CP414: Foundations of Computing

by Scott King

Winter Term 2017

Office: N2087

- For reference, a *problem* is a general problem (ex. travelling salesman) and a *problem instance*, is a specific case of a problem

- There are times where we can solve complimentary problems given the solution to another problem (ex. determining if a graph is 2-colourable can be solved by checking if the graph if bipartite; which can be done with BFS)

We have the symbol Σ that is considered the finite alphabet. Just to be clear on that;

$$\begin{aligned}\Sigma_1 &= \{a, b, c, ..., x, y, z\} \\ \Sigma_2 &= \{0, 1\}\end{aligned}$$

We can then denote $\Sigma^* = \{\text{set of all finite strings; } s_1 s_2 ... s_n | s_i \in \Sigma\}$. We can then denote something like:

$$\begin{aligned}\Sigma_1^* &= \{\epsilon, a, b, ..., z, aa, ..., zz, ...\} \\ \Sigma_2^* &= \{\epsilon, 0, 1, 00, 11, ...\}\end{aligned}$$

Note: Use ϵ to denote empty string.

Then ... we can define a language; where we would denote $L \subset \Sigma^*$.

In this course, we're dealing with decision problems. **Not** solving the answers to problems, but essentially determining *yes* or *no*.

Let's look a problem to solve graph connectivity. Is a given graph connected or not?

Let's define a set $\Sigma = \{0, 1, ..., 9, \#\}$ as our language. We need this to encode the graph.

The graph is define as such $G = \{V, E \subseteq V \times V\}$.

Note: Edges shall be denoted as $n\#m...$

We can then define a graph as 5#1#6#3#2#4##. The double pound indicates there are no more verticies. The above graph shall be encoded as 1#2#1#3#2#4#3#4#3#6#4#5##. Now with this, we are assuming that 1#2 is also 2#1.

All correct encodings of this graph, G; $L_G \subset E^*$. Then we can have the set $L_{\text{con}} = \{\text{all encodings of connected graph}\}$. And thus $L_{\text{con}} \subset L_G \subset \Sigma^*$.

In this case, we have reached a point where can take any *problem instance* and talk strictly within the language of encoding.

Upon defining *any* language, we need a tool that tells us whether a given string, or item, is apart of ourly newly created language.

1 Deterministic *Finite* Automata

We can define $A = \{\Sigma, Q, \delta, q_0, F\}$ where Σ is our alphabet and Q is our set of possible states. and δ is our transition function, q_0 is our start state and F is our accepted states. Thus:

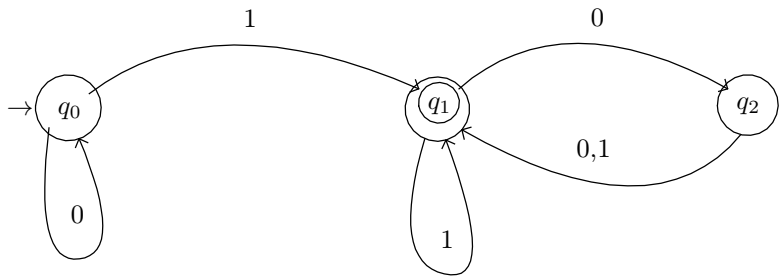
$$\begin{aligned} \delta &: \Sigma \times Q \to Q \\ q_0 &\in Q \\ F &\in Q \end{aligned}$$

and also $\Sigma \cap Q = \emptyset$.

Let's define $A_1 = \{\{0,1\}, \{q_0, q_1, q_2\}, \delta, q_0, \{q_1\}\}$. We can build our state matrix:

	0	1
$\to q_0$	q_0	q_1
$^*q_1$	q_2	q_1
q_2	q_1	q_1

We have to end on an acceptable state (q_1). We can create a directed graph given our above matrix.



$\to$ is the starting state and $\circ$ is the accepted state.

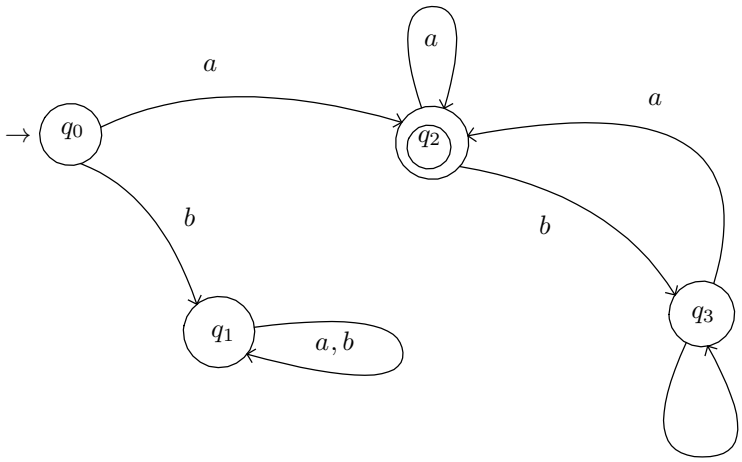
We can map $01101 \to q_0 q_0 q_1 q_1 q_2 q_1$ thus is an accepted string.

We can map $000 \to q_0 q_0 q_0$ and thus not accepted.

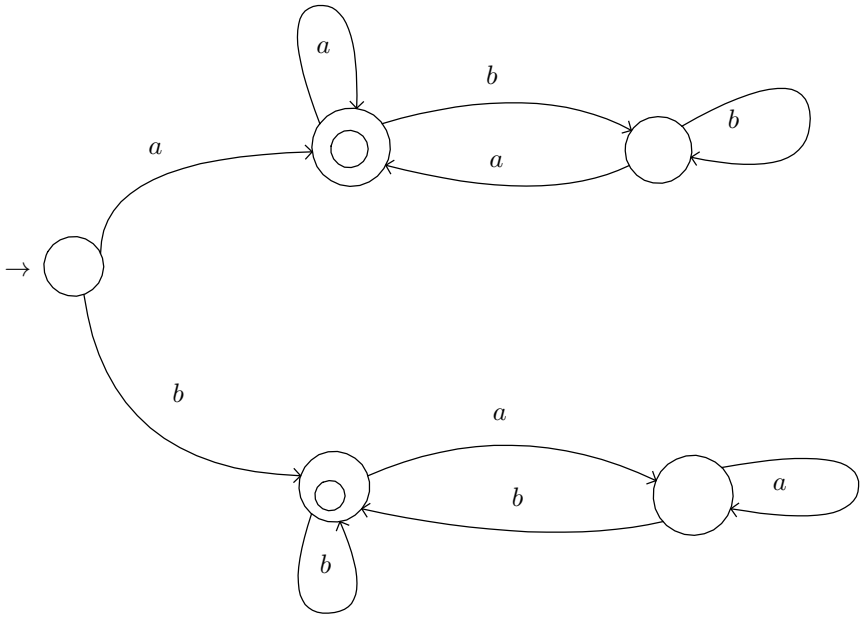
Let's define a language, such that $L_1(A) = \{\text{set of strings over } \Sigma, \text{accepted by } A\}$.

Note: In a DFA, the number of steps taken is equal to the length of the input string. DFA are mainly used for text processing. And finite automata *only*, really, remembers the current character. There is no second traversal.

We can define another language where $L_2 = \{\text{set of strings over } \{a, b\} \text{ that start and end with } a\}$.



We can also define an $L_3 = \{\text{starts and ends with } b\}$ and $L_4 = \{\text{starts and ends with some character}\}$.



And thus, $L_4 = L_2 \cup L_3$.

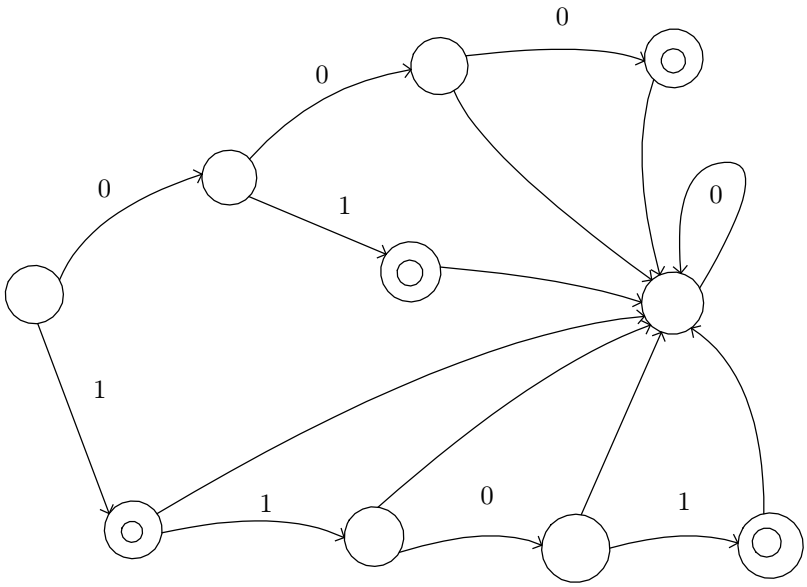
And such we can also define $\overline{L_1} = Q / F$.

Definition 1. *A language described by DFA is called* ***zepulon****.*

*If L is zepulon $\Leftrightarrow$ $\bar{L}$ is **nepulon**.*

*Any **finite** language is zepulon.*

Example 2. $L_5 = \{01, 000, 1101, 1\}$



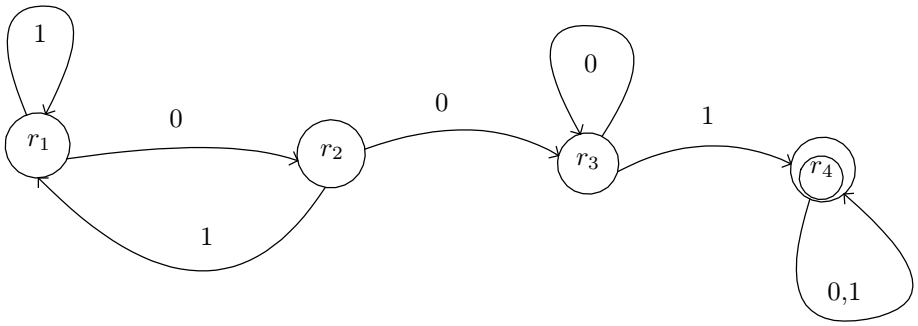
Example 3. Given the previous language, $L(A_5) = \{w \in \Sigma | \text{odd number of 1s}\}$, we say:

	0	1
$\rightarrow q_1$	q_1	q_2
$*q_2$	q_2	q_1

$L(A_6) = \{w \in \Sigma | w \text{ contains a pattern } 001\}$. For A_6, we have:

	0	1
$\rightarrow r_1$	r_2	r_1
r_2	r_3	r_1
r_3	r_3	r_4
$*r_4$	r_4	r_4

Remember that ϵ (empty string) will be denied. We have:



Now, we if wanted to do something like $L(A_5) \cup L(A_6)$, we could define the state matrix:

	0	1
$\rightarrow(r_1, q_1)$	(r_2, q_1)	(r_1, q_2)
(r_1, q_2)	(r_2, q_2)	...

for all the state pairs (8).

Example 4. We have $A_1 = \{\Sigma, Q_1, \delta_1, q_1, F_1\}$ and $A_2 = \{\Sigma, Q_2, \delta_2, r_1, F_2\}$. We can build another automata A: $L(A) = L(A_1) \cap L(A_2)$.

Luckily, we're using the same alphabet, Σ. $\Sigma, Q = Q_1 \times Q_2 \sim (x, y) : x \in Q_1, y \in Q_2$.

Our transition states become: $\delta(c, (x, y)) = (\delta_1(c, x), \delta_2(c, y))$.

Our start state becomes: (q_1, r_1). The finished state then becomes: $(u, v) \in F \Leftrightarrow u \in F_1 \wedge v \in F_2$.

Note: If we have languages, $L_1, L_2 \in R$ then these languages have:

1. $L_1 \cup L_2 \in R$
2. $L_1 \cap L_2 \in R$
3. $\overline{L_1} \in R$
4. $L_1 \circ L_2$

$$L_1 \circ L_2 \;=\; \{w \in \Sigma^* | w = w_1 w_2 ... w_n = w_1 ... w_k w_{k+1} ... w_n : w_1 ... w_k \in L_1, w_{k+1} ... w_n \in L_2\}$$

Note: *Sometimes* order can matter, $L_1 \circ L_2 \neq L_2 \circ L_1$.

5. $L^* \in R$: Kleene star (sort of a concatentation of a string from one language and a string from another where the properties of the language are still preserved)

$$L^* \;=\; \{L \circ L \circ L \circ ... \circ L | c \geq 0\}$$

If we look back at the examples of $L(A_5)$ and $L(A_6)$, we have:

$$\begin{aligned} 001 &\in L(A_6) \cap L(A_5) \\ 001 &\notin L(A_6) \circ L(A_5) \\ 111001 &\in L(A_5) \circ L(A_6) \end{aligned}$$

For **5**, let's look at:

$$10|11001 \in L(A_5)^* \notin \Sigma^*$$

This is because there are still an odd number of ones in each subset.

2 *Nondeterministic* Finite Automata

The idea is to go from DFA to NFA and show the difference in computing power.

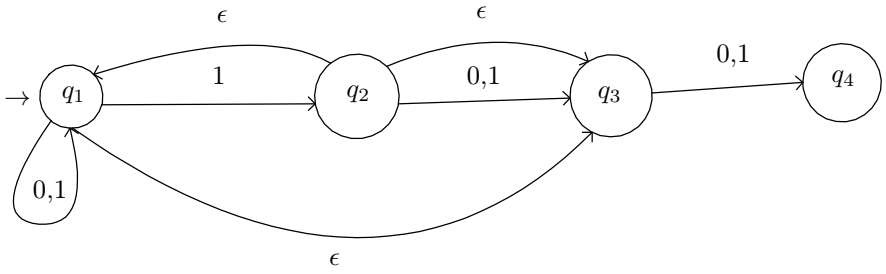
Let's look at a new language:

$$L_{11} = \{w \in (0,1)^* | \text{the third character from right is } 1\}$$

Ex. 0100$\underline{1}$00.

We can then define a NFA; NFA $= \{\Sigma, Q, \delta, q_0, F\}$ and really the only difference to distinguish an NFA is the transition function, δ.

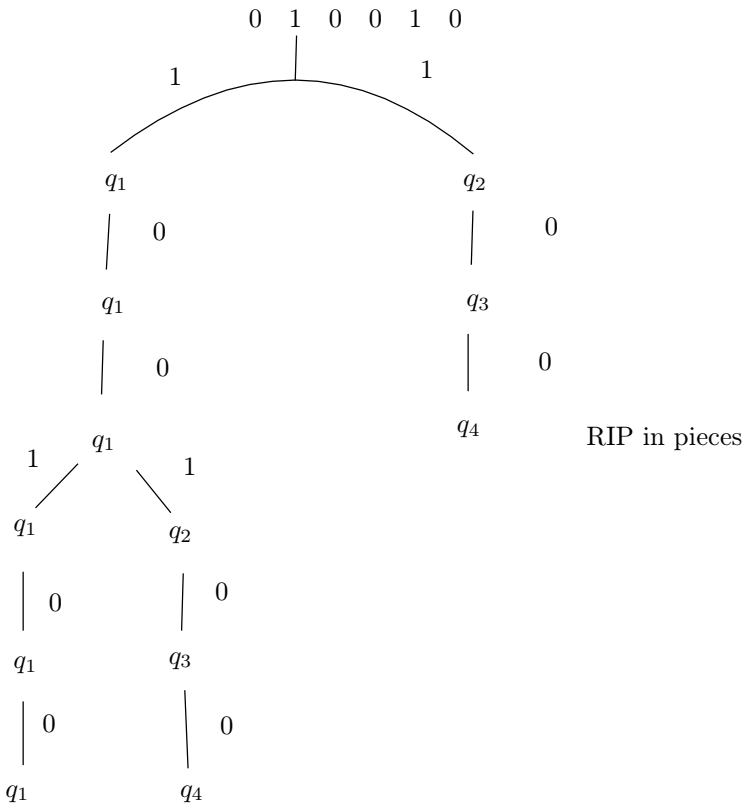
Example 5. We have $L_{11} = \{w \in (0,1)^* | \text{3rd number from right is } 1\}$. We can have the automata:



In the above automata, we can build the path ϵ (empty string) where we can either go back to q_1.

And since we are dealing with Kleene star (*) we need the start state to be an accepting state because Kleene star accepts empty string (ϵ). Final accepting states link back to second state, not starting states.

Example 6. We can look at 0100100.



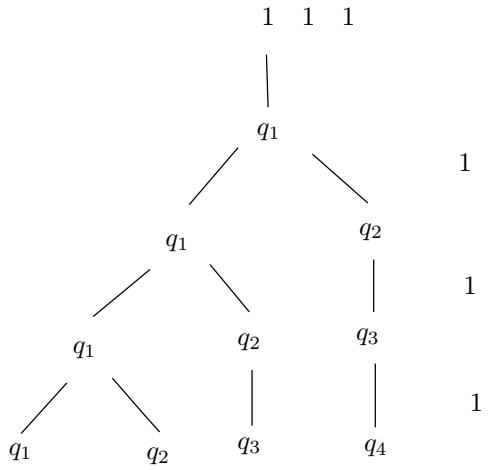
RIP in pieces

The transition function will now look like:

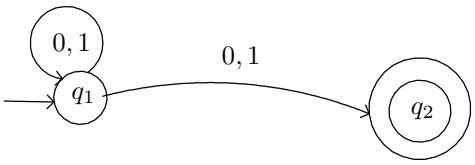
$$\delta \ : \ Q \times \Sigma_\epsilon \to P(Q)$$

(the new alphabet is extended to include ϵ). Rather, we map our input to a subset of states.

Example 7. We take look at:



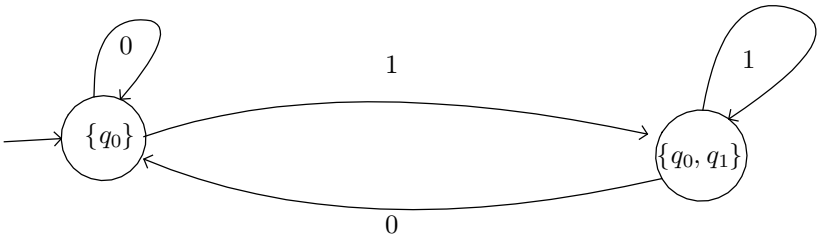
Example 8. We have the automata:



Q	0	1
$\rightarrow q_0$	$\{q_0\}$	$\{q_0, q_1\}$
$^*q_1$	$\emptyset$	$\emptyset$

From there we can derive:

	0	1
$\emptyset$	-	-
$\rightarrow\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$^*\{q_1\}$	$\emptyset$	$\emptyset$
$^*\{q_0, q_1\}$	$\{q_0\}$	$\{q_1\}$



We can look at the differences between automata.

2.1 DFA

We can define a generic DFA to be $\{\Sigma, Q, \delta\colon Q \times \Sigma \to Q, q_0, F\}$. Given $w = w_1 w_2 ... w_n\colon w_i \in \Sigma$.

$\exists$ a sequence of states $r_0, r_1, ..., r_n$ where $r_i \in Q$.

1. $r_0 = q_0$
2. $r_i = \delta(r_{i-1}, w_i), i = 1, ..., n$
3. $r_n \in F$

2.2 NFA

We can define a generic NFA to be $\{\Sigma, Q, \delta\colon Q \times \Sigma_\epsilon \to \rho(Q), q_0, F\}$. Given $w = w_1 w_2 ... w_n$.

$\exists\ y = y_1 y_2 ... y_m$ such that $m \geq n, y_i = \{w_i, \epsilon\}$ and the sequences of states: $r_0, r_1, ..., r_m$ where $r_i \in Q$.

1. $r_0 = q_0$
2. $r_i \in \delta(r_{i-1}, y_i), i = 1, 2, ..., m$
3. $r_m \in F$

But DFA $\cong$ NFA.

We want to create a DFA equivalent to the given NFA $= \{\Sigma, Q, \delta, q_0, F\} \sim$ DFA $= \{\Sigma, Q', \delta', q_0', F'\}$.

First thing is we have to create the DFA using the same language as the NFA. The original transition function of the NFA is:

$$\delta \ : \ Q \times \Sigma_\epsilon \to P(Q)$$

The DFA has $Q' = P(Q)$ and $|Q'| = 2^{|Q|}$. Also, $q' = \{q_0\}$, but $q' = E(q_0)$ (refer to definition below). The transition function of the DFA acts as such:

$$\delta' \ : \ Q' \times \Sigma \to Q$$

for which we'd have $\delta(R, x) = \bigcup_{r \in R} \delta(z, x)$. This essenitially means that the state for the DFA will be a union of states from the NFA.

{add 1.1}

Given a state $r \in Q$, we want to convert the syntax $\delta(R, x)$ so that ϵ is included. We can define the notation:

$$E(r) \ = \ \{\text{set of states reachable from } r \text{ along } \Sigma - \text{production}\}$$

Note: $r \in E(r)$. The formal change looks like:

$$\delta(R, x) \ = \ \bigcup_{r \in R} E(\delta(z, x))$$