Computer Science 351

DFA Design and Verification — Part One

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #3

Goals for Today

Goals for Today:

- *Beginning* of a presentation of a process that can be used to ***design*** a deterministic finite automaton that has a given language.

A “Longer Term” Goal

A Longer Term Goal:

- This is one of many ***processes***, that can be followed to complete various tasks, that you will learning about.
- ***Good News:*** Because deterministic finite automata are reasonably *simple*, this design process is reasonably easy to understand and use — so that can focus on *learning how to follow a process that is being introduced in a course* in a reasonably simple setting.

  This might make things easier — because you have already figured out how to do things — later on, when more complicated processes are being introduced.

A “Longer Term” Goal

- ***Not-So-Good News:*** Because this process (and problems solved, using it) is so simple, it can be tempting to skip over it!

  However, this might result in your falling behind — because you have so much more to learn — later on, when more complicated processes *are* being introduced.

DFA Design

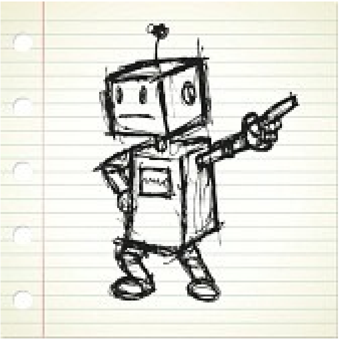
Good Advice from "Introduction to the Theory of Computation:"

> "Whether it be automata or artwork, ***design is a creative process.*** As such it cannot be reduced to a simple recipe or formula. However, you might find a particular approach helpful when designing various types of automata. That is, put *yourself* in the place of the machine you are trying to design and then see how you would go about performing the machine's task. ***Pretending that you are the machine is a psychological trick that helps engage your whole mind in the design process.***"

"Being a DFA"

That is:

Be the Machine!



What We are Given

When beginning this process, we are given the following information.

- An ***alphabet*** Σ — generally given by listing the symbols that are included in Σ.
- The ***language*** $L \subseteq \Sigma^\star$ of the DFA that is to be designed. See the material for Lecture #1, for a description of how L might be given to us.

Where We Want to End Up

At the end of our process, we should have given the following:

- A deterministic finite automaton

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

  whose alphabet is the alphabet, Σ, that we were given.
- A ***proof*** that $L = L(M)$ — or enough information so that it is clear that this proof could be written.

Where We Will Start

Remember that a deterministic finite automaton processes processes a string by ***considering one symbol in the string at a time***, in the order in which the symbols occur in the string.

- The *only* information that you (as the automaton) know about what has seen so far is represented by the ***current state***.
- A deterministic finite automaton has only ***finitely many states***, so only a *finite* amount of information can be remembered, even though the input string can be arbitrarily long.

Starting the Process

To start things off we will — somehow — try to answer the following question.

> What do you need to remember about the part of the string you have seen so far?

Note: You do *not* need to be sure about the answer in order to start! Answering this question is the step that requires the most imagination and creativity in this design process.

Ongoing Example

The process being developed will be used to design a deterministic finite automata for the following language over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$:

$$L = \{\omega \in \Sigma^\star \mid \text{either } \omega \text{ does not include an "}\mathtt{a}\text{"} \quad \text{or } \omega \text{ does not include a "}\mathtt{b}\text{"}\}.$$

Clarification: In the definition of L, "or" does not mean "exclusive or" — *both* conditions might be satisfied. Thus L includes all the strings that only include $\mathtt{c}$'s.

Application to Example — A First Attempt

Since

$$L = \{\omega \in \Sigma^\star \mid \text{either } \omega \text{ does not include an "a"} \\ \text{or } \omega \text{ does not include a "b"}\}.$$

we probably need to remember *whether both an* “a” *and a* “b” *have already been seen* — because the string that has been processed is in L if this is *not* the case.

Continuing the Process

To continue, use your answer to the question (about what the DFA must remember) to define *a collection*

$$S_0, S_1, \ldots, S_{n-1}$$

of subsets of $\Sigma^\star$, for some positive integer n — such that the information you are remembering, about a string $\omega \in \Sigma^\star$, is the same as remembering which, of these subsets, ω belongs to.

Application to Example — A First Attempt

Since we will try to remember whether both an “a” and a “b” have already been seen — and the string processed, so far, is *not* in L when this is true, we will consider a pair of subsets of $\Sigma^\star$, namely

$$S_{\text{yes}} = \{\omega \in \Sigma^\star \mid \text{either } \omega \text{ does not include an “a”} \\ \text{or } \omega \text{ does not include a “b”}\}$$

and

$$S_{\text{no}} = \{\omega \in \Sigma^\star \mid \omega \text{ includes both an “a” and a “b”}\}.$$

Now $n = 2$, $S_0 = S_{\text{yes}} = L$, and $S_1 = S_{\text{no}} = \{\omega \in \Sigma^\star \mid \omega \notin L\}$.

Continuing the Process: A First “Sanity Check”

Before doing much more you should confirm that the following properties hold.

1. Only a *finite* number of subsets of $\Sigma^\star$ have been identified — so that these really *can* be numbered

$$S_0, S_1, \ldots, S_{n-1}$$

for some positive integer n.

Explanation: Each of these subsets will eventually correspond to a different ***state*** in the automaton being designed — and a DFA can only have a *finite* number of states.

Application to the Example — A First Attempt

- This is condition is satisfied for this example, since only ***two*** subsets, $S_0 = S_{\text{no}}$ and $S_1 = S_{\text{yes}}$, have been identified.
- We will (initially) be trying to design a deterministic finite automaton with a set of states

$$Q = \{q_{\text{no}}, q_{\text{yes}}\}$$

  where the state q_{no} "corresponds to" the set S_{no} and where the state q_{yes} "corresponds to" the set S_{yes} in some way.
- This kind of "correspondence" will be described, more completely, later on in this process.

Continuing the Process: A Second "Sanity Check"

2. Every string $\omega \in \Sigma^\star$ belongs to ***exactly one*** of the sets

$$S_0, S_1, \ldots, S_{n-1}$$

   that have been identified.

Explanation: When our automaton sees (and processes) the string ω it should end up in ***exactly one*** state.

Application to the Example — A First Attempt

- This condition is also satisfied for this example, because $S_{\text{yes}} = L$ and $S_{\text{no}} = \{\omega \in \Sigma^\star \mid \omega \notin L\}$ — so that it is sufficient to examine the definitions, of S_{yes} and S_{no}, in order to see that every string in $\Sigma^\star$ must belong to exactly one of these subsets.

Discovering More: Identifying the Start State

If the second "sanity check," above, was passed, then the empty string λ belongs to ***exactly one*** of the subsets $S_0, S_1, \ldots, S_{n-1}$.

The ***start state*** should now be set to be q_i, for the unique integer i such that $0 \leq i \leq n-1$ and $\lambda \in S_i$.

Simplification: If necessary, let us renumber states (and subsets) as needed, so that $\lambda \in S_0$ and q_0 is the start state.

Application to the Example — A First Attempt

- In our example, $\lambda \in S_{\text{yes}}$ — so the start state of our DFA should be the state, q_{yes}, that corresponds to this subset of $\Sigma^{\star}$ — as given above.

How Subsets Correspond To States

The ***correspondence*** between subsets of $\Sigma^\star$ and states in Q can now be described more precisely:

Desired Property: For every string $\omega \in \Sigma^\star$ and for every integer i such that $0 \leq i \leq n-1$,

$$\omega \in S_i$$

if and only if

$$\delta^\star(q_0, \omega) = q_i$$

in the automaton $M = (Q, \Sigma, \delta, q_0, F)$ that is being designed.

Application to the Example — A First Attempt

- Since q_{yes} is the start state, for our example, this correspondence is as follows: For every string $\omega \in \Sigma^\star$,

$$\omega \in S_{\text{yes}} \text{ if and only if } \delta^\star(q_{\text{yes}}, \omega) = q_{\text{yes}}$$

and

$$\omega \in S_{\text{no}} \text{ if and only if } \delta^\star(q_{\text{yes}}, \omega) = q_{\text{no}}.$$

The Example, So Far

Desired Properties:

- $\delta^\star(q_{\text{yes}}, \omega) = q_{\text{yes}}$ for every string $\omega \in S_{\text{yes}}$, that is, for all ω such that either ω does not include an "a" or ω does not include a "b" (or both);
- $\delta^\star(q_{\text{yes}}, \omega) = q_{\text{no}}$ for every string $\omega \in S_{\text{no}}$, that is, for all ω such that ω includes both an "a" and a "b".

Continuing the Process: A Third "Sanity Check"

Consider a subset S_i (for $0 \leq i \leq n-1$) that has been identified, and the state, $q_i \in Q$, that it corresponds to.

- If q_i is an accepting state then it must be true that $\omega \in L$ for every string $\omega \in \Sigma^\star$ such that $\delta^\star(q_0, \omega) = q_i$. That is, it must be true that $\omega \in L$ whenever $\omega \in S_i$ — so that $S_i \subseteq L$.
- On the other hand, if q_i is *not* an accepting state then it must be true that $\omega \notin L$ for every string $\omega \in \Sigma^\star$ such that $\delta^\star(q_0, \omega) = q_i$. That is, it must be true that $\omega \notin L$ whenever $\omega \in S_i$ — so that $S_i \cap L = \emptyset$.

Continuing the Process: A Third "Sanity Check"

This is the reason for the following third "sanity check":

3. Either $S_i \subseteq L$ or $S_i \cap L = \emptyset$ for each integer i such that $0 \leq i \leq n-1$.

If this condition is satisfied, because $S_i \subseteq L$, then q_i should be an accepting state. On the other hand, q_i should *not* be an accepting state if the condition is satisfied with $S_i \cap L = \emptyset$.

Application to the Example — A First Attempt

- Recall that

$$L = S_{\text{yes}} = \{\omega \in \Sigma^\star \mid \text{either } \omega \text{ does not include an ``a''} \\ \text{or } \omega \text{ does not include a ``b''}\}$$

and that $S_{\text{no}} = \{\omega \in \Sigma^\star \mid \omega \text{ includes both an ``a'' and a ``b''}\}$.

Since $S_{\text{yes}} \subseteq L$ the corresponding state, q_{yes}, *is* an accepting state. Since $S_{\text{no}} \cap L = \emptyset$ the corresponding state, q_{no}, is *not* an accepting state.

The Example, So Far

Desired Properties:

- $\delta^\star(q_{\text{yes}}, \omega) = q_{\text{yes}}$ for every string $\omega \in S_{\text{yes}}$, that is, for all ω such that either ω does not include an "a" or ω does not include a "b" (or both);
- $\delta^\star(q_{\text{yes}}, \omega) = q_{\text{no}}$ for every string $\omega \in S_{\text{no}}$, that is, for all ω such that ω includes both an "a" and a "b".

Continuing the Process: A Fourth "Sanity Check"

Consider a subset S_i (for $0 \leq i \leq n-1$) and a symbol $\sigma \in \Sigma$.

- Since transitions should be well-defined, there must also exist an integer j such that $0 \leq j \leq n-1$ and

$$\delta(q_i, \sigma) = q_j$$

— where $q_i \in Q$ corresponds to S_i and $q_j \in Q$ corresponds to S_j.

- Thus, if $\omega \in \Sigma^\star$ such that $\omega \in S_i$, then

$$\begin{aligned}
\delta^\star(q_0, \omega \cdot \sigma) &= \delta(\delta^\star(q_0, \omega), \sigma) \quad \text{(as shown in Lecture \#2)} \\
&= \delta(q_i, \sigma) \quad \text{(since } \omega \in S_i\text{, so } \delta^\star(q_0, \omega) = q_i\text{)} \\
&= q_j
\end{aligned}$$

— implying that $\omega \cdot \sigma \in S_j$.

Continuing the Process: A Fourth "Sanity Check"

- Since ω was arbitrarily chosen from $\Sigma^\star$ such that $\omega \in S_i$,

$$\omega \cdot \sigma \in S_j$$

for all strings $\omega \in S_i$ (where $j = \delta(q_i, \sigma)$).

- The same property can be written as follows (for the same integers i and j, and the same symbol σ):

$$\{\omega \cdot \sigma \mid \omega \in S_i\} \subseteq S_j.$$

Continuing the Process: A Fourth "Sanity Check"

This is the reason for the fourth "sanity check":

4. For every integer i such that $0 \leq j \leq n - 1$ and for every symbol $\sigma \in \Sigma$, there exists an integer j such that $0 \leq j \leq n - 1$ and such that

$$\{\omega \cdot \sigma \mid \omega \in S_i\} \subseteq S_j.$$

   Then $\delta(q_i, \sigma) = q_j$, when q_i is the state corresponding to S_i and when q_j is the state corresponding to S_j.

Application to the Example: A First Attempt

Let us begin by considering the state q_{no}, which corresponds to the set

$$S_{\text{no}} = \{\omega \in \Sigma^\star \mid \omega \text{ includes both an “a” and a “b”}\}.$$

- If $\omega \in S_{\text{no}}$, so that ω includes both an “a” and a “b”, then so do $\omega \cdot \mathtt{a}$, $\omega \cdot \mathtt{b}$, and $\omega \cdot \mathtt{c}$. Thus

  $$\{\omega \cdot \sigma \mid \omega \in S_{\text{no}}\} \subseteq S_{\text{no}}$$

  for $\sigma = \mathtt{a}$, for $\sigma = \mathtt{b}$, and for $\sigma = \mathtt{c}$, so that

  $$\delta(q_{\text{no}}, \mathtt{a}) = \delta(q_{\text{no}}, \mathtt{b}) = \delta(q_{\text{no}}, \mathtt{c}) = q_{\text{no}}.$$

The Example, So Far

Adding the transitions that have been discovered, we now have the following.

Application to the Example — A First Attempt

We must also discover transitions out of the state q_{yes}, which corresponds to the set

$$S_{\text{yes}} = \{\omega \in \Sigma^\star \mid \text{either } \omega \text{ does not include an "a"} \\ \text{or } \omega \text{ does not include an "b"}\}$$

Application to the Example — A First Attempt

- First, Some Good News: Suppose that $\omega \in S_{\text{yes}}$, so that either ω does not include an "a" or ω does not include an "b". Consider the string $\omega \cdot \text{c}$.
  - If ω does not include an "a" then $\omega \cdot \text{c}$ does not include an "a" either, so that $\omega \cdot \text{c} \in S_{\text{yes}}$.
  - If ω does not include a "b" then $\omega \cdot \text{c}$ does not include a "b" either, so that $\omega \cdot \text{c} \in S_{\text{yes}}$, once again.

Since $\omega \cdot \text{c} \in S_{\text{yes}}$ in every possible case, it follows that

$$\{\omega \cdot \text{c} \mid \omega \in S_{\text{yes}}\} \subseteq S_{\text{yes}},$$

so that

$$\delta(q_{\text{yes}}, \text{c}) = q_{\text{yes}}.$$

The Example, So Far

Adding the transition that has been discovered, we now have the following.

Application to the Example — A First Attempt

There are two more transitions to be discovered out of the state q_{yes}, which corresponds to the set S_{yes}.

- Suppose that $\omega \in S_{\text{yes}}$, so that either ω does not include an “a” or ω does not include an “b”. Consider the string $\omega \cdot \mathtt{a}$.

  If ω does not include an “b” then $\omega \cdot \mathtt{a}$ does not include an “b”, either — so that $\omega \cdot \mathtt{a} \in S_{\text{yes}}$.

  Unfortunately, if ω does not include an “a” then $\omega \cdot \mathtt{a}$ *does* include one!

  - If ω did not include a “b” either, then $\omega \cdot \mathtt{a} \in S_{\text{yes}}$, because $\omega \cdot \mathtt{a}$ (still) does not include a “b”.
  - However, if ω *does* include a “b” then $\omega \cdot \mathtt{a}$ includes both an “a” and a “b” — so that $\omega \cdot \mathtt{a} \in S_{\text{no}}$, instead.

Application to the Example — A First Attempt

Something similar happens when one considers a string $\omega \cdot \mathtt{b}$, for $\omega \in S_{\text{yes}}$:

- If ω does not include an "a" then $\omega \cdot \mathtt{b} \in S_{\text{yes}}$.
- On the other hand, if ω does not include a "b" then
  - $\omega \cdot \mathtt{b} \in S_{\text{yes}}$ if ω does not include an "a", either, but
  - $\omega \cdot \mathtt{b} \in S_{\text{no}}$, instead, if ω *does* include an "a".

Question: What has happened here, and what does it mean?

Application to the Example – A First Attempt

Answer:

The fourth “sanity check” has failed.

Application to the Example

- In particular, we have confirmed that it is ***not*** possible to identify well-defined transitions out of the state q_{yes} for the symbols "a" or "b".
- ***Conclusion (For Now):*** The automaton must remember ***different*** information — or, possibly, ***additional*** information — in order to recognize the language L.

To Be Continued...

- We will see, next time, that ***all that time and effort was worthwhile*** and that there is a way to make use of what we have done, so far, to solve this problem.