Kleene’s Theorem

**Definition.** A language is **regular** iff it is equal to $L(M)$, the set of strings accepted by some deterministic finite automaton $M$.

**Theorem.**

(a) For any regular expression $r$, the set $L(r)$ of strings matching $r$ is a regular language.

(b) Conversely, every regular language is the form $L(r)$ for some regular expression $r$. 

Example of a regular language

Recall the example DFA we used earlier:

$$M \triangleq q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} q_3$$

In this case it’s not hard to see that $L(M) = L(r)$ for

$$r = (a|b)^* aaa(a|b)^*$$
Example

\( M \triangleq \)

\( L(M) = L(r) \) for which regular expression \( r \)?

Guess: \( r = a^* | a^* b(ab)^* aaaa^* \)
Example

$L(M) = L(r)$ for which regular expression $r$?

Guess: $r = a^* | a^* b(ab)^* aaaa^*$

WRONG! since $baabaa \in L(M)$

but $baabaa \notin L(a^* | a^* b(ab)^* aaaa^*)$

We need an algorithm for constructing a suitable $r$ for each $M$ (plus a proof that it is correct).
Lemma. Given an NFA $M = (Q, \Sigma, \Delta, s, F)$, for each subset $S \subseteq Q$ and each pair of states $q, q' \in Q$, there is a regular expression $r^{S}_{q, q'}$ satisfying

$$L(r^{S}_{q, q'}) = \{ u \in \Sigma^* \mid q \xrightarrow{u}^* q' \text{ in } M \text{ with all intermediate states of the sequence of transitions in } S \}.$$ 

Hence if the subset $F$ of accepting states has $k$ distinct elements, $q_1, \ldots, q_k$ say, then $L(M) = L(r)$ with $r \triangleq r_1 \mid \cdots \mid r_k$ where

$$r_i = r^{O}_{s, q_i} \quad (i = 1, \ldots, k)$$

(in case $k = 0$, we take $r$ to be the regular expression $\emptyset$).
Lemma on p23 is proved by induction on \# of elements in $S$

Base case $S = \emptyset$:

Given states $q, q'$ in $M$, if
\[ q \xmapsto{a} q' \]
holds for just $a = a_1, \ldots, a_k$ then can take

\[
\begin{align*}
\emptyset & \triangleq \{ a_1 \ldots a_k \} & \text{if } q \neq q' \\
\{ a_1 \ldots a_k \} \varepsilon & \text{if } q = q'
\end{align*}
\]
Lemma on p83 is proved by induction on \# of elements in S

Base case $S = \emptyset$:

Given states $q, q'$ in $M$, if $q \xrightarrow{a} q'$ holds for no $a$ then can take

$$\emptyset, q, q' = \begin{cases} \emptyset & \text{if } q \neq q' \\ \emptyset & \text{if } q = q' \end{cases}$$
Induction step: $S$ has $n+1$ elements.

Pick any $q_0 \in S$. So can apply induction hyp. to $S \setminus \{q_0\} = \{q \in S \mid q \neq q_0\}$ since it has $n$ elts.
Induction step: $S$ has $n+1$ elements.

Pick any $q_0 \in S$. So can apply induction hyp. to $S \setminus \{q_0\} = \{q \in S | q \neq q_0\}$ since it has $n$ elts.
Induction step: \( S \) has \( n+1 \) elements.

Pick any \( q_0 \in S \). So can apply induction hyp. to \( S \setminus \{q_0\} = \{q \in S \mid q \neq q_0\} \) since it has \( n \) elts.

\[
\begin{align*}
\sigma_{q, q'} &= \sigma_{S \setminus \{q_0\}, q, q'} \\
&\ldots
\end{align*}
\]
Induction step: $S$ has $n+1$ elements. Pick any $q_0 \in S$. So can apply induction hyp. to $S \setminus \{q_0\} = \{q \in S \mid q \neq q_0\}$ since it has $n$ elts.

\[
\begin{align*}
\begin{aligned}
\require{AMScd}
\begin{CD}
M @>S>> S
\end{CD}
\end{aligned}
\end{align*}
\]

\[
\begin{align*}
q_0
\end{align*}
\]

\[
\begin{align*}
\begin{aligned}
r_{S}^{q,q'} &= r_{S \setminus \{q_0\}}^{q,q'} \mid r_{S \setminus \{q_0\}}^{q,q_0} (r_{S \setminus \{q_0\}}^{q_0,q_0} (r_{S \setminus \{q_0\}}^{q_0,q'}))
\end{aligned}
\end{align*}
\]
**Induction step:** $S$ has $n+1$ elements.

Pick any $q_0 \in S$. So can apply induction hyp. to $S \backslash \{q_0\} = \{q \in S \mid q \neq q_0\}$ since it has $n$ elts.

\[
\begin{align*}
    r_{q,q'}^S &= r_{q,q'}^{S \backslash \{q_0\}} \mid r_{q,q_0}^{S \backslash \{q_0\}} (r_{q_0,q_0}^{S \backslash \{q_0\}})^* r_{q_0,q'}^{S \backslash \{q_0\}} 
\end{align*}
\]
Induction step: $S$ has $n+1$ elements.

Pick any $q_0 \in S$. So can apply induction hyp.

to $S \setminus \{q_0\} = \{q \in S | q \neq q_0\}$ since it has $n$ elts.

$$r_{S, q, q'} = r_{S \setminus \{q_0\}, q, q'} \mid r_{S \setminus \{q_0\}, q, q_0} \left( r_{S \setminus \{q_0\}, q_0, q_0} \right)^* \mid r_{S \setminus \{q_0\}, q_0, q'}$$
**Induction step**: $S$ has $n+1$ elements

Pick any $q_0 \in S$. So can apply induction hyp. to $S \setminus \{q_0\} = \{q \in S | q \neq q_0\}$ since it has $n$ elts.

\[
rs_{q,q'} = rs_{q,q'} \mid rs_{q_0,q'} (rs_{q_0,q_0})^* rs_{q_0,q_0}
\]
By direct inspection we have:

\[
\begin{array}{c|ccc}
  \{0\} & 0 & 1 & 2 \\
  \hline
  0 & \emptyset & \varepsilon & a \\
  1 & aa^* & a^*b & \varepsilon \\
  2 & & & \\
\end{array}
\]

\[
\begin{array}{c|ccc}
  \{0,2\} & 0 & 1 & 2 \\
  \hline
  0 & a^* & a^*b \\
  1 & & & \\
  2 & & & \\
\end{array}
\]

(we don’t need the unfilled entries in the tables)
Example p 87

Remove 1 from $\{0, 1, 2\}$

$\{0, 1, 2\} \not\Delta \{0, 1\} \cup \{0, 1\} \cup \{0, 1\}$

$a^* \quad a^* b$
Example p 87

Want $r_{0,0}^{\{0,1,2\}}$

\[ r_{0,0}^{\{0,1,2\}} = a^* \mid a^* b (r_{1,1}^{\{0,2\}})^* r_{1,0}^{\{0,1\}} \]

\[ r_{1,1}^{\{0,2\}} \triangleq r_{1,1}^{\{0\}} \mid r_{1,1}^{\{0\}} (r_{2,2}^{\{0\}})^* r_{2,1}^{\{0\}} \]

\[ = \varepsilon \mid a (\varepsilon)^* a^* b \]

\[ = \varepsilon \mid a a^* b \]

**equivalence:** $r = s \triangleq \mathcal{L}(r) = \mathcal{L}(s)$
Example p 87

$$r_{0,0}^{\{0,1,2\}} = a^* | a^* b (\varepsilon | a a^* b)^* r_{1,0}^{\{0,1\}}$$

$$r_{1,0}^{\{0,2\}} \triangleq r_{1,0}^{\{0\}} | r_{1,2}^{\{0\}} (r_{2,2}^{\{0\}})^* r_{2,0}^{\{0\}}$$

$$= \emptyset | a (\varepsilon)^* a a^*$$

$$= a a a^*$$
Example p 87

Want \( r_{0,0}^{0,1,2} \)

\[
r_{0,0}^{0,1,2} = a^* | a^* b (\varepsilon | a a^* b)^* a a d^* \]

Some questions

(a) Is there an algorithm which, given a string $u$ and a regular expression $r$, computes whether or not $u$ matches $r$?

(b) In formulating the definition of regular expressions, have we missed out some practically useful notions of pattern?

(c) Is there an algorithm which, given two regular expressions $r$ and $s$, computes whether or not they are equivalent, in the sense that $L(r)$ and $L(s)$ are equal sets?

(d) Is every language (subset of $\Sigma^*$) of the form $L(r)$ for some $r$?
Given DFA $M = (Q, \Sigma, \delta, s, F)$, then $\text{Not}(M)$ is the DFA with

- set of states $= Q$
- input alphabet $= \Sigma$
- next-state function $= \delta$
- start state $= s$
- accepting states $= \{q \in Q \mid q \not\in F\}$.

(i.e. we just reverse the role of accepting/non-accepting and leave everything else the same)

Because $M$ is a deterministic finite automaton, then $u$ is accepted by $\text{Not}(M)$ iff it is not accepted by $M$:

$$L(\text{Not}(M)) = \{u \in \Sigma^* \mid u \not\in L(M)\}$$
Given reg. exp. \( r \), can construct reg. exp. \( \sim r \) such that

\[
L(\sim r) = \{ u \in \Sigma^* \mid u \notin L(r) \}
\]
Given reg. exp. $r$, can construct reg. exp. $\sim r$ such that

$$L(\sim r) = \{n \in \Sigma^* | n \notin L(r)\}$$

Kleene ($a$)

$\xrightarrow{r} M$

$L(M) = L(r)$
Given regex $r$, can construct regex $\sim r$ such that:

$$L(\sim r) = \{ u \in \Sigma^* | u \notin L(r) \}$$

Kleene (a) $r \xrightarrow{\text{Kleene (a)}} M$ $L(M) = L(r)$

Kleene (b) $\text{Not}(M) \xrightarrow{\text{Kleene (b)}} \sim r$ $L(\sim r) = L(\text{Not}(M))$
Given reg. exp. $r$, can construct reg. exp. $\sim r$ such that

$$L(\sim r) = \{u \in \Sigma^* | u \notin L(r)\}$$

Kleene (a)

$$r \xrightarrow{L(M)=L(r)} M \xrightarrow{Not(M)} \sim r \xrightarrow{L(Not(M))}$$

Kleene (b)

$$L(\sim r) = L(Not(M)) = \Sigma^* \setminus L(r) = \Sigma^* \setminus L(M)$$

so: $$L(\sim r) = L(Not(M)) = \Sigma^* \setminus L(M) = \Sigma^* \setminus L(r)$$
Regular languages are closed under intersection

**Theorem.** If $L_1$ and $L_2$ are a regular languages over an alphabet $\Sigma$, then their intersection $L_1 \cap L_2 = \{ u \in \Sigma^* \mid u \in L_1 \land u \in L_2 \}$ is also regular.

**Proof.** Note that $L_1 \cap L_2 = \Sigma^* \setminus ((\Sigma^* \setminus L_1) \cup (\Sigma^* \setminus L_2))$

(cf. de Morgan’s Law: $p \land q = \neg(\neg p \lor \neg q)$).

So if $L_1 = L(M_1)$ and $L_2 = L(M_2)$ for DFAs $M_1$ and $M_2$, then $L_1 \cap L_2 = L(\text{Not}(PM))$ where $M$ is the NFA $\varepsilon$ Union$(\text{Not}(M_1), \text{Not}(M_2))$.

[It is not hard to directly construct a DFA $\text{And}(M_1, M_2)$ from $M_1$ and $M_2$ such that $L(\text{And}(M_1, M_2)) = L(M_1) \cap L(M_2)$ – see Exercise 4.7.]
Regular languages are closed under intersection

**Corollary:** given regular expressions $r_1$ and $r_2$, there is a regular expression, which we write as $r_1 \& r_2$, such that a string $u$ matches $r_1 \& r_2$ iff it matches both $r_1$ and $r_2$.

**Proof.** By Kleene (a), $L(r_1)$ and $L(r_2)$ are regular languages and hence by the theorem, so is $L(r_1) \cap L(r_2)$. Then we can use Kleene (b) to construct a regular expression $r_1 \& r_2$ with $L(r_1 \& r_2) = L(r_1) \cap L(r_2)$.

□