Deterministic Finite Automata

5 September 2023
Assignment 1 due today

Assignment 1 corrections due on Thursday

Example solutions will be posted after class on Ed
CMPU 240 · Fall 2023

Guide to Assignments

Learning requires doing

A goal for this course is for you to gain experience using models of computation, formal languages, and proofs to solve problems. The homework assignments are essential practice for you to think about the material outside of class and identify what you don’t understand.

In many courses, homework turns into a painful cycle: If you make mistakes on an assignment, turn it in, get your grade back a week and a half later, throw it in a notebook because the class has moved on to other material, and don’t think about those problems again until exam time, you won’t learn from your mistakes, and the time and effort you spent on the homework will have been wasted.

This semester we’re making an effort to do better. Instead of submitting assignments (often late) and then waiting for me to grade and comment on your work after all assignments are in, I will be
Where are we?
Languages are sets of Strings, which are finite sequences of Characters. Alphabets are finite, nonempty sets of Characters.
For example, if $\Sigma = \{a, b\}$,
For example, if $\Sigma = \{a, b\}$,
For example, if $\Sigma = \{a, b\}$,
For example, if $\Sigma = \{a, b\}$,
For example, if $\Sigma = \{a, b\}$,
For example, if $\Sigma = \{a, b\}$,
Check your understanding

True or false:

\[ \Sigma \subseteq \Sigma^*? \]
\[ \varepsilon \in \Sigma? \]
\[ \varepsilon \in \Sigma^*? \]
\[ a = aa? \]
\[ ab = ba? \]
We represent computational problems as languages. For example, the problem of testing whether a number is prime could be treated as the language

\[ \{w \in \{0, \ldots, 9\}^* \mid \text{the number with decimal representation } w \text{ is prime} \} \]
Check your understanding

True or false:

\[ \varepsilon \in abc? \]

\[ \{abc\} \cup \emptyset = \{abc\}? \]

\[ \{abc\} \cup \{\varepsilon\} = \{abc\}? \]
A finite automaton is a collection of states joined by transitions.

Some state is designated as the start state.

Some number of states are designated as accept states.

The automaton processes a string by beginning in the start state and following the indicated transitions.

If the automaton ends in an accept state, it accepts the input.

Otherwise, the automaton rejects the input.

The language of an automaton is the set of strings it accepts.
Designing DFAs
Design strategy for DFAs

At each point in its execution, the DFA can only remember what state it’s in.

Therefore, build each state to correspond to some piece of information that you need to remember.

Each state acts as an indicator of what you’ve already seen, sufficient to let you decide what to do next.

There can only be finitely many states, so the DFA can only remember finitely many things.
Example

Consider the language

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring} \} \]

How can we design a DFA to recognize \( L \)?
$L = \{w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring}\}$
\[ L = \{w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring}\} \]
\[ L = \{w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring} \} \]
\[ L = \{w \in \{0, 1\}^* \mid w \text{ contains 11 as a substring}\} \]
\[ L = \{ w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring} \} \]
\( L = \{w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring}\} \)
$L = \{w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring}\}$
Example

```c
#include <stdio.h>

/* print Fahrenheit-Celsius table
   for fahr = 0, 20, ..., 300; floating-point version */
main()
{
    float fahr, celsius;
    int lower, upper, step;

    lower = 0;    /* lower limit of temperature table */
    upper = 300;  /* upper limit */
    step = 20;    /* step size */

    fahr = lower;
    while (fahr <= upper) {
        celsius = (5.0/9.0) * (fahr-32.0);
        printf("%3.0f %6.1f\n", fahr, celsius);
        fahr = fahr + step;
    }
}
```
Example

C-style comment

```c
#include <stdio.h>

/* print Fahrenheit-Celsius table
   for fahr = 0, 20, ..., 300; floating-point version */
main()
{
    float fahr, celsius;
    int lower, upper, step;

    lower = 0;        /* lower limit of temperature table */
    upper = 300;      /* upper limit */
    step = 20;        /* step size */

    fahr = lower;
    while (fahr <= upper) {
        celsius = (5.0/9.0) * (fahr-32.0);
        printf("%3.0f %6.1f\n", fahr, celsius);
        fahr = fahr + step;
    }
}
```
Example

Consider the language

\[ L = \{ w \in \{a, *, /\}^* \mid w \text{ represents a C-style comment} \} \]

We’re using the symbol \( a \) as a placeholder for any character that isn’t a star or slash (including spaces) to keep things simple.
Example

Just like when you’re programming, it helps to come up with sets of test cases to accept and reject.

\[ L = \{ w \in \{a, *, /\}^* \mid w \text{ represents a C-style comment}\} \]
\[ L = \{w \in \{a, *, /\}^* \mid w \text{ represents a C-style comment}\} \]
If a machine can’t remember all the symbols it has seen so far in an input string, it has to change state based on other information, e.g.,

$L_1 = \text{the set of all strings with an odd number of } 1\text{s over } \Sigma = \{0, 1\}$

Don’t need to remember exactly how many 1s have been seen – just whether we’ve read an even or odd number.
Exercise

Build an automaton to recognize the set of strings that end with *ing*.
Exercise

Design a finite automaton to recognize decimal numbers.
Formal DFAs
Formal definition of a deterministic finite automaton (DFA)

A DFA is represented as a five-tuple \((Q, \Sigma, \delta, q_0, F)\) where

- \(Q\) is a finite set of states,
- \(\Sigma\) is the alphabet, a finite set of input symbols,
- \(\delta: Q \times \Sigma \rightarrow Q\) is the transition function,
- \(q_0 \in Q\) is the start state, and
- \(F \subseteq Q\) is a set of zero or more accept states.
Transition function $\delta$

Takes two arguments: a state and an input symbol.

$\delta(q, a) = \text{the state the DFA goes to when it is in state } q \text{ and reads input symbol } a.$

Since $\delta$ is a total function; there is always a next state.

If there's no transition you want, you must add a “dead state”.
Formal description of how finite automata compute

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.

Let $w$ be a string, $w_1w_2\ldots w_n$, where each $w_i \in \Sigma$.

$M$ accepts $w$ if a sequence of states $r_0, r_1, \ldots, r_n$ exists where each $r_i \in Q$ and

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, \ldots, n-1$
3. $r_n \in F$
Formal description of how finite automata compute

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.

Let $w$ be a string, $w_1w_2\ldots w_n$, where each $w_i \in \Sigma$.

$M$ accepts $w$ if a sequence of states $r_0, r_1, \ldots, r_n$ exists where each $r_i \in Q$ and

1. $r_0 = q_0$

2. $\delta(r_i, w_{i+1}) = r_{i+1}$
   for $i = 0, \ldots, n-1$

3. $r_n \in F$

*It begins at the start state, \ itends at the accept state.*
Formal description of how finite automata compute

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.

Let $w$ be a string, $w_1w_2\ldots w_n$, where each $w_i \in \Sigma$.

$M$ accepts $w$ if a sequence of states $r_0, r_1, \ldots, r_n$ exists where each $r_i \in Q$ and

1. $r_0 = q_0$

2. $\delta(r_i, w_{i+1}) = r_{i+1}$
   for $i = 0, \ldots, n-1$

3. $r_n \in F$

It begins at the start state, each transition is allowed by the transition function for the corresponding input symbol, and
Formal description of how finite automata compute

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.

Let $w$ be a string, $w_1w_2\ldots w_n$, where each $w_i \in \Sigma$.

$M$ accepts $w$ if a sequence of states $r_0, r_1, \ldots, r_n$ exists where each $r_i \in Q$ and

1. $r_0 = q_0$
   \hspace{1cm} It begins at the start state,

2. $\delta(r_i, w_{i+1}) = r_{i+1}$
   \text{ for } i = 0, \ldots, n-1
   \hspace{1cm} each transition is allowed by the transition function for the corresponding input symbol, and

3. $r_n \in F$
   \hspace{1cm} it ends in an accept state.
Example

What’s the formal definition of this DFA?
What’s the formal definition of this DFA?
Example

What’s the formal definition of this DFA?
Example

What’s the formal definition of this DFA?

$$M_{\text{news}} = (Q, \Sigma, \delta, q_0, F)$$
Example

What’s the formal definition of this DFA?

\[ M_{\text{news}} = (Q, \Sigma, \delta, q_0, F) \]

\[ Q = \{q_0, q_5, q_{10}, q_{15}, q_{20}, q_{25}\} \]
\[ M_{\text{news}} = (Q, \Sigma, \delta, q_0, F) \]

- \( Q = \{q_0, q_5, q_{10}, q_{15}, q_{20}, q_{25}\} \)
- \( \Sigma = \{n, d, q\} \)

**Example**

What’s the formal definition of this DFA?
Example

What's the formal definition of this DFA?

$M_{\text{news}} = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_0, q_5, q_{10}, q_{15}, q_{20}, q_{25}\}$

$\Sigma = \{n, d, q\}$

$\delta(q_0, n) = q_5$
Example

What's the formal definition of this DFA?

Mathematically, the DFA is defined as:

$$M_{\text{news}} = (Q, \Sigma, \delta, q_0, F)$$

where:

- $Q = \{q_0, q_5, q_{10}, q_{15}, q_{20}, q_{25}\}$
- $\Sigma = \{n, d, q\}$

The transition function $\delta$ is defined as:

- $\delta(q_0, n) = q_5$
- $\delta(q_0, d) = q_{10}$

The diagram visually represents the transitions and states of the DFA.
Example

What’s the formal definition of this DFA?

\[ M_{\text{news}} = (Q, \Sigma, \delta, q_0, F) \]

\[ Q = \{q_0, q_5, q_{10}, q_{15}, q_{20}, q_{25}\} \]

\[ \Sigma = \{n, d, q\} \]

\[ \delta(q_0, n) = q_5 \]
\[ \delta(q_0, d) = q_{10} \]
\[ \delta(q_0, q) = q_{25} \]
\[ \delta(q_5, n) = q_{10} \]
\[ \delta(q_5, d) = q_{15} \]
\[ \delta(q_5, q) = q_{25} \]
\[ \vdots \]

\[ F = \{q_{25}\} \]
Example

What's the formal definition of this DFA?

\[ M_{\text{news}} = (\{q_0, q_5, q_{10}, q_{15}, q_{20}, q_{25}\}, \{n, d, q\}, \delta, q_0, \{q_{25}\}) \]

\[ \delta(q_0, n) = q_5 \]
\[ \delta(q_0, d) = q_{10} \]
\[ \delta(q_0, q) = q_{25} \]
\[ \delta(q_5, n) = q_{10} \]
\[ \delta(q_5, d) = q_{15} \]
\[ \delta(q_5, q) = q_{25} \]
\[ \vdots \]
Tabular DFAs
Another way we can write down the transition function for a DFA is as a transition table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_0$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_0$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_2$</td>
<td>$q_2$</td>
</tr>
</tbody>
</table>
Another way we can write down the transition function for a DFA is as a transition table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_0$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_0$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$*$</td>
<td>$q_2$</td>
<td>$q_2$</td>
</tr>
</tbody>
</table>

By marking the start state with $\rightarrow$ and accept states with $*$, the transition table that defines $\delta$ also specifies the entire DFA!
Tabular DFAs suggest how easy it is to implement a DFA in software.

```python
transition_table = {
    "q0": {"0": "q0", "1": "q1"},
    "q1": {"0": "q0", "1": "q2"},
    "q2": {"0": "q2", "1": "q2"}
}

accept_states = ["q2"]

def run_dfa(word: str) -> bool:
    state = "q0"
    for char in word:
        state = transition_table[state][char]
    return state in accept_states
```
Regular languages
Regular languages

DEFINITION A language $L$ is called a *regular language* if there exists a DFA $D$ such that $L(D) = L$.

If $L$ is a language and $L(D) = L$, we say that $D$ *recognizes* the language $L$. 
Acknowledgments

This lecture incorporates material from:

- David Chiang, University of Notre Dame
- Nancy Ide, Vassar College
- Keith Schwarz, Stanford University
- Michael Sipser, *Introduction to the Theory of Computation*