Report: Programming with Automata

Max Randall

February 23, 2025

Contents

1 Introduction

In this homework, we explore the implementation of deterministic finite automata (DFAs) in Python. We will go beyond the simple graphical representation of DFAs and build them using Python types.

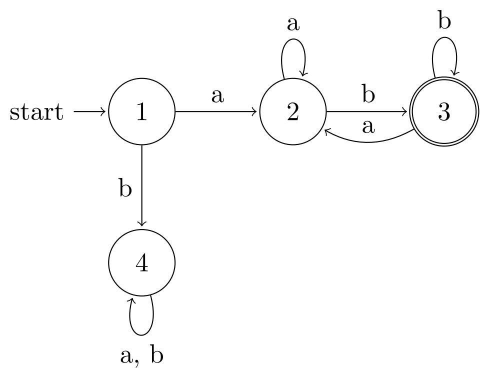
2 Programming with Automata

We will impliment the follwing DFAs in Python using this DFA class:

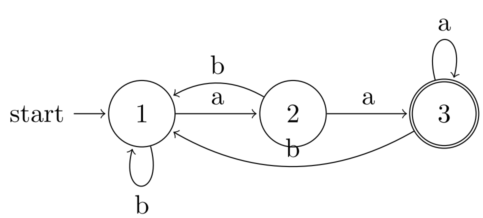
```
class DFA:

    def __init__(self, Q, Sigma, delta, q0, F):
        self.Q = Q  # Set of states
        self.Sigma = Sigma  # Alphabet
        self.delta = delta  # Transition function (dict)
        self.q0 = q0  # Initial state
        self.F = F  # Set of accepting states

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

    def run(self, w):
        current_state = self.q0  # Start at initial state
        loop = 0
        for symbol in w:
            loop += 1
            if symbol not in self.Sigma:
                return False  # Reject if symbol is not in the alphabet
            if (current_state, symbol) not in self.delta:
                return False  # Reject if there's no valid transition
            current_state = self.delta[(current_state, symbol)]  # Move to next state

        return current_state in self.F  # Accept if final state is in F
```

Exercise 1: Word Processing with DFAs

Given DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$:



Automaton $\mathcal{A}_1$



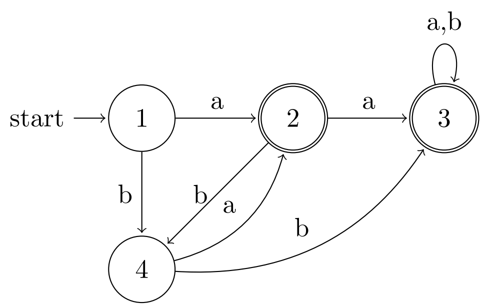
Automaton $\mathcal{A}_2$

Here is how we initialize the DFAs in python:

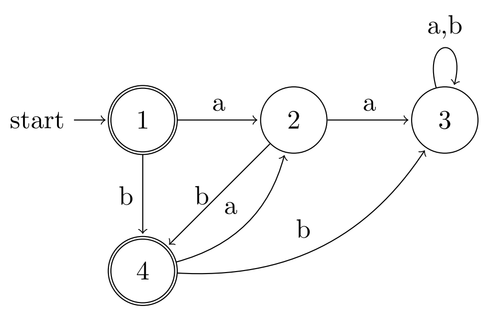
```
# DFA A1
Q = {1,2,3,4}
Sigma = {'a','b'}
delta = {(1,'a'):2, (1,'b'):4, (2,'a'):2, (2,'b'):3,
         (3,'a'):2, (3,'b'):2, (4,'a'):4, (4,'b'):4}
q0 = 1
F = {3}
A1 = dfa.DFA(Q, Sigma, delta, q0, F)

# DFA A2
Q = {1,2,3}
Sigma = {'a','b'}
delta = {(1,'a'):2, (1,'b'):1, (2,'a'):3, (2,'b'):1,
         (3,'a'):3, (3,'b'):1}
q0 = 1
F = {3}
A2 = dfa.DFA(Q, Sigma, delta, q0, F)
```

3 Exercise 4: Complement DFA

Construct an automaton A_0 such that A_0 accepts exactly the words that A refuses and vice versa. Implement the method `refuse` in `dfa.py` to return this new DFA.



To construct an automaton A_0 that only accepts words that A refuses we can simply swap the accepting states with the previously not accepted states.



We can represent this operation with a refuse() function in Python.

```
def refuse(A):
    """Constructs a DFA A0 that accepts exactly the words that A refuses and vice versa."""
    Q0 = A.Q
    Sigma0 = A.Sigma
    delta0 = A.delta
    q0_0 = A.q0
    F0 = Q0 - A.F  # Complement of the accepting states

    return dfa.DFA(Q0, Sigma0, delta0, q0_0, F0)
```

Summary of chapter 2.2.4

A **Deterministic Finite Automaton (DFA)** is a formal model of computation that processes input sequences while maintaining a single, well-defined state at any given time. The term "deterministic" means that for each input symbol, the automaton transitions to exactly one state.

Components of a DFA

A DFA consists of five elements:

1. **Finite set of states** Q
2. **Finite set of input symbols** Σ
3. **Transition function** δ: $Q \times \Sigma \to Q$, mapping states and inputs to new states
4. **Start state** $q_0 \in Q$
5. **Set of accepting states** $F \subseteq Q$

A DFA is often represented as a **five-tuple**:

$$A = (Q, \Sigma, \delta, q_0, F)$$

Processing Strings

The DFA starts in q_0 and processes an input string sequentially. The transition function determines the next state. If, after processing the entire string, the DFA reaches a state in F, the string is **accepted**; otherwise, it is **rejected**.

Representations

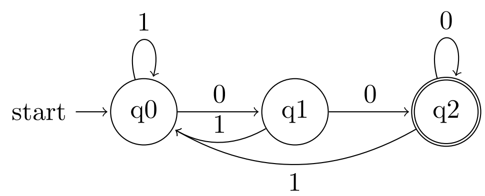
DFAs can be represented in multiple ways:

- **Transition diagrams:** Directed graphs with labeled edges.
- **Transition tables:** Tabular representation of δ.

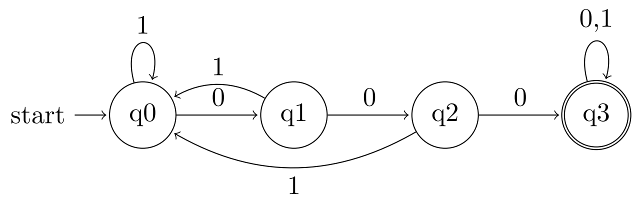
DFAs define **formal languages** by recognizing sets of accepted strings.

3.1 Exercise 2.4.4

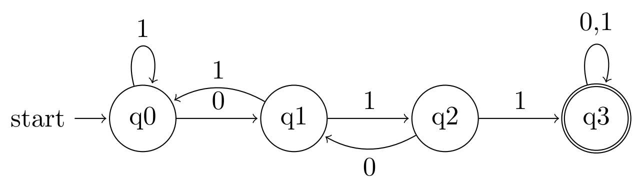
(a) DFA for strings ending in 00



(b) DFA for strings containing 000 as a substring



(c) DFA for strings containing 011 as a substring



4 Conclusion

A **Deterministic Finite Automaton (DFA)** is a theoretical computational model used to recognize formal languages. A DFA consists of a finite set of states Q, an input alphabet Σ, a transition function $\delta : Q \times \Sigma \to Q$, a start state q_0, and a set of accepting states F. The machine processes strings sequentially, transitioning between states according to δ. If, after consuming the entire string, the DFA ends in an accepting state, the input is accepted; otherwise, it is rejected. DFAs can be represented using transition diagrams or transition tables, which illustrate how states change based on input symbols. The concept of an extended transition function allows DFAs to process entire strings iteratively. Examples of DFAs include those recognizing substrings like "01" or enforcing conditions such as even parity of 0s and 1s.

The Python implementation models a DFA using a `DFA` class, which defines the state set, alphabet, transitions, initial state, and accepting states. The `run` method processes input strings and determines acceptance based on the transition function. Additionally, the `refuse` function constructs the complement DFA by inverting the accepting and refusing states, accepting only the words that the original DFA rejects. Several DFAs are defined and tested against a set of generated words, demonstrating their functionality. This implementation provides a practical means of experimenting with DFAs, allowing for the exploration of language recognition, state transitions, and DFA complement operations in a programmatic way.

An Interesting Question: How can the DFA implementation be optimized to handle extremely long input strings efficiently, considering that DFA state transitions form a directed graph with $O(V + E)$ complexity?

References

[1] Hopcroft, J. E., Motwani, R., Ullman, J. D. *Introduction to Automata Theory, Languages, and Computation*. Pearson, 2007.