DFA. So,

   $$L(\mathcal{A}) = L(\mathcal{A}')$$

Homework 2:

1. **Language accepted by $\mathcal{A}$:** The NFA accepts all binary strings that have the substring `010`.
2. **Specification of $\mathcal{A}$:**
   - $Q = \{q_0, q_1, q_2, q_3\}$
   - $\Sigma = \{0, 1\}$
   - q_0 is the start state
   - $F = \{q_3\}$
   - Transition function δ:

   $$\begin{aligned} &\delta(q_0, 0) = \{q_0\} \\ &\delta(q_0, 1) = \{q_0, q_1\} \\ &\delta(q_1, 0) = \{q_2\} \\ &\delta(q_2, 0) = \{q_3\}, \quad \delta(q_2, 1) = \{q_1\} \\ &\delta(q_3, 0) = \{q_3\}, \quad \delta(q_3, 1) = \{q_3\} \end{aligned}$$

3. **Extended transition: compute $\hat{\delta}(q_0, 10110)$ step-by-step:**

- Step 0: $\{q_0\}$
- Read 1: $\{q_0, q_1\}$
- Read 0: $\{q_0, q_2\}$
- Read 1: $\{q_0, q_1\}$
- Read 1: $\{q_0, q_1\}$
- Read 0: $\{q_0, q_2\}$
- Final result: $\hat{\delta}(q_0, 10110) = \{q_0, q_2\}$

4. **All paths for $v = 1100$ and $w = 1010$:**

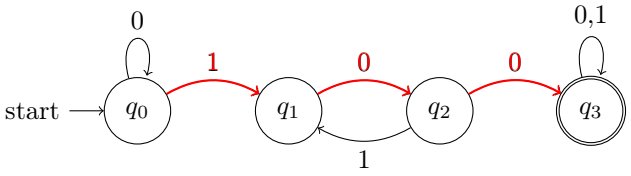
   Step-by-step for $v = 1100$:

   - Start at $\{q_0\}$
   - Read 1: $\{q_0, q_1\}$
   - Read 1: $\{q_0, q_1\}$
   - Read 0: $\{q_0, q_2\}$
   - Read 0: $\{q_0, q_3\} \rightarrow$ accepted

   Step-by-step for $w = 1010$:

   - Start at $\{q_0\}$
   - Read 1: $\{q_0, q_1\}$
   - Read 0: $\{q_0, q_2\}$
   - Read 1: $\{q_0, q_1\}$
   - Read 0: $\{q_0, q_2\} \rightarrow$ rejected

   Common diagram of all paths:



5. **Powerset construction for DFA $\mathcal{A}^D$:**

   Start from state $\{q_0\}$:

   - $\delta(\{q_0\}, 0) = \{q_0\}$
   - $\delta(\{q_0\}, 1) = \{q_0, q_1\}$

   Continue expanding:

   - $\delta(\{q_0, q_1\}, 0) = \{q_0, q_2\}$
   - $\delta(\{q_0, q_1\}, 1) = \{q_0, q_1\}$
   - $\delta(\{q_0, q_2\}, 0) = \{q_0, q_3\}$
   - $\delta(\{q_0, q_2\}, 1) = \{q_0, q_1\}$

- $\delta(\{q_0, q_3\}, 0) = \{q_0, q_3\}$, etc.

Accepting states: all subsets that contain q_3

6. **Verification and minimization:** $L(\mathcal{A}) = L(\mathcal{A}^D)$ by construction. A simple DFA for the language contains substring '010' has 4 states, so $\mathcal{A}^D$ can be simplified further.

Comments and Questions

How does the powerset construction in Homework 2 show the relationship between DFAs and NFAs, and how does this relate to the transformation approach in Homework 1?

2.6 Week 6

Notes

Comments and Questions

2.7 Week 7

Notes

Homework

DFA Regular Expressions and State Elimination (Exercise 3.2.1)

Given a transition table:

	0	1
$\rightarrow q_1$	q_2	q_1
q_2	q_3	q_1
$*q_3$	q_3	q_2

a. $R_{ij}^{(0)}$ (Initial regular expressions with no intermediate states)

Let

$$R_{ij}^{(0)} = \begin{cases} \text{the symbol(s) labeling direct transitions from state } i \text{ to state } j, \\ \varnothing, \quad \text{if no direct transition,} \\ \epsilon, \quad \text{if } i = j. \end{cases}$$

Thus,

$$\begin{aligned} R_{11}^{(0)} &= \epsilon, \quad R_{12}^{(0)} = 0, \quad R_{13}^{(0)} = \varnothing, \\ R_{21}^{(0)} &= 1, \quad R_{22}^{(0)} = \epsilon, \quad R_{23}^{(0)} = 0, \\ R_{31}^{(0)} &= \varnothing, \quad R_{32}^{(0)} = 1, \quad R_{33}^{(0)} = 0 \mid \epsilon. \end{aligned}$$

b. $R_{ij}^{(1)}$ (Using state q_1 as intermediate)

Using the formula

$$R_{ij}^{(k)} = R_{ij}^{(k-1)} \cup R_{ik}^{(k-1)} \left(R_{kk}^{(k-1)}\right)^* R_{kj}^{(k-1)},$$

with q_1 as the singular intermediate state we get:

$$\begin{aligned}
R_{11}^{(1)} &= \epsilon \cup (\epsilon)(\epsilon)^*\epsilon = \epsilon, \\
R_{12}^{(1)} &= 0 \cup (\epsilon)(\epsilon)0 = 0, \\
R_{13}^{(1)} &= \varnothing \cup (\epsilon)(\epsilon)\varnothing = \varnothing, \\
R_{21}^{(1)} &= 1 \cup (1)(\epsilon)^*\epsilon = 1, \\
R_{22}^{(1)} &= \epsilon \cup (1)(\epsilon)0 = 10, \\
R_{23}^{(1)} &= 0 \cup (1)(\epsilon)\varnothing = 0, \\
R_{31}^{(1)} &= \varnothing \cup (1)(\epsilon)^*\epsilon = \varnothing, \\
R_{32}^{(1)} &= 1 \cup (1)(\epsilon)0 = 1, \\
R_{33}^{(1)} &= 0 \cup (1)(\epsilon)\varnothing = 0.
\end{aligned}$$

c. $R_{ij}^{(2)}$ (Using q_1 and q_2 as intermediates)

Now we add q_2 as an intermediate state:

$$\begin{aligned}
R_{11}^{(2)} &= \epsilon \cup 0\,(10)^*\,1, \\
R_{12}^{(2)} &= 0 \cup 0\,(10)^*\,10, \\
R_{13}^{(2)} &= 0\,(10)^*\,0, \\
R_{21}^{(2)} &= 1 \cup 10\,(10)^*\,1, \\
R_{22}^{(2)} &= 10\,(10)^*\,10 \cup \epsilon, \\
R_{23}^{(2)} &= 0 \cup 10\,(10)^*\,0, \\
R_{31}^{(2)} &= \varnothing \cup 1\,(10)^*\,1, \\
R_{32}^{(2)} &= 1 \cup 1\,(10)^*\,10, \\
R_{33}^{(2)} &= 0 \cup 1\,(10)^*\,0.
\end{aligned}$$

d. Regular Expression for the Language

Start state: q_1 Final state: q_3.

The regex is $R_{13}^{(2)}$, so

$$R = 0\,(10)^*\,0.$$

e. Transition Diagram and State Elimination (eliminate q_2)

A simpler diagram of the DFA is:

$$q_1 \xrightarrow{0} q_2 \xrightarrow{0} q_3 \quad \text{(final)}$$

with loops and additional transitions that are indicated in the original DFA. After getting rid of q_2, the path from q_1 to q_3 becomes:

$$(1 \cup 01)00(0),$$

so final regular expression becomes

$$R = (1 \cup 01)00(0).$$

DFA Regular Expressions and State Elimination (Exercise 3.2.2)

Given the transition table:

	0	1
$\rightarrow q_1$	q_2	q_3
q_2	q_1	q_3
$*q_3$	q_2	q_1

a. $R_{ij}^{(0)}$ (Initial regular expressions with no intermediate states)

Let

$$\begin{aligned} R_{11}^{(0)} &= \epsilon, \quad R_{12}^{(0)} = 0, \quad R_{13}^{(0)} = 1, \\ R_{21}^{(0)} &= 0, \quad R_{22}^{(0)} = \epsilon, \quad R_{23}^{(0)} = 1, \\ R_{31}^{(0)} &= 1, \quad R_{32}^{(0)} = 0, \quad R_{33}^{(0)} = \epsilon. \end{aligned}$$

b. $R_{ij}^{(1)}$ (Using q_1 as intermediate)

Apply:

$$R_{ij}^{(1)} = R_{ij}^{(0)} \cup R_{i1}^{(0)} \, (R_{11}^{(0)})^* \, R_{1j}^{(0)}.$$

This gives:

$$\begin{aligned} R_{11}^{(1)} &= \epsilon \cup \epsilon \cdot \epsilon \cdot \epsilon = \epsilon, \\ R_{12}^{(1)} &= 0 \cup \epsilon \cdot \epsilon \cdot 0 = 0, \\ R_{13}^{(1)} &= 1 \cup \epsilon \cdot \epsilon \cdot 1 = 1, \\ R_{21}^{(1)} &= 0 \cup 0 \cdot \epsilon \cdot \epsilon = 0, \\ R_{22}^{(1)} &= \epsilon \cup 0 \cdot \epsilon \cdot 0 = \epsilon \cup 00, \\ R_{23}^{(1)} &= 1 \cup 0 \cdot \epsilon \cdot 1 = 1 \cup 01, \\ R_{31}^{(1)} &= 1 \cup 1 \cdot \epsilon \cdot \epsilon = 1, \\ R_{32}^{(1)} &= 0 \cup 1 \cdot \epsilon \cdot 0 = 0 \cup 10, \\ R_{33}^{(1)} &= \epsilon \cup 1 \cdot \epsilon \cdot 1 = \epsilon \cup 11. \end{aligned}$$

c. $R_{ij}^{(2)}$ (Using q_1 and q_2 as intermediates)

Apply:

$$R_{ij}^{(2)} = R_{ij}^{(1)} \cup R_{i2}^{(1)} \, (R_{22}^{(1)})^* \, R_{2j}^{(1)}.$$

For example:

$$\begin{aligned} R_{11}^{(2)} &= \epsilon \cup 0 \, ((\epsilon \cup 00))^* \, 0, \\ R_{12}^{(2)} &= 0 \cup 0 \, ((\epsilon \cup 00))^* \, (\epsilon \cup 00), \\ R_{13}^{(2)} &= 1 \cup 0 \, ((\epsilon \cup 00))^* \, (1 \cup 01), \end{aligned}$$

and in a similar manner for the other entries.

d. Regular Expression for the Language

Start state: q_1 Final state: q_3.

Thus,

$$R = R_{13}^{(2)} = 1 \cup 0 \, ((\epsilon \cup 00))^* (1 \cup 01).$$

e. Transition Diagram and State Elimination (eliminate q_2)

A simplified diagram is as follows:

$$q_1 \xrightarrow{0} q_2 \xrightarrow{0} q_1 \quad \text{and} \quad q_3 \xleftarrow{1} q_3.$$

By getting rid of q_2 and considering the other paths:

$$q_1 \xrightarrow{0} q_2 \xrightarrow{1} q_3 \quad \Rightarrow \quad 01,$$

$$q_1 \xrightarrow{0} q_2 \xrightarrow{0} q_1 \quad \Rightarrow \quad 00 \text{ (loop)},$$

the final regex becomes:

$$R = (1 \cup 0(00)^*1)\,.$$

DFA Minimization (Exercise 4.4.1)

Given the transition table:

	0	1
$\to A$	B	A
B	A	C
C	D	B
$*D$	D	A
E	D	F
F	G	E
G	F	G
H	G	D

a. Table of Distinguishabilities

We make a lower-triangular table marking different pairs of states.

Step 1: Mark all the pairs that have one state as final (D) and the other as non-final.

Step 2: Propagate markings by checking transitions.

Final table (checkmark = marked/inequivalent, blank = equivalent):

	A	B	C	D	E	F	G	H
B	✓							
C	✓	✓						
D	✓	✓	✓		✓	✓	✓	✓
E	✓	✓	✓	✓				
F	✓	✓	✓	✓	✓			
G	✓	✓	✓	✓	✓	✓		
H	✓	✓	✓	✓	✓	✓	✓	

The unmarked (equivalent) states are E, F, and G.

b. Minimum-State Equivalent DFA

Merge equal states:

$$[E, F, G] \to \texttt{EFG}.$$

The new transition table is:

State	0	1	Accept?
A	B	A	No
B	A	C	No
C	D	B	No
D	D	A	Yes
`EFG`	`EFG`	`EFG`	No
H	`EFG`	D	No

DFA Minimization (Exercise 4.4.2)

Given the transition table:

	0	1
$\rightarrow A$	B	E
B	C	F
$*C$	D	H
D	E	I
E	F	H
$*F$	G	B
G	H	B
H	I	C
$*I$	A	E

a. Table of Distinguishabilities

Final states: C, F, and I.

Mark all the pairs with one final and one non-final state, then make distinctions. The final table (checkmark = distinguishable) is:

	A	B	C	D	E	F	G	H	I
B	✓								
C	✓	✓							
D	✓	✓	✓						
E	✓	✓	✓	✓					
F	✓	✓	✓	✓	✓				
G	✓	✓	✓	✓	✓	✓			
H	✓	✓	✓	✓	✓	✓	✓		
I	✓	✓	✓	✓	✓	✓	✓	✓	

The equivalent states (unmarked) are C, F, and I, which we merge as:

$$[C, F, I] \rightarrow \texttt{CFI}.$$

b. Minimum-State Equivalent DFA

The new transition table becomes:

State	0	1	Accept?
A	B	E	No
B	`CFI`	`CFI`	No
`CFI`	D	H	Yes
D	E	`CFI`	No
E	`CFI`	H	No
G	H	B	No
H	`CFI`	`CFI`	No

This minimized DFA reduces the original 9 states to 7 states with the final state being `CFI`.

Comments and Questions

In Exercise 3.2.1, how does the order of state elimination change the final regular expression's structure, and could a different order get us to a more simplified form?

3 Synthesis

4 Evidence of Participation

5 Conclusion

References

[BLA] Author, Title, Publisher, Year.