# 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 $\Sigma$ 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 $\epsilon$ 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 $\Sigma$ is our alphabet and $Q$ is our set of possible states. and $\delta$ is our transition function, $q_0$ is our start state and $F$ is our accepted states. Thus:

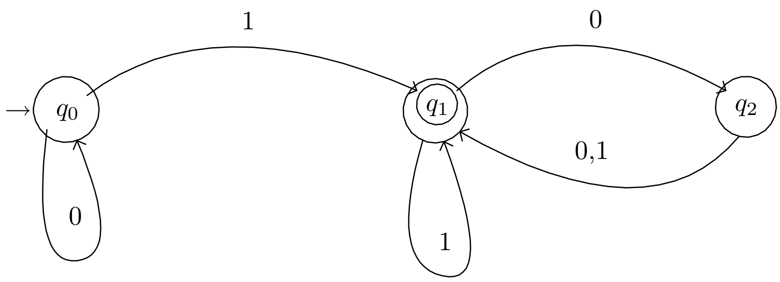
$$\begin{aligned} \delta &: \Sigma \times Q \rightarrow Q \\ q_0 &\in Q \\ F &\in Q \end{aligned}$$

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 |
|---|---|---|
| $\rightarrow q_0$ | $q_0$ | $q_1$ |
| $^*q_1$ | $q_2$ | $q_1$ |
| $q_2$ | $q_1$ | $q_1$ |

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




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

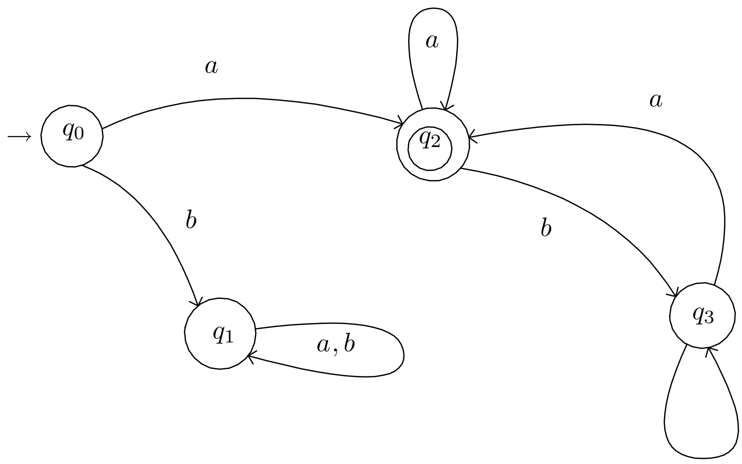
We can map $01101 \rightarrow q_0 q_0 q_1 q_1 q_2 q_1$ thus is an accepted string.

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

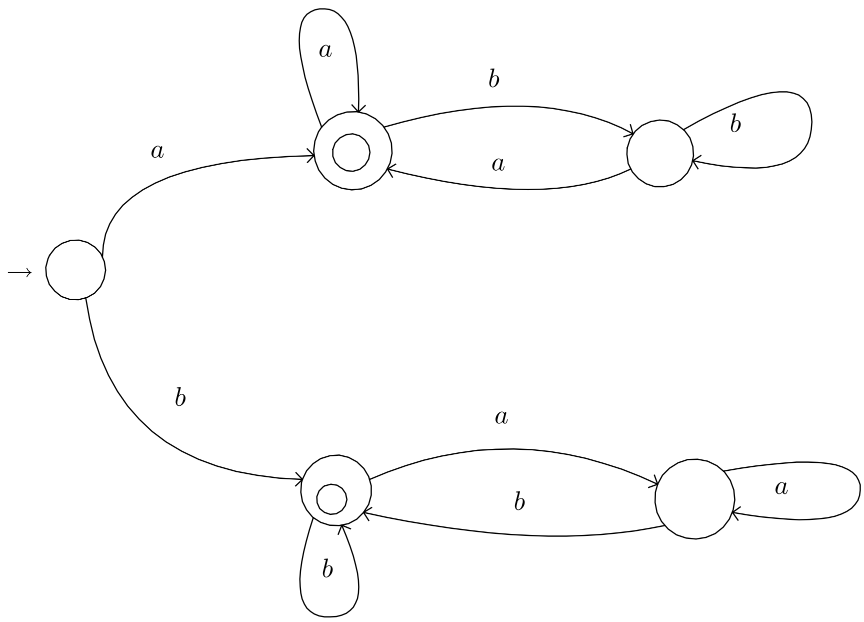
Let's define a language, such that $L_1(A) = \{\text{set of strings over } \Sigma, \text{accepted by } A\}$.

***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, $L_4 = L_2 \cup L_3$.

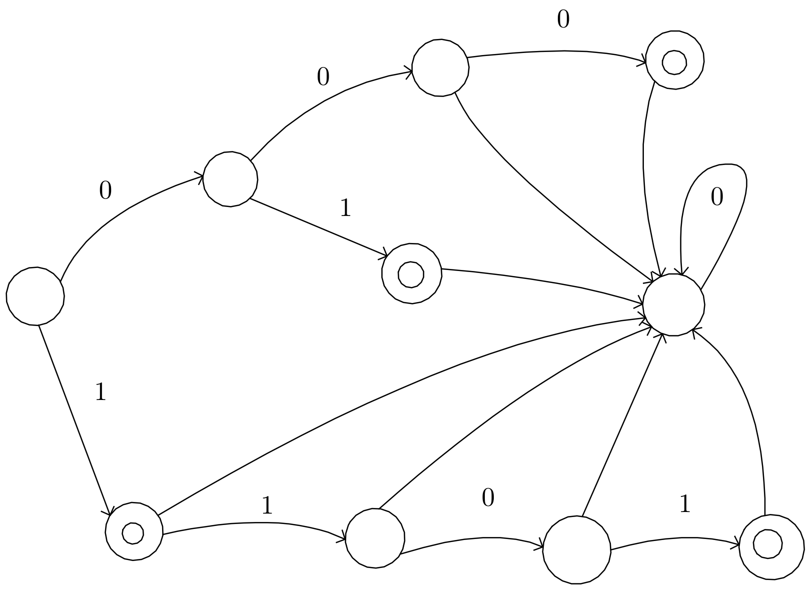
And such we can also define $\overline{L_1} = Q / F$.

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



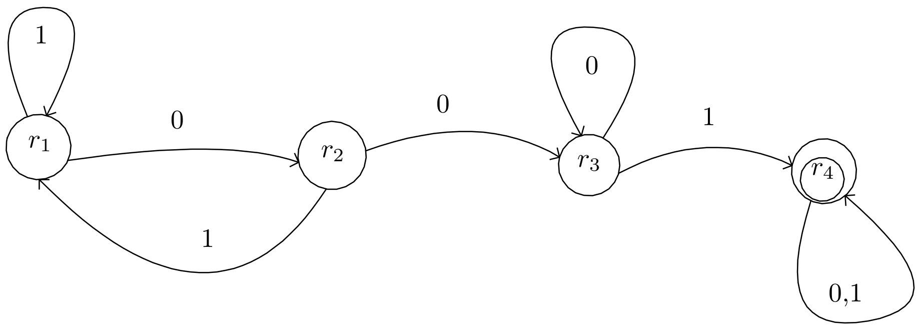

**Example 3.** Given the previous language, $L(A_5) = \{w \in \Sigma | \text{odd number of 1s}\}$, we say:

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

$L(A_6) = \{w \in \Sigma | w \,\text{contains}\, a \,\text{pattern}\, 001\}$. 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 $\epsilon$ (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, $\Sigma$. $\Sigma, Q = Q_1 \times Q_2 \sim (x, y) : x \in Q_1, y \in Q_2$.

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. $L_1 \circ L_2$

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

***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)

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

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.

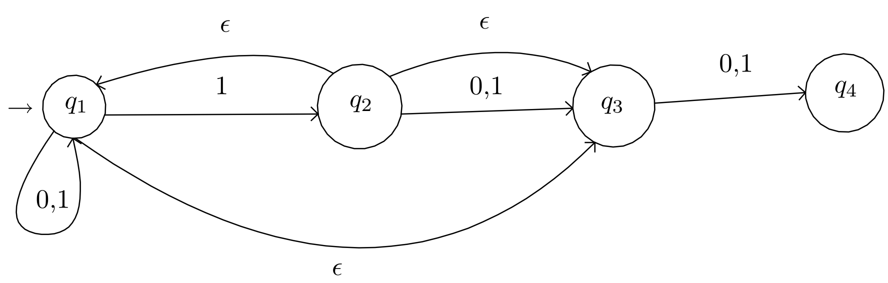
Let's look at a new language:

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

Ex. 0100<u>1</u>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, $\delta$.

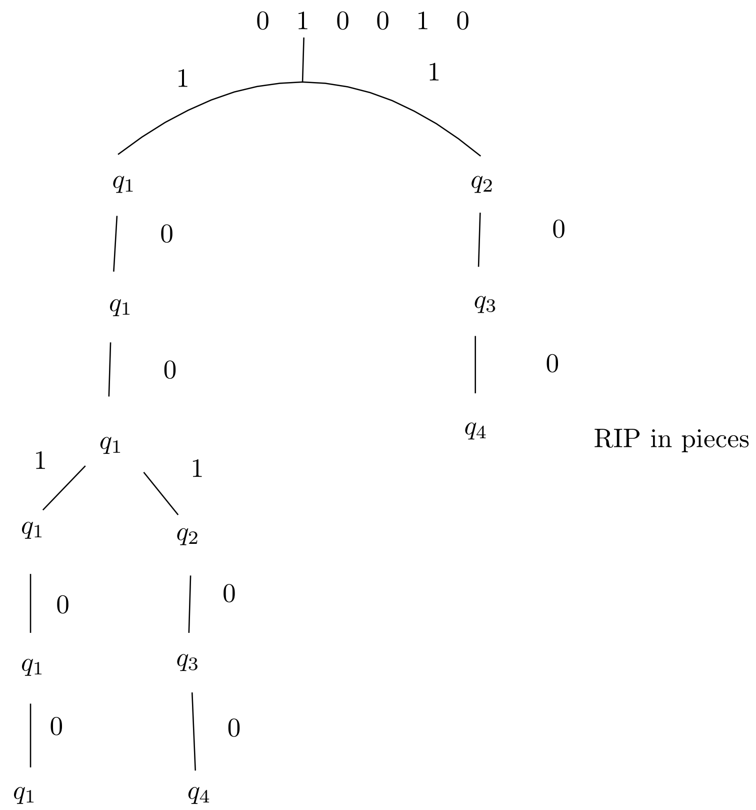
**Example 5.** We have $L_{11} = \{w \in (0,1)^* | \text{3rd number from right is } 1\}$. We can have the automata:




In the above automata, we can build the path $\epsilon$ (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 ($\epsilon$). Final accepting states link back to second state, not starting states.

**Example 6.** We can look at 0100100.

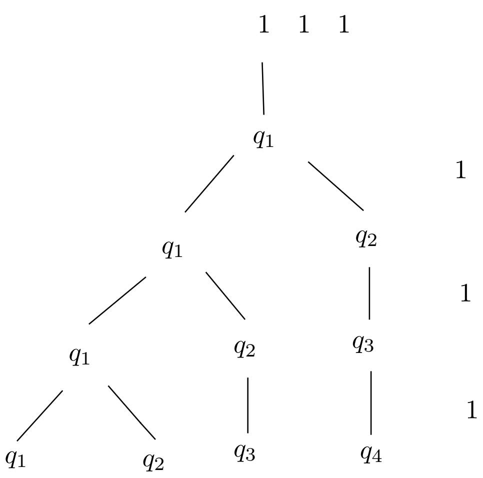



The transition function will now look like:

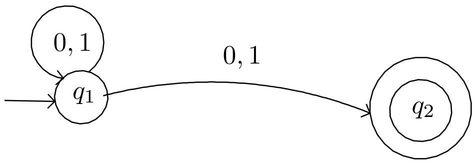
$$\delta \ : \ Q \times \Sigma_\epsilon \to P(Q)$$

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

**Example 7.** We take look at:



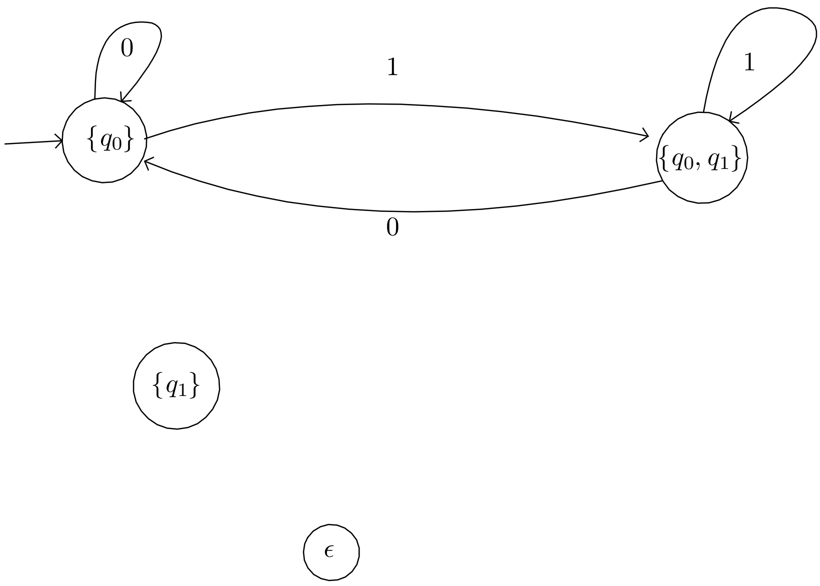
**Example 8.** We have the automata:




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

From there we can derive:

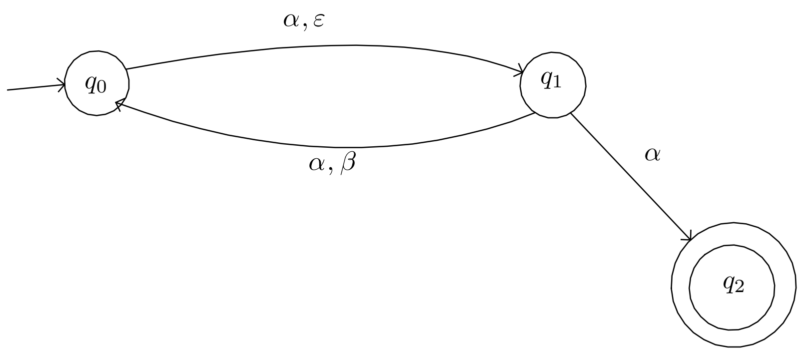
| | 0 | 1 |
|---|---|---|
| $\emptyset$ | - | - |
| $\rightarrow\{q_0\}$ | $\{q_0\}$ | $\{q_0, q_1\}$ |
| $^*\{q_1\}$ | $\emptyset$ | $\emptyset$ |
| $^*\{q_0, q_1\}$ | $\{q_0\}$ | $\{q_1\}$ |



We can look at the differences between automata.

**Example 9.** Given the matrix and automata:

| | $\alpha$ | $\beta$ | $\varepsilon$ |
|---|---|---|---|
| $q_0$ | $\{q_1\}$ | $\emptyset$ | $\{q_1\}$ |
| $q_1$ | $\{q_1, q_2\}$ | $\{q_0\}$ | $\emptyset$ |
| $q_2$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |




Let's look at our new state matrix:

| | $\alpha$ | $\beta$ |
|---|---|---|
| $\emptyset$ | | |
| $\{q_0\}$ | | |
| $\{q_1\}$ | | |
| $^*\{q_2\}$ | | |
| $\rightarrow\{q_0, q_1\}$ | $\{q_0, q_1, q_2\}$ | $\{q_0, q_1\}$ |
| $^*\{q_0, q_2\}$ | | |
| $^*\{q_1, q_2\}$ | | |
| $\circledast\{q_0, q_1, q_2\}$ | $\{q_0, q_1, q_2\}$ | $\{q_1, q_2\}$ |

## 2.1 DFA

We can define a generic DFA to be $\{\Sigma, Q, \delta\colon Q \times \Sigma \rightarrow Q, q_0, F\}$. Given $w = w_1 w_2 \ldots w_n\colon w_i \in \Sigma$. $\exists$ a sequence of states $r_0, r_1, \ldots, r_n$ where $r_i \in Q$.

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

## 2.2 NFA

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

$\exists\ y = y_1 y_2 ... y_m$ 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 $\delta(R, x) = \underset{r \in R}{\cup} \delta(z, x)$. 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 $\epsilon$ 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:

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

# 3 Regular Expressions

RegExp $\cong$ DFA $\leftarrow$ NFA.

Let's define a few things:

1. $a \in \Sigma$ (path $a$ to accepting state)
2. $\varepsilon$ (points to accepting state)
3. $\emptyset$ (points to null node)
4. $e_1 \cup e_2$ (create new node that has to path, $e \to e_1 | e_2$
5. $e_1 \circ e_2$ (replace the last accepting state of $e_1$ and point an $\varepsilon$ to the start state of $e_2$; keep in mind that only accepting states of $e_1$ will do: $\varepsilon \to \text{start}(e_2)$)

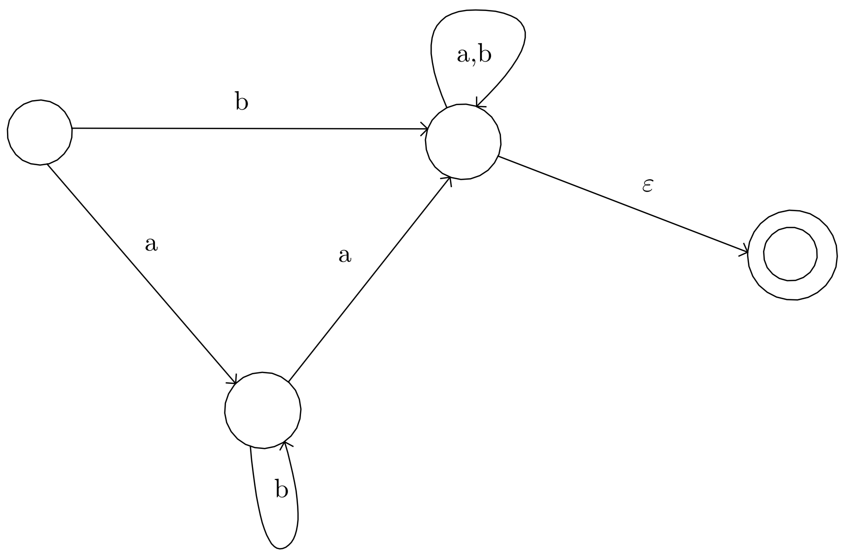
   ***Note:*** When you concat automata for RegExp, the accepting states at the end of the first automata are no longer accepting states.

## 3.1 DFA→NFA

In GNFA, we allow the transition state to be a RegExp.

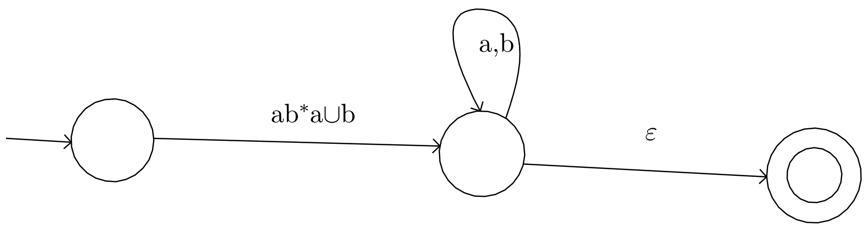
1. Start state only has outbound directed edges (otherwise, add new start state with only $\varepsilon$-edge out)
2. $\exists$ 1 accept state and no outbound edges
3. The automata is strongly connected as long as it follows 1 and 2

**Example 10.** Given automata:




This automata accepts string that start with b or has at least 2 a's.

We can produce the regex:



And thus, ab*a∪b(a∪b*) ∈ Σ.

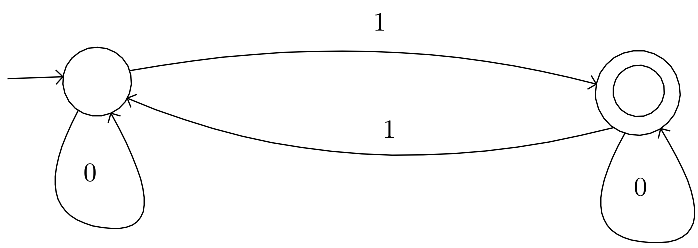
## 4 Pumping Lemma

If $L \in R$ (set of regular languages), then $\exists\ p > 0\ \forall\ s \in L$, $|s| \geq p$ can be written as $s = x\,y\,z$ with:

1. $|y| > 0$
2. $|x\,y| \leq p$ (length of string $=$ $p$)
3. $\forall\ i = 0, 1, 2, \ldots\ x\,y^i\,z \in L$

$\exists$ DFA: $L(\text{DFA}) \equiv L$ and $p \geq |Q|$ (thus, there must be some repetition).

**Example 11.** $L =$




We have $p = 2$.

- $|01| \geq p$ with:
  - $x = \varepsilon$
  - $y = 0$ (and $\therefore$ $|y| > 0$)
  - $z = 1$
- $|10| \geq p$ with:
  - $x = \varepsilon$
  - $y = 0$ (the path "0") (and $\therefore$ $|y| > 0$)
  - $z = \varepsilon$

The lemma holds for a finite languages as long as $p$ is greater than the length of the longest word.

$\therefore$ The Pumping Lemma holds for all regular languages.

This lemma can also be used to prove that a language is not regular.

$\forall\ p > 0$, $\exists\ s \in L$ with $|s| \geq p$ : no matter how you split, the string $s = x\,y\,z$ has:

1. $|x\,y| \leq p$
2. $|y| > 0$
3. $\exists\ i = 0, 1, 2, \ldots : x\,y^i\,z \notin L$

**Example 12.** Given $L = \{0^n 1^n | n > 0\}$

We'll have $|0^i| = |1^i| = p$. If we were to split, all of $|x\,y|$ would be in the first set. Then, if we pumped $y$ up, it would change the cardinality: $|0^{i+n}| > |1^i|$ and this would break the rule of our language.

**Example 13.** $L = \{1^{n^2} | n \geq 1\}$

We have an arbitrary $p$ where $1^{p^2}$:

$$\begin{aligned} |x\,y\,y\,z| &= |x\,y\,z| + |y| \\ &\leq p^2 + p \end{aligned}$$

where $|x\,y\,z| = p^2$ and $|y| = p$. This violates our language, since the next $p$ will be $1^{(1+p)^2}$.

Since $(1+p)^2 = p^2 + 2p + 1 > p^2 + 2p > p^2 + p$, this cannot be in the language.

Basically, given an arbitrary $p$, we can pump up once and get a string outside of the language.

Given $L_1$ and $L_2$:

$$\begin{aligned} L_1 &= \{0^n 1^n | n \geq 0\} \\ L_1 &\notin R \\ L_2 &= \{0^* 1^*\} \\ L_2 &\in R \end{aligned}$$

There exists properties:

- $L_1 \cup L_2$
- $L_1 \cap L_2$
- $L_1 \circ L_2$
- $L_1^*$
- $\overline{L_1}$

# 5 Context-Free Languages

We can define a $G = \{\Sigma, V, R, S\}$ (context-free grammar) where $\Sigma$ are terminals (digits).

$\Sigma \cap V = \emptyset$

$V$ is a set of variables (non-terminals); $S \in V$.

$R$ is the set of rewrite variables; $R: X \to \gamma$. $(X \in V,\ \gamma \in (\Sigma \cup V)^*)$

We can derive:

$$\begin{aligned} \alpha X \beta &\Rightarrow \alpha \gamma \beta \\ \text{where} \quad & \alpha, \beta, \gamma \in (\Sigma \cup V)^* \end{aligned}$$

We can do:

$$\alpha \overset{*}{\Rightarrow} \beta$$

where $\alpha = \alpha_0, a_1, \ldots, \alpha_n = \beta$. Finally we can show:

$$L(G) = \left\{ \alpha \in \Sigma^* | S \overset{*}{\Rightarrow} \alpha \right\}$$

"The language is a set of terminals that can be derived from start string symbols."

**Example 14.** $\Sigma = \{0, 1\}, V = \{S\}, S \to 01 \,\text{or}\, S \to 0S1$

***Note:*** Keep in mind, we can apply any rule as many times as we want.

Let's look at both rules:

$$\begin{aligned} S &\Rightarrow 0S1\,(\text{sentential form}) \\ &\Rightarrow 00S11 \\ &\Rightarrow 000111 \end{aligned}$$

Thus, we will end up with the form: $\underset{n}{0\ldots0}S\underset{n}{1\ldots1}$, where $S$ can be replaced as such: 01.

Let's look at Chomsky's hierarchy of grammars:

- **Type-0**: recursively enumerable grammars

  $$\alpha \Rightarrow \beta$$

  where $\alpha, \beta \in (V \cup \Sigma)^*$.

- **Type-1**: context-sensitive grammars produce *context-sensitive languages*

  $$\alpha X \beta \Rightarrow \alpha \gamma \beta$$

  $\gamma$ cannot be empty. With $X$ a nonterminal and $\alpha, \beta, \gamma$ are strings of terminals or nonterminals.

- **Type-2**: context-free grammars produce *context-free languages*

  $$X \Rightarrow \gamma$$

  with $X$ a nonterminal $\gamma$ a string of terminals or non-terminals.

- **Type-3**: regular grammars produce *regular languages*

***Note***: The union of two CFL's is context-free.

**Example 15.** Given $L = \{w|w_j = \{0,1\} \text{ with } \#0 = \#1\}$. Also, given the rules: $S \rightarrow 0S1|1S0|SS|\varepsilon$

"Produces strings that start with 0 or 1; where $s$ is a string that has an equal number of 0's and 1's and ends with 0,1 respectively. But 01110010 is in the language but not in the context-free grammer.

***Note***: A language, $L$, is context-free is we can derive it from a CFG.

$$\begin{aligned} S_1 &\rightarrow V_1 \\ S_2 &\rightarrow V_2 \\ &\therefore \\ S &\rightarrow S_1 S_2 \end{aligned}$$

Based on the above mini proof, we have that a CFL has $\circ$, $\cup$, $^*$. Intersection ($\cap$) is not available.

Given something like $L = \{0^n 1^n 2^n\} \cap \{0^n 1^n 2^m\}$ with respective rules:

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow 0A1|\varepsilon \\ B &\rightarrow 2B|\varepsilon \end{aligned}$$

and

$$\begin{aligned} U &\rightarrow XY \\ X &\rightarrow 0X|\varepsilon \\ Y &\rightarrow 1YZ|\varepsilon \end{aligned}$$

The result $L$, is not a context-free language.

⊛***BUT***⊛ if we have a regular language intersected with a CFL, that resulting language is context-free. See in the textbook how the derivation can take place with trees.