˜ CS 373: Theory of Computation
˜ Sariel Har-Peled and Madhusudan Parthasarathy

Discussion 2: Examples of DFAs

27 January 2009

Purpose: This discussion demonstrates a few constructions of DFAs. However, its main purpose is to show how to move from a diagram describing a DFA into a formal description, in particular of the transition function.
This material here (probably) can not be covered in one discussion section.

Questions on homework 1?

Any questions? Complaints, etc?

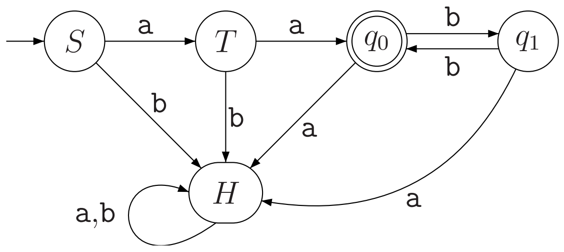
1 Languages that depend on k

1.1 $\texttt{aab}^{2i}$

Consider the following language:

$$L_2 = \left\{ \texttt{aab}^n \;\middle|\; n \text{ is a multiple of } 2 \right\}.$$

Its finite automata is



Advice To TA:: Do not erase this diagram from the board, you would need to modify it shortly, for the next example. **end**

This automata formally is the tuple $(Q, \Sigma, \delta, S, F)$.

1. $Q = \{S, T, H, q_0, q_1\}$ - states.
2. $\Sigma = \{\texttt{a}, \texttt{b}\}$ - alphabet.
3. $\delta : Q \times \Sigma \to Q$, described in the table.

	a	b
S	T	H
T	q_0	H
H	H	H
q_0	H	q_1
q_1	H	q_0

4. S is the start state.

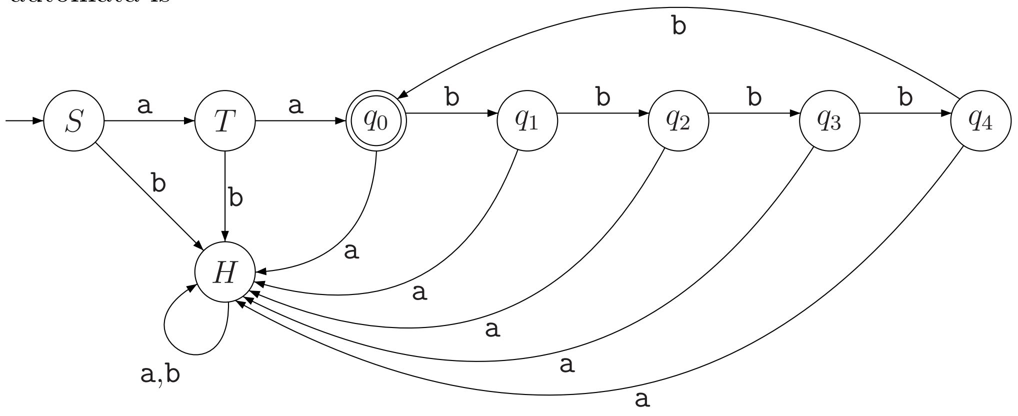
5. $F = \{q_0\}$ is the set of accepting states.

1.2 $\texttt{aab}^{5i}$

Consider the following language:

$$L_5 = \left\{ \texttt{aab}^n \;\middle|\; n \text{ is a multiple of } 5 \right\}.$$

Its finite automata is



Advice To TA:: Do not erase this diagram from the board, you would need to modify it shortly, for the next example. **end**

This automata formally is the tuple $(Q, \Sigma, \delta, S, F)$.

1. $Q = \{S, T, H, q_0, q_1, q_2, q_3, q_4\}$ - states.
2. $\Sigma = \{\texttt{a}, \texttt{b}\}$ - alphabet.
3. $\delta : Q \times \Sigma \to Q$ - see table.
4. S is the start state.
5. $F = \{q_0\}$ is the set of accepting states.

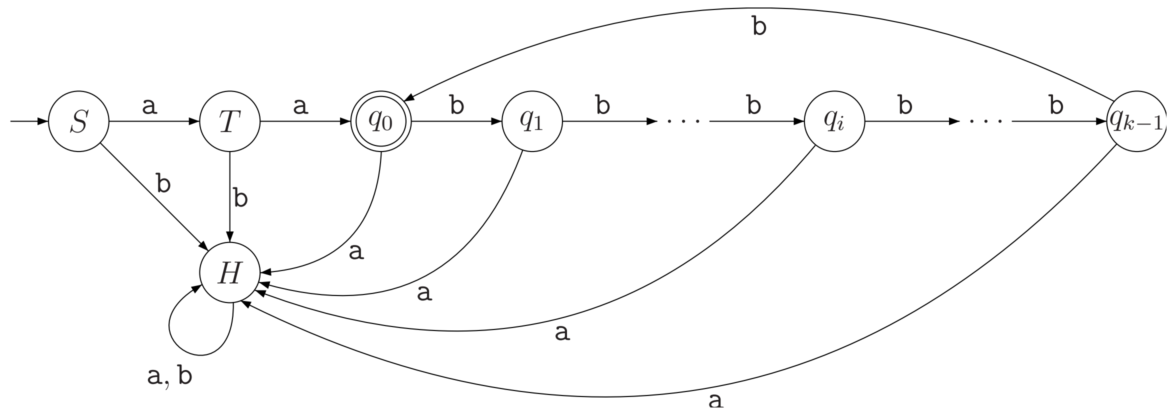
δ	a	b
S	T	H
T	q_0	H
H	H	H
q_0	H	q_1
q_1	H	q_2
q_2	H	q_3
q_3	H	q_4
q_4	H	q_0

1.3 $\texttt{aab}^{ki}$

Let k be a fixed constant, and consider the following language:

$$L_k = \left\{ \texttt{aab}^n \;\middle|\; n \text{ is a multiple of } k \right\}.$$

Its finite automata is



This automata formally is the tuple $(Q, \Sigma, \delta_k, S, F)$.

1. States: $Q = \{S, T, H, q_0, q_1, \ldots, q_{k-1}\}$.
2. $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ - alphabet.
3. $\delta_k : Q \times \Sigma \to Q$ - see table.
4. S is the start state.
5. $F = \{q_0\}$ is the set of accepting states.

	a	b
S	T	H
T	q_0	H
H	H	H
q_0	H	q_1
q_1	H	q_2
$\vdots$	$\vdots$	$\vdots$
q_i	H	q_{i+1}
$\vdots$	$\vdots$	$\vdots$
q_{k-1}	H	q_0

1.3.1 Explicit formula for δ

Advice To TA:: Skip the first two forms in the discussion. Show only the one in Eq. (1).
end

Another way of writing the transition function δ_k, for the above example, is the following:

$$\begin{aligned}
\delta_k(S, \mathtt{a}) &= T,\\
\delta_k(S, \mathtt{b}) &= H,\\
\delta_k(T, \mathtt{a}) &= q_0,\\
\delta_k(T, \mathtt{b}) &= H,\\
\delta_k(H, \mathtt{a}) &= H,\\
\delta_k(H, \mathtt{b}) &= H,\\
\delta_k(q_i, \mathtt{a}) &= H, \quad \forall i\\
\delta_k(q_i, \mathtt{b}) &= q_{i+1} \quad \text{for } i < k-1,\\
\delta_k(q_{k-1}, \mathtt{b}) &= q_0.
\end{aligned}$$

Another way to write the same information

$$\delta_k(s,x) = \begin{cases} T & s = S, x = \texttt{a} \\ H & s = S \text{ or } T, x = \texttt{b} \\ q_0 & s = T, x = \texttt{a} \\ H & s = H, x = \texttt{a} \text{ or } \texttt{b} \\ H & s = q_i, x = \texttt{a}, \qquad i = 0, \ldots, k-1 \\ q_{i+1} & s = q_i, x = \texttt{b}, \qquad i < k-1, \\ q_0 & s = q_{k-1}, s = \texttt{b}. \end{cases}$$

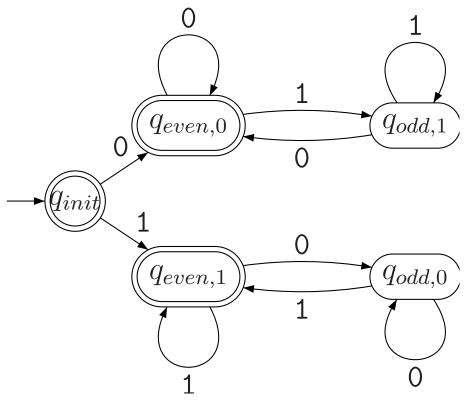
This can be made slightly more compact using the mod notation:

$$\delta_k(s,x) = \begin{cases} T & s = S, x = \texttt{a} \\ H & s = S \text{ or } T, x = \texttt{b} \\ q_0 & s = T, x = \texttt{a} \\ H & s = H, x = a \text{ or } \texttt{b} \\ H & s = q_i, x = \texttt{a}, \qquad \forall i \\ q_{(i+1) \bmod k} & s = q_i, x = \texttt{b}, \qquad \forall i. \end{cases} \tag{1}$$

Note, that using good state names would help you to describe the automata compactly (thus q_0 here is not that the starting state). Generally speaking, the shorter your description is, the least work needed to be done, and the chance you make a silly mistake is lower.

2 Number of changes from 0 to 1 is even

Let us build a DFA that recognizes all binary strings, such that the number of times the string changes from consecutive zeroes to ones (and vice versa) is even. Thus 0000 is in our language (number of changes is 0, but 0000111111 is not (number of changes is one). We need to remember what was the last character in the input, and whether the number of changes so far was even or odd. As such, we need 4 states. But wait! What about the empty string ϵ. Clearly, it is in the language (i.e., the number of changes between 0 and 1 is zero, which is even). As such, we need a special state to handle it.



1. States: $Q = \{q_{init}, q_{even,0}, q_{even,1}, q_{odd,0}, q_{odd,1}\}$.
2. $\Sigma = \{0, 1\}$ - alphabet.
3. $\delta : Q \times \Sigma \to Q$, see table on the right.
4. q_{init} is the start state.
5. Set of accepting states: $F = \{q_{even,0}, q_{even,1}, q_{init}\}$.

	0	1
q_{init}	$q_{even,0}$	$q_{even,1}$
$q_{even,0}$	$q_{even,0}$	$q_{odd,1}$
$q_{even,1}$	$q_{odd,0}$	$q_{even,1}$
$q_{odd,0}$	$q_{odd,0}$	$q_{even,1}$
$q_{odd,1}$	$q_{even,0}$	$q_{odd,1}$

The resulting finite automata is the tuple $(Q, \Sigma, \delta, q_{init}, F)$.

3 State explosion

`Advice To TA::` Show first the longer way to solve this problem. Use your board cleverly, so that you can move from the tedious form into the shorter form, by just erasing things. `end`

Because automatas do not have an explicit way of storing information, we need to encode the information to solve the problem explicitly in the states of the DFA. That can get very tedious, as the following example shows.

Let L be the language of all binary strings such that the third character from the end is zero.

To design an automata that accepts this language, we clearly need to be able to remember the last three characters of the input. To this end, let us introduce the states $q_{000}, q_{001}, q_{010}, q_{011}, q_{100}, q_{101}, q_{110}, q_{111}$. Here the automata would be in state q_{011} if the last three characters are 0, 1 and 1. Naturally, we need to also be able to handle the first one or two characters arriving to the input, which forces us to introduce special states for this case. Here is the resulting automata in formal description. Here, the automata is the tuple $(Q, \Sigma, \delta, e, F)$.

1. States:
   $Q = \{e, q_0, q_1, q_{00}, q_{01}, q_{10}, q_{11}, \quad q_{000}, q_{001}, q_{010}, q_{011},$
   $q_{100}, q_{101}, q_{110}, q_{111}\}$
2. $\Sigma = \{0, 1\}$ - alphabet.
3. $\delta : Q \times \Sigma \rightarrow Q$ - see table.
4. e is the start state.
5. $F = \{q_{000}, q_{001}, q_{010}, q_{011}\}$ is the set of accepting states.

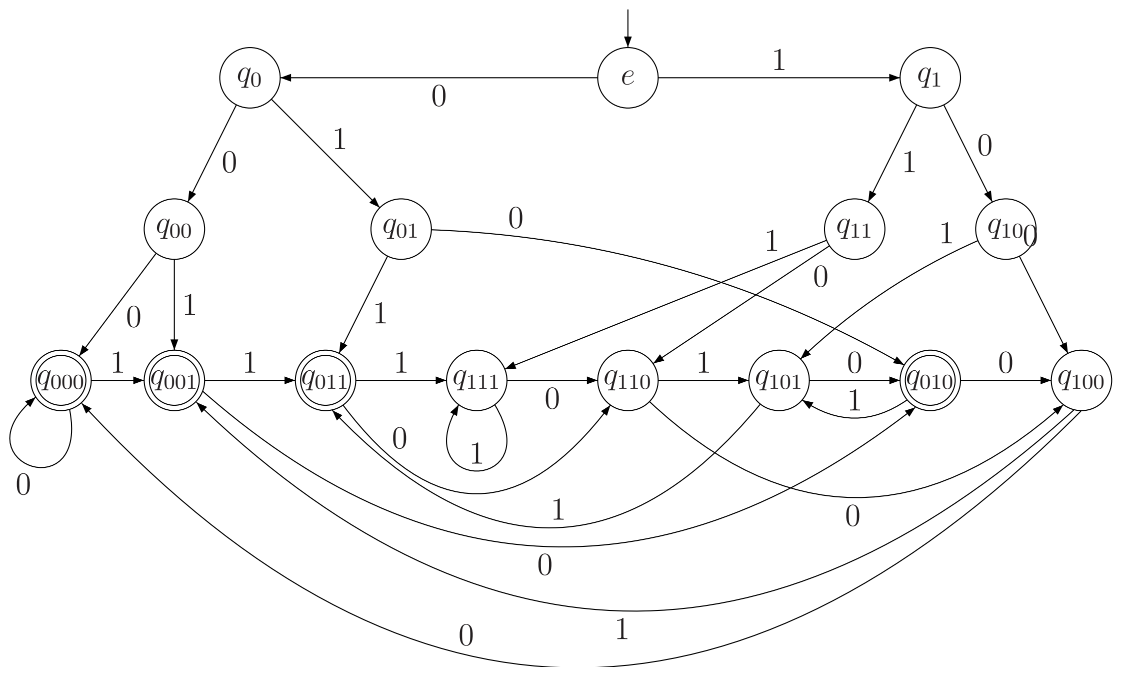
	0	1
e	q_0	q_1
q_0	q_{00}	q_{01}
q_1	q_{10}	q_{11}
q_{00}	q_{000}	q_{001}
q_{01}	q_{010}	q_{011}
q_{000}	q_{000}	q_{001}
q_{001}	q_{010}	q_{011}
q_{010}	q_{100}	q_{101}
q_{011}	q_{110}	q_{111}
q_{100}	q_{000}	q_{001}
q_{101}	q_{010}	q_{011}
q_{110}	q_{100}	q_{101}
q_{111}	q_{110}	q_{111}

`Advice To TA::` Please show the explicit long way of writing the transition table (shown above), and only then show the following more compact way. Its beneficial to see how using more formal representation, can save you a lot of time and space. Say it explicitly in the discussion. `end`

A more sane way to write the transition table for δ is

	0	1
e	q_0	q_1
q_x	q_{x0}	q_{x1}
q_{xy}	q_{xy0}	q_{xy1}
q_{xyz}	q_{yz0}	q_{yz1}

Which is clearly the most compact way to describe this transition function. And here is a drawing of this automata:



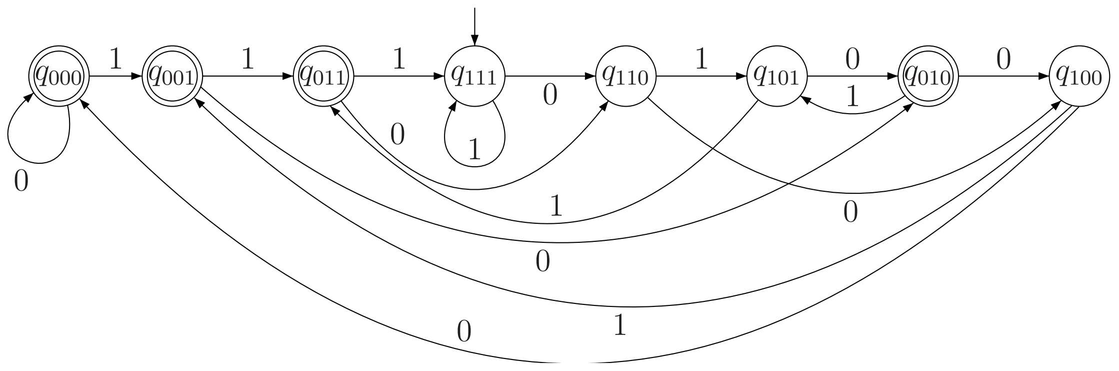
Advice To TA:: Trust me. You do not want to erase this diagram before doing the next example.. **end**

3.1 Being smarter

Since we care only if the third character from the end is zero, we could pretend that when the input starts, the automata already saw, three ones on the input. Thus, setting the initial state to q_{111}. Now, we can get rid of the special states we had before.

1. States:
   $Q = \{q_{000}, q_{001}, q_{010}, q_{011},$
   $q_{100}, q_{101}, q_{110}, q_{111}\}$
2. $\Sigma = \{0, 1\}$ - alphabet.
3. $\delta : Q \times \Sigma \rightarrow Q$ - see table.
4. q_{111} is the start state.
5. $F = \{q_{000}, q_{001}, q_{010}, q_{011}\}$ is the set of accepting states.

	0	1
q_{xyz}	q_{yz0}	q_{yz1}



This brings to the forefront several issues: (i) the most natural way to design an automata does not necessarily leads to the simplest automata, (ii) a bit of thinking ahead of time will save you much pain, and (iii) how do you know that what you came up with is the simplest (i.e., fewest number of states) automata accepting a language?

The third question is interesting, and we will come back to it later in the course.

4 None of the last k characters from the end is 0

Let L'_k be the language of all binary strings such that none of the least k characters is 0. We can of course adapt the previous automata to this language, by changing the accepting states and the start state. However, we can solve it more efficiently, by remembering what is the maximum length of the suffix of the input seen so far, such that its all one.

In particular, let q_i be the state such that the suffix of the input is a zero followed by i ones. Clearly, q_k is the accept state, and q_0 is the starting state. The transition function is also simple. If the automata sees a 0, it goes back to q_0. If it at q_i and it accepts a 1 it goes to q_{i+1}, if $i < k$. . We get the following automata.

1. States: $Q = \left\{ q_i \,\middle|\, i = 0, \ldots, k \right\}$.
2. $\Sigma = \{0, 1\}$ - alphabet.
3. $\delta_k : Q \times \Sigma \to Q$, where
$$\delta_k(q_i, x) = \begin{cases} q_0 & x = 0 \\ q_{\min(i+1,k)} & x = 1. \end{cases}$$
4. q_0 is the start state.
5. Set of accepting states: $F = \{q_k\}$.

So, a little change in the definition of the language can make a dramatic difference in the number states needed.