Finite Automata

DFA, NFA

A model of computation

In this lecture we study a simple kind of computers (abstract)

Imagine a computer (a machine) that is programmed with only one program (does only one job – e.g. a washing machine)

Since such a computer has only one program, we will abstractly think of that computer and the program on it as the same thing

A simple programming language

Consider the following program (or computer), which we will explain it in a bit

$$q_0 b q_1$$
$$q_0 a q_2$$
$$q_1 b q_1$$
$$q_1 a q_1$$
$$q_2 b q_3$$
$$q_2 a q_2$$
$$q_3 b q_3$$
$$q_3 a q_2$$

The syntax here is $q_i x q_j$ where $i, j \in \mathbb{N}$ and $x \in \Sigma$ (some fixed alphabet)

The brown coloring is to indicate that the state is a *final state* (we will explain what that means)

How to interpret the syntax?

The program on the previous slide represents a machine that has four possible states: q_0, q_1, q_2, q_3

That machine can take as an input any string made of the symbols a, b. We say $\Sigma = \{a, b\}$ is the machine's alphabet

The program on the previous slide is saying the following:

The machine starts at state q_0. If the machine is given a string from Σ^*, it will start reading from left to right and does the following:

Line 1: If it reads b, then switch to state q_1 and move to the right (one step to read the next symbol)

Line 2: If it reads a, then switch to state q_2 and move to the right

Now the machine is either in state q_1 or state q_2 , and the later lines in the program show how each state will act

Line 3: If the state is q_1 and the symbol read is b, keep the sate unchanged and move to the right

Line 4: If the state is q_1 and the symbol read is a, keep the sate unchanged and move to the right

Line 5: If the state is q_2 and the symbol read is b, then switch to state q_3 and move to the right

Line 6: If the state is q_2 and the symbol read is a, keep the sate unchanged and move to the right

Line 7:

Line 8:

So, what does the given machine (program) do?

Example 1: Suppose the input string is $ababb$

The red represents the symbol being read

$$q_0: \mathbf{a}babb$$
$$q_2: a\mathbf{b}abb$$
$$q_3: ab\mathbf{a}bb$$
$$q_2: aba\mathbf{b}b$$
$$q_3: abab\mathbf{b}$$
$$q_3: ababb$$

Done reading all symbols ✓

Finishing in a final state ✓

What happened?

The machine read the entire input and went to a final state. In that case, we will say the machine *accepts* the input string

Example 2: $ababa$

$q_0: ababa$
$q_2: ababa$
$q_3: ababa$
$q_2: ababa$
$q_3: ababa$
$q_2: ababa$

Done reading all symbols ✓

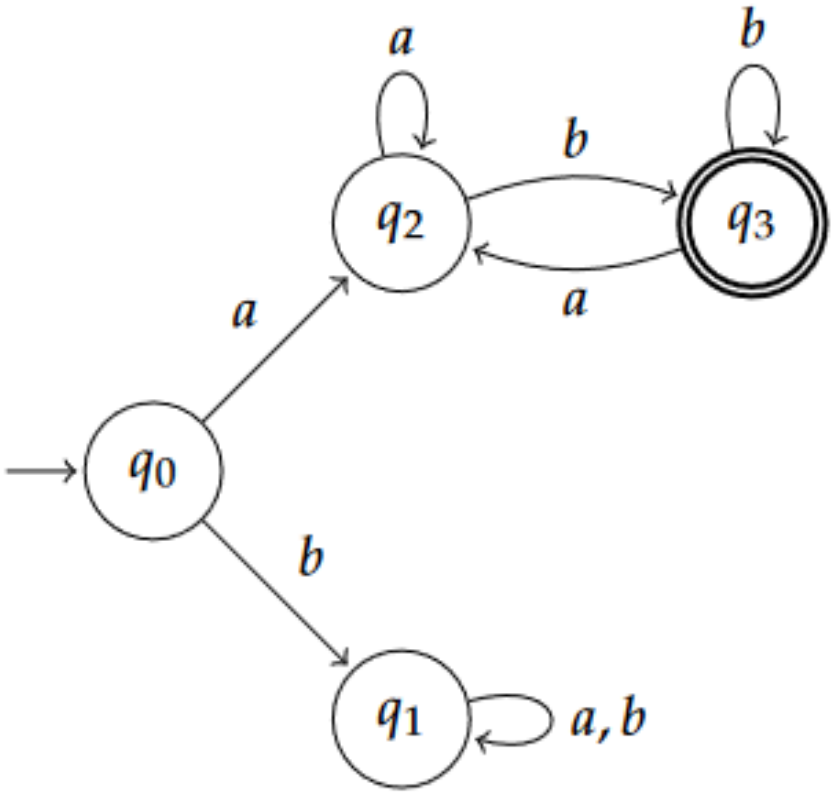
Finishing in a final state ✗

What happened in Example 2 ?

The machine *rejected* the input string

We know that because it finished reading the input and went into a state that is not final

The machine/computer/program can be represented as a flowchart



Or as a table

δ	a	b
q_0	q_2	q_1
q_1	q_1	q_1
q_2	q_2	q_3
q_3	q_2	q_3

The Transition Function

The table on the previous slide represents a function δ

$$\delta: \{q_0, q_1, q_2, q_3\} \times \Sigma \rightarrow \{q_0, q_1, q_2, q_3\}$$

This function is essentially the program. It guides the machine.

A Formal Generalization

The kind of program/machine/computer we described can be defined by its 5 components:

The alphabet Σ

The set of states Q

The initial state (it belongs to Q, and normally we denote it as s or q_0)

The set of final states $F \subseteq Q$

The transition function $\delta: Q \times \Sigma \to Q$

And we call such a program a *DFA* (*Deterministic Finite Automaton*)

Determinism

The kind of program we described has deterministic behavior because every line can only lead to one line

If we run a DFA on the same input multiple times it will give the same answer every time

The transition function δ **determines** what the transition will be from a particular state on a particular symbol

Nondeterminism

Consider now the following manipulation of our previously given program. We added a new command (acceptable syntax) which tells the machine if the state is q_2 reading b then move to the right and remain in state q_1

$$q_0 b q_1$$
$$q_0 a q_2$$
$$q_1 b q_1$$
$$q_1 a q_1$$
$$q_2 b q_3$$
$$q_2 a q_2$$
$$q_3 b q_3$$
$$q_3 a q_2$$
$$q_2 b q_1$$

Now the machine has two options for the transition after q_1 reading b. The machine can either transition to state q_3 or to state q_1. The δ is no longer a function :$Q \times \Sigma \to Q$. This kind of program/machine/computer is called an *NFA* (*Nondeterministic Finite Automaton*)

A Formal Generalization (NFA)

The kind of program/machine/computer we described can be defined by its 5 components:

The alphabet Σ

The set of states Q

The initial state (it belongs to Q, and normally we denote it as s or q_0)

The set of final states $F \subseteq Q$

The transition function $\delta: Q \times \Sigma \to \mathcal{P}(Q)$

(or, think of δ as a transition relation $\delta \subseteq (Q \times \Sigma) \times Q$)

And we call such a program a *NFA* (*Non-deterministic Finite Automaton*)

Remark

The word finite comes from the fact that the set of states is finite.

Other synonyms to a *finite automaton* (plural: *automata*) : *Finite-state machine* (*FSM*), *Finite-state automaton* (*FSA*)

Notice that finite automata have no memory for the symbols they read (they just read and move on in one direction).

A Turing machine (just for culture)

A finite state machine that is equipped with a memory (tape) which keeps the string and can move the reading head back (left) and forth (right) is known as a Turing machine

Turing programs can be written using lines looking as follows

$$q_i a q_j b D$$

Such a line means that if the machine is reading a at state q_i, it will then overwrite a with b, transition into state q_j, then make a move in the direction $D = L\ (left) or\ R(right)$

Turing machines are more powerful than finite automata. In fact, Turing machines can do anything a modern computer can do.

Connection with Regular Languages

Regular languages are exactly the languages acceptable by some finite automaton (deterministic or non-deterministic)

More precisely, given an alphabet Σ and an arbitrary language L over Σ, we have the following:

L is regular

iff

There exists a DFA over Σ that accepts L

iff

There exists an NFA over Σ that accepts L

Recall: An FA M accepts a language L means that $L = \{w \in \Sigma^* : M \text{ accepts } w\}$.

Example

The program on slide 3 accepts a string iff it starts with a and ends with b (How can you prove this?)

Conclusion:

The language $\{awb: w \in \{a, b\}^*\}$ is a regular language over the alphabet $\{a, b\}$.

Correctness of DFAs

A helpful notation

The transition δ has domain $Q \times \Sigma$, but we can naturally extend the definition to $Q \times \Sigma^*$ recursively as follows:

$$\delta(q, \epsilon) = q$$
$$\delta(q, wa) = \delta(\delta(q, w), a)$$

This recursive definition leads to the following (can be proved by induction):

$$\delta(q, w) = \text{the state the DFA reaches if it starts in state } q \text{ on input } w$$

For example:

In our program from slide 3, $\delta(q_0, ababb) = q_3$, $\delta(q_0, ababa) = q_2$, $\delta(q_2, bb) = q_3$, $\delta(q_1, bb) = q_1$

We should be using a new symbol for this new δ, but the textbook does not. We will abuse notation the same way the book is doing, no big harm.

State Invariants

Let M be some DFA on an alphabet Σ. Let q be a state which M can take. Let s be the starting state for M.

A *state invariant* for q is a predicate $P_q(x)$ over Σ^* (i,.e. x can be any string over Σ) such that: $\forall w \in \Sigma*$

$$P_q(w) \text{ is true}$$
$$\text{iff}$$
$$M \text{ when run on input } w, M \text{ finishes in state } q \text{ (i.e. } \delta(s, w) = q)$$

A state invariant for a state q specifies exactly which inputs will cause the machine to reach the state q (what can we tell the machine so it gives us a certain state)

About State Invariants

Notice that the definition of a state invariant implies the following two properties:

1. $q \neq q' \Rightarrow \{w: P_q(w)\} \cap \{w: P_{q'}(w)\} = \emptyset$ (mutually exclusive).

No string w can satisfy more than one state invariant

Another way to write the mutual exclusivity property:

$$q \neq q' \Rightarrow \forall w \in \Sigma*, \neg(P_q(w) \wedge P_{q'}(w))$$

2. $\bigcup_{q \in Q}\{w: P_q(w)\} = \Sigma^*$ (*Exhaustive)*

Every string w must satisfy one of the invariants

Another way to write the exhaustiveness property:

$$\forall w \in \Sigma^*, \exists q \in Q, P_q(w)$$

Proving Correctness of a DFA using State Invariants

Firstly, for every state q, **suggest** a predicate $P_q(x)$ to be a state invariant. It should be natural at this first step to make sure the predicates are mutually exclusive and exhaustive

Secondly, show that $\forall q \in Q,\ P_q(x)$ is indeed a state invariant (basically prove your suggestions are valid). The following is some strategy to do so (induction)

1. Prove $P_s(\epsilon)$ (i.e., prove that $P_s(\epsilon)$ is true)
2. Prove that: $\forall q \in Q, \forall w \in \Sigma^*, \forall z \in \Sigma, P_q(w) \Longrightarrow P_{\delta(q,z)}(wz)$

Why does this strategy work? The answer is explained on the next 3 slides.

Thirdly (and finally), prove that: $(\forall w \in \Sigma*)[w \in L \Leftrightarrow \exists q \in F, P_q(w)]$, which means that the DFA accepts exactly L.

Why 1,2 imply that $\forall q \in Q,\ P_q(x)$ is indeed a state invariant?

Notice that 2 implies the following,

$$(\forall z \in \Sigma)[P_s(\epsilon) \Longrightarrow P_{\delta(s,z)}(z)]$$

Since from 1, $P_s(\epsilon)$ is true, we can conclude that $P_{\delta(s,z)}(z)$ for every $z \in \Sigma$.

Again, from 2, for every q immediately reachable from the starting state, and every $w \in \Sigma^*$ of length 1:

$$(\forall z \in \Sigma)[P_q(w) \Longrightarrow P_{\delta(q,z)}(wz)]$$

By the same thinking above, this implies that $P_q(w)$ is true for every state q reachable after two transitions from the starting state by reading a string $w \in \Sigma^*$ of length 2

Conclusion

1,2 imply the following for any alphabet symbols $w_1, w_2, w_3, \ldots$:

$$P_s(\epsilon) \Longrightarrow P_{\delta(s,w_1)}(w_1) \Longrightarrow P_{\delta(\delta(s,w_1),w_2)}(w_1 w_2) \Longrightarrow P_{\delta(\delta(\delta(s,w_1),w_2),w_3)}(w_1 w_2 w_3) \Rightarrow \cdots \Rightarrow P_{\delta(s,w)}(w)$$

Which tells us that, for every $w \in \Sigma^*$, $P_{\delta(s,w)}(w)$

Which is exactly the following:

$$(\forall q \in Q)(\forall w \in \Sigma*)[\delta(s,w) = q \Rightarrow P_q(w) \text{ is true}]$$

Does this say that $\forall q \in Q, P_q(x)$ is a state invariant for q ?

Not yet, we still need $(\forall q \in Q)(\forall w \in \Sigma*)[\delta(s,w) \neq q \Rightarrow P_q(w) \text{ is false}]$

$\delta(s,w) \neq q \Rightarrow P_q(w)$ is false

Your design of $P_q(x)$ at the very beginning should imply directly that they are mutually exclusive in a clear way, which should lead to the implication above instantly

More clearly, if $\delta(s,w) \neq q$, then $\delta(s,w) = q'$, where q' is a different state from q. We proved that $(\forall q \in Q)(\forall w \in \Sigma*)\big[\delta(s,w) = q \Rightarrow P_q(w) \text{ is true}\big]$. Therefore, $P_{q'}(w)$ is true. Since by mutual exclusivity it is impossible that $P_q(x)$ and $P_{q'}(w)$ are true at the same time, we must have that $P_q(w)$ is false.

Examples

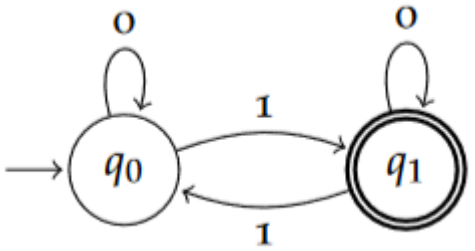
On proving correctness using state invariants

Example 1
(textbook example on p. 75 and 76)

Prove that the language $L = \{w \in \Sigma^* : w \text{ has an odd number of } 1s\}$ is a regular language over the alphabet $\Sigma = \{0,1\}$.

Proof: We will define a DFA and show that it accepts exactly the strings in L.

Consider the following DFA:



Example 1 continued

Which can be written as

$$q_0 0 q_0$$
$$q_0 1 q_1$$
$$q_1 0 q_1$$
$$q_1 1 q_0$$

where q_0 is the starting state and the set of final states $\{q_1\}$.

How did we think of this DFA?

Intuition: If the number of 1s in the string is odd, then the state will end up q_1. If the number of 1s is even, the machine will end up in its starting state.

Example 1 continued (intuition)

Consider the following input 010110

The machine will take the following configurations to complete reading this input

$$q_0: \mathbf{0}10110$$
$$q_0: 0\mathbf{1}0110$$
$$q_1: 01\mathbf{0}110$$
$$q_1: 010\mathbf{1}10$$
$$q_0: 0101\mathbf{1}0$$
$$q_1: 01011\mathbf{0}$$
$$q_1: 010110$$

Observation: the state flips if 1 is read

An even number of flips, will keep the sate as the same as the start

An odd number of flips, will make the state the opposite to how it started

Example 1 continued

Let us suggest state invariants. The following choice is natural

$$P_{q_1}(x)\colon x \text{ has an odd number of } 1\text{s}$$

We are forced now to have $P_{q_0}(x)\colon x$ has an even number of 1s

It should be obvious that these work, but let us formally continue the proof that they are actually state invariants

Example 1 continued

1) $P_{q_0}(\epsilon)$ is true.

Indeed, the number of 1s in ϵ (the empty string) is 0, which is even.

2) Now we want to show that:

$$\forall q \in Q, \forall w \in \Sigma^*, \forall z \in \Sigma, P_q(w) \Longrightarrow P_{\delta(q,z)}(wz)$$

Let $q \in Q$, $w \in \Sigma^*$, and $z \in \Sigma$ be arbitrary. Assume that $P_q(w)$ is true. We want to show that $P_{\delta(q,z)}(wz)$ is true.

Case $q = q_0$:

$P_{q_0}(w)$ is true means, by the definition of P_{q_0} , that w has an even numbers of 1s.

Subcase $z = 0$:

We have by the definition of the DFA that $\delta(q_0, 0) = q_0$. So, $P_{\delta(q,z)}(wz) = P_{q_0}(w0)$ which is true. It is true because 1) the number of 1s in $w0$ is the same as in w, which is even. 2) the definition of P_{q_0} .

Example 1 continued

Subcase $z = 1$:

We have by the definition of the DFA that $\delta(q_0, 1) = q_1$. So, $P_{\delta(q,z)}(wz) = P_{q_1}(w1)$ which is true.

It is true because

1) the number of 1s in $w1$ is odd as it is 1+ the number of 1s in w.
2) the definition of P_{q_1} ($P_{q_1}(x)$ is true when x has odd number of 1s)

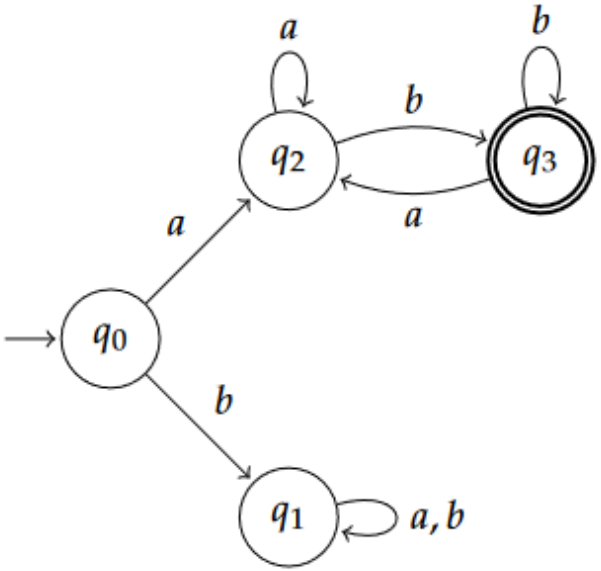
Case $q = q_1$: Similar to the previous case.

Example 2

Prove that the language $L = \{awb : w \in \{a, b\}^*\}$ is a regular language over the alphabet $\Sigma = \{a, b\}$.

We will show that the DFA we have on slide 3 accepts exactly the strings of L.

Recall, the DFA was:



Example 2 continued

We want to prove the following for an arbitrary $w \in \{a, b\}^*$:

$$w \text{ starts with } a \text{ and ends with } b \iff \delta(q_0, w) = q_3$$

The following are natural state invariants to suggest:
Tracing the graph is helpful here

$P_{q_3}(x)$: x starts with $a \wedge x$ ends with b (*why did I start with this one?)*
$P_{q_2}(x)$: x starts with $a \wedge x$ ends with a
$P_{q_1}(x)$: x starts with $b \wedge (x$ ends with $a \vee x$ ends with $b)$
$P_{q_0}(x)$: x is empty

Example 2 continued

Remark: $P_{q_1}(x)$ can be simplified to be $P_{q_1}(x)$: x starts with b

Do you see why? It should be obvious but here is a full reasoning

x starts with b

iff

x starts with $b \wedge$ TRUE

iff

x starts with $b \wedge (x$ ends with $a \vee x$ ends with $b \vee x$ is empty)

iff

(x starts with $b \wedge x$ ends with $a) \vee (x$ starts with $b \wedge x$ ends with $b) \vee (x$ starts with $b \wedge x$ is empty)

iff

(x starts with $b \wedge x$ ends with $a) \vee (x$ starts with $b \wedge x$ ends with $b) \vee$ FALSE

iff

(x starts with $b \wedge x$ ends with $a) \vee (x$ starts with $b \wedge x$ ends with $b)$

iff

x starts with $b \wedge (x$ ends with $a \vee x$ ends with $b)$

Example 2 continued

So, the following are our suggestions for state invariants

$P_{q_0}(x)$: x is empty

$P_{q_1}(x)$: x starts with b

$P_{q_2}(x)$: x starts with $a \wedge x$ ends with a

$P_{q_3}(x)$: x starts with $a \wedge x$ ends with b

Sanity checks:

Are they mutually exclusive? Obvious

Are they exhaustive? Obvious as well but we will clarify it

Proof of exhaustiveness

If x is not empty, then $P_{q_0}(x)$ is not true.

Otherwise, if x is not empty, then it must start with a symbol and end with a symbol.

It can start with a or b, no other choice.

If it does not start with a, i.e. starts with b, then $P_{q_1}(x)$ is true.

Otherwise, it starts with a, and ends with a or b.

If it ends with a, then $P_{q_2}(x)$ is true

Otherwise, it ends with b, and then $P_{q_3}(x)$ is true

Example 2 continued

Now let us prove that

1. $P_{q_0}(\epsilon)$ is true
2. $\forall q \in \{q_0, q_1, q_2, q_3\}, \forall w \in \{a, b\}^*, \forall z \in \{a, b\},$
$$P_q(w) \Longrightarrow P_{\delta(q,z)}(wz)$$

$P_{q_0}(\epsilon)$ is true by the definition of P_{q_0}

Example 2 continued

Now let $w \in \{a, b\}^*$ be arbitrary.

Case $q = q_0$ and $z = a$:

We have $\delta(q, z) = \delta(q_0, a) = q_1$

So, we want to show that if $P_{q_0}(w)$ is true, then $P_{q_2}(wa)$ is true.

Assume $P_{q_0}(w)$ is true. Then, $w = \epsilon$. So, $wa = a$ which starts with a and ends with a. So, $P_{q_2}(wa)$ is true (by the definition of P_{q_2}).

Example 2 continued

Case $q = q_0$ and $z = b$:

We have $\delta(q, z) = \delta(q_0, b) = q_1$

So, we want to show that if $P_{q_0}(w)$ is true, then $P_{q_1}(wb)$ is true.

Assume $P_{q_0}(w)$ is true. Then, $w = \epsilon$. So, $wb = b$ which starts with b.
So, $P_{q_1}(wb)$ is true (by the definition of P_{q_1}).

Example 2 continued

Case $q = q_1$ and $z = a$:

We have $\delta(q, z) = \delta(q_1, a) = q_1$

So, we want to show that if $P_{q_1}(w)$ is true, then $P_{q_1}(wa)$ is true.

Assume $P_{q_1}(w)$ is true. Then, w starts with b. So, wa also starts with b.
So, $P_{q_1}(wa)$ is true (by the definition of P_{q_1}).

Case $q = q_1$ and $z = b$:

The exact same conclusion as the previous case because w and wb also start with b.

Example 2 continued

Case $q = q_2$ and $z = a$:

We have $\delta(q, z) = \delta(q_2, a) = q_2$

So, we want to show that if $P_{q_2}(w)$ is true, then $P_{q_2}(wa)$ is true.

Assume $P_{q_2}(w)$ is true. Then, w starts with a and ends with a. So, wa also starts with a and ends with a. So, $P_{q_2}(wa)$ is true (by the definition of P_{q_2}).

Case $q = q_2$ and $z = b$:

We have $\delta(q, z) = \delta(q_2, b) = q_3$

So, we want to show that if $P_{q_2}(w)$ is true, then $P_{q_3}(wa)$ is true.

Assume $P_{q_2}(w)$ is true. Then, w starts with a and ends with a. So, wb starts with a and ends with b. So, $P_{q_3}(wb)$ is true (by the definition of P_{q_3}).

Example 2 continued

Case $q = q_3$ and $z = a$:

We have $\delta(q, z) = \delta(q_3, a) = q_2$

So, we want to show that if $P_{q_3}(w)$ is true, then $P_{q_2}(wa)$ is true.

Assume $P_{q_3}(w)$ is true. Then, w starts with a and ends with b. So, wa starts with a and ends with b. So, $P_{q_2}(wa)$ is true (by the definition of P_{q_2}).

Case $q = q_3$ and $z = b$:

We have $\delta(q, z) = \delta(q_3, b) = q_3$

So, we want to show that if $P_{q_3}(w)$ is true, then $P_{q_3}(wb)$ is true.

Assume $P_{q_3}(w)$ is true. Then, w starts with a and ends with b. So, wb starts with a and ends with b. So, $P_{q_3}(wb)$ is true (by the definition of P_{q_3}).

Example 2 continued

Finally, we prove that $w \in L \Leftrightarrow \ P_{q_3}(w)$

Obvious by our definition of P_{q_3}.

Another approach to DFA correctness

Inspired by loop invariants

Similar to Loop Invariants

Notice that a DFA acts as an iterative program (parses a string)

How do we prove correctness of iterative programs?

The equivalent of a loop invariant here is a predicate $P(j)$ where $j \in \{1, \dots, |w| + 1\}$ is the iteration index assuming the input is w.

Let $q(j)$ be the state of the machine the moment it is reading the jth symbol in the input string. Notice that $q(j) = \delta(s, w_1 \dots w_{j-1})$.

Remark 1: when $j = 1, w_1 \dots w_{j-1} = \epsilon$.

Remark 2: when $j = |w| + 1$, the machine is done reading the input w.

Remark 3: $q(1) = s,\ q(|w| + 1) = \delta(s, w)$.

Induction on iterations

Suppose now you want to prove that the DFA accepts exactly some language L.

You can proceed as follows:

Let $w \in \Sigma^*$ be arbitrary.

Prove the following is true for all $j \in \{1, \dots, |w| + 1\}$:

$$w_1 \dots w_{j-1} \in L \Leftrightarrow q(j) \in F$$

Surprise: The loop invariant (or better call it: *iteration invariant*) is the following predicate

$$P(j)\colon\ w_1 \dots w_{j-1} \in L \Leftrightarrow q(j) \in F$$

Applying the new approach to Example1

Let us attempt proving the correctness of the DFA in Example 1 using that new approach.

Base case $j = 1$:
We have $q(1) = q_0$. So,

$$P(1)\text{: } \epsilon \in L \Leftrightarrow q_0 \in F$$

$P(1)$ is true because $\epsilon \in L$ is false and $q_0 \in F$ is also false.

I.H.: Assume that $P(j)$ is true for some $j \in \{1, \dots, |w|\}$.
Which means that

$$w_1 \dots w_{j-1} \in L \Leftrightarrow q(j) \in F$$

Or more clearly,

$$w_1 \dots w_{j-1} \text{ has an odd number of 1s iff } q(j) = q_1$$

Applying the new approach to Example 1

Now we want to prove that $P(j+1)$ is true. I.e.,

$w_1 \dots w_j$ has an odd number of 1s iff $q(j+1) = q_1$

Notice that: the number of 1s in $w_1 \dots w_j$ = the number of 1s in $w_1 \dots w_{j-1}$ + w_j

Case $w_j = 0$: In this case $P(j+1)$ becomes

$w_1 \dots w_{j-1}$ has an odd number of 1s iff $q(j+1) = q_1$

We know from the I.H. that $w_1 \dots w_{j-1}$ has an odd number of 1s iff $q(j) = q_1$. So, here we need to show that

$q(j) = q_1$ iff $q(j+1) = q_1$

In other words, we need to show that $q(j) = q(j+1)$.

Indeed, $q(j+1) = \delta(s, w_1 \dots w_j) = \delta(s, w_1 \dots w_{j-1}0) = \delta(\delta(w_1 \dots w_{j-1}), 0) = \delta(q(j), 0) = q(j)$.

Applying the new approach to Example 1

Case $w_j = 1$: In this case $P(j+1)$ becomes

$w_1 \dots w_{j-1}$ has an even number of 1s iff $q(j+1) = q_1$

We know from the I.H. that $w_1 \dots w_{j-1}$ has an even number of 1s iff $q(j) = q_0$. So, here we need to show that

$$q(j) = q_0 \text{ iff } q(j+1) = q_1$$

In other words, we need to show that $q(j) \neq q(j+1)$.

Indeed, $q(j+1) = \delta(s, w_1 \dots w_j) = \delta(s, w_1 \dots w_{j-1}1) = \delta(\delta(w_1 \dots w_{j-1}), 1) = \delta(q(j), 1) \neq q(j)$.

Proving non-regularity

In this part we comment on pages 77,78 of the textbook

You probably have studied this part on your own already to solve A3.

Proving non-regularity

1) We mentioned the Pumping Lemma but did not discuss it in class as the textbook uses a different approach
2) The textbook's approach is to show that: For every positive natural k, the language cannot be accepted by a DFA with k states. This clearly implies that no DFA accepts the language, whence it is non-regular.

Example

Consider the language $L = \{0^n 1^n : n \in \mathbb{N}\}$. Prove that no DFA accepts (exactly) L.

Proof: Assume towards a contradiction that there is a DFA D which accepts L. Let k be the size of D (i.e., k is the number of states in D).

Consider the following states $\delta(s, 0^i), 1 \leq i \leq k+1$, where s is the starting state of D.

Since D has only k states, it must be the case that for some $i \neq j, i, j \in \{1, \dots, k+1\}$ that $\delta(s, 0^i) = \delta(s, 0^j)$.

This is obvious (the pigeonhole principle).

Now we have that $\delta(s, 0^i 1^j) = \delta(s, 0^j 1^j)$. This immediately gives a contradiction (why?)

Example (contradiction explained)

We assumed at the beginning that our DFA D accepts L.

So, we must have D accept $0^j 1^j$ and reject $0^i 1^j$, which means

$$\delta(s, 0^i 1^j) \neq \delta(s, 0^j 1^j).$$

In general

The textbook's idea is to:

1. Assume the number of states is k (arbitrary)
2. Find a loop of in the DFA (maybe graphing the DFA helps) within $\leq k + 1$ transitions by pigeonhole
3. The loop should inspire you to find different sequences $w \in L$ and $w' \notin L$ leading to the same state (i.e. $\delta(s, w) = \delta(s, w')$).
4. Finding two such strings presents a contradiction