Discrete Mathematics

Lecture 23

Regular Languages

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 of the form $L(r)$ for some regular expression r.

Kleene part (a): from regular expressions to automata

Given a regular expression r, over an alphabet Σ say, we wish to construct a DFA M with alphabet of input symbols Σ and with the property that for each $u \in \Sigma^*$, u matches r iff u is accepted by M, so that $L(r) = L(M)$.

Note that by the Theorem on Slide 60 it is enough to construct an NFA$^\varepsilon$ N with the property $L(N) = L(r)$. For then we can apply the subset construction to N to obtain a DFA $M = PN$ with $L(M) = L(PN) = L(N) = L(r)$. Working with finite automata that are non-deterministic and have ε-transitions simplifies the construction of a suitable finite automaton from r.

Let us fix on a particular alphabet Σ and from now on only consider finite automata whose set of input symbols is Σ.

The construction of an NFA$^\varepsilon$ for each regular expression r over Σ proceeds by *induction on the size (= number of vertices) of regular expression abstract syntax trees*, as indicated on the next slide. Thus starting with step (i) and applying the constructions in steps (ii)–(iv) over and over again, we eventually build NFA$^\varepsilon$s with the required property for every regular expression r.

Put more formally, one can prove the statement

> *for all $n \geq 0$, and for all regular expressions abstract syntax trees of size $\leq n$, there exists an NFA$^\varepsilon$ M such that $L(r) = L(M)$*

by mathematical induction on n, using step (i) for the base case and steps (ii)–(iv) for the induction steps.

(i) **Base cases:** show that $\{a\}$, $\{\varepsilon\}$ and $\emptyset$ are regular languages.

(ii) **Induction step for $r_1|r_2$:** given NFA$^\varepsilon$s M_1 and M_2, construct an NFA$^\varepsilon$ $Union(M_1, M_2)$ satisfying

$$\boxed{L(Union(M_1, M_2)) = \{u \mid u \in L(M_1) \vee u \in L(M_2)\}}$$

Thus if $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$, then $L(r_1|r_2) = L(Union(M_1, M_2))$.

(iii) **Induction step for r_1r_2:** given NFA$^\varepsilon$s M_1 and M_2, construct an NFA$^\varepsilon$ $Concat(M_1, M_2)$ satisfying

$$\boxed{\begin{aligned} L(Concat(M_1, M_2)) = \{u_1u_2 \mid u_1 \in L(M_1)\ \&\\ u_2 \in L(M_2)\}\end{aligned}}$$

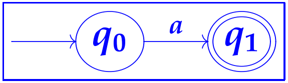
Thus $L(r_1r_2) = L(Concat(M_1, M_2))$ when $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$.

(iv) **Induction step for r^*:** given NFA$^\varepsilon$ M, construct an NFA$^\varepsilon$ $Star(M)$ satisfying

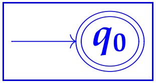
$$\boxed{L(Star(M)) = \{u_1u_2 \ldots u_n \mid n \geq 0 \text{ and each } u_i \in L(M)\}}$$

Thus $L(r^*) = L(Star(M))$ when $L(r) = L(M)$.

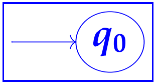
NFAs for regular expressions a, ϵ, $\emptyset$



just accepts the one-symbol string a



just accepts the null string, ε

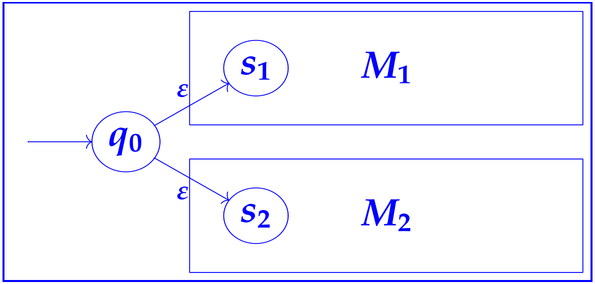


accepts no strings

Union(M_1, M_2)

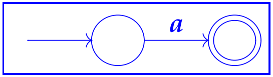
$M_1 = (Q_1, \Sigma, \Delta_1, s_1, F_1)$

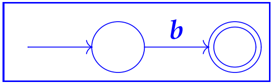
$M_2 = (Q_2, \Sigma, \Delta_2, s_2, F_2)$

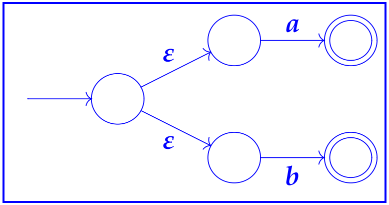


accepting states = union of accepting states of M_1 and M_2

For example,

if $M_a =$ 

and $M_b =$ 

then $Union(M_a, M_b) =$ 

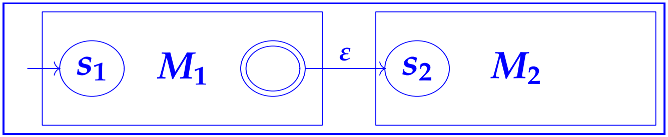
Induction step for $r_1|r_2$

Given NFA$^\varepsilon$s $M_1 = (Q_1, \Sigma, \Delta_1, s_1, T_1)$ and $M_2 = (Q_2, \Sigma, \Delta_2, s_2, T_2)$, the construction of $Union(M_1, M_2)$ is pictured on Slide 68. First, renaming states if necessary, we assume that $Q_1 \cap Q_2 = \emptyset$. Then the states of $Union(M_1, M_2)$ are all the states in either Q_1 or Q_2, together with a new state, called q_0 say. The start state of $Union(M_1, M_2)$ is this q_0 and its set of accepting states is the union $F_1 \cup F_2$ of the sets of accepting states in M_1 and M_2. Finally, the transitions of $Union(M_1, M_2)$ are given by all those in either M_1 or M_2, together with two new ε-transitions out of q_0, one to the start states s_1 of M_1 and one to the start state s_2 of M_2.

Thus if $u \in L(M_1)$, i.e. if we have $s_1 \overset{u}{\Rightarrow} q_1$ for some $q_1 \in F_1$, then we get $q_0 \xrightarrow{\varepsilon} s_1 \overset{u}{\Rightarrow} q_1$ showing that $u \in L(Union(M_1, M_2))$. Similarly for M_2. So $L(Union(M_1, M_2))$ contains the union of $L(M_1)$ and $L(M_2)$. Conversely if u is accepted by $Union(M_1, M_2)$, there is a transition sequence $q_0 \overset{u}{\Rightarrow} q$ with $q \in F_1$ or $q \in F_2$. Clearly, in either case this transition sequence has to begin with one or other of the ε-transitions from q_0, and thereafter we get a transition sequence entirely in one or other of M_1 or M_2 (*because we assumed that Q_1 and Q_2 are disjoint*) finishing in an acceptable state for that one. So if $u \in L(Union(M_1, M_2))$, then either $u \in L(M_1)$ or $u \in L(M_2)$. So we do indeed have

$$L(Union(M_1, M_2)) = \{u \mid u \in L(M_1) \vee u \in L(M_2)\}$$

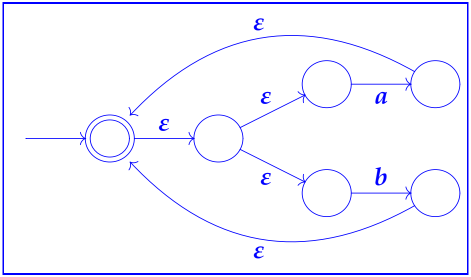
$Concat(M_1, M_2)$



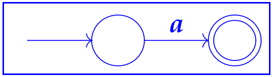
accepting states are those of M_2

For example,

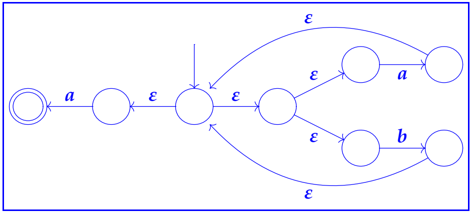
if $M_1 =$



and $M_2 =$



then $Concat(M_1, M_2) =$



Induction step for $r_1 r_2$

Given NFA$^\varepsilon$s $M_1 = (Q_1, \Sigma, \Delta_1, s_1, T_1)$ and $M_2 = (Q_2, \Sigma, \Delta_2, s_2, T_2)$, the construction of $Concat(M_1, M_2)$ is pictured on Slide 71. First, renaming states if necessary, we assume that $Q_1 \cap Q_2 = \emptyset$. Then the set of states of $Concat(M_1, M_2)$ is $Q_1 \cup Q_2$. The start state of $Concat(M_1, M_2)$ is the start state s_1 of M_1. The set of accepting states of $Concat(M_1, M_2)$ is the set F_2 of accepting states of M_2. Finally, the transitions of $Concat(M_1, M_2)$ are given by all those in either M_1 or M_2, together with new ε-transitions from each accepting state of M_1 to the start state s_2 of M_2 (only one such new transition is shown in the picture).

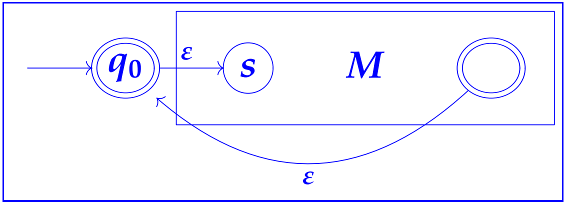
Thus if $u_1 \in L(M_1)$ and $u_2 \in L(M_2)$, there are transition sequences $s_1 \overset{u_1}{\Rightarrow} q_1$ in M_1 with $q_1 \in F_1$, and $s_2 \overset{u_2}{\Rightarrow} q_2$ in M_2 with $q_2 \in F_2$. These combine to yield

$$s_1 \overset{u_1}{\Rightarrow} q_1 \xrightarrow{\varepsilon} s_2 \overset{u_2}{\Rightarrow} q_2$$

in $Concat(M_1, M_2)$ witnessing the fact that $u_1 u_2$ is accepted by $Concat(M_1, M_2)$. Conversely, it is not hard to see that every $v \in L(Concat(M_1, M_2))$ is of this form: for any transition sequence witnessing the fact that v is accepted starts out in the states of M_1 but finishes in the disjoint set of states of M_2. At some point in the sequence one of the new ε-transitions occurs to get from M_1 to M_2 and thus we can split v as $v = u_1 u_2$ with u_1 accepted by M_1 and u_2 accepted by M_2. So we do indeed have

$$L(Concat(M_1, M_2)) = \{u_1 u_2 \mid u_1 \in L(M_1) \,\&\, u_2 \in L(M_2)\}$$

$Star(M)$

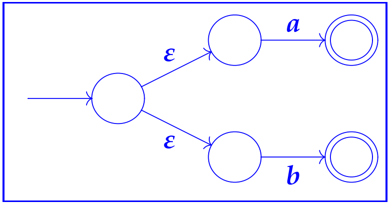


the only accepting state of $Star(M)$ is q_0

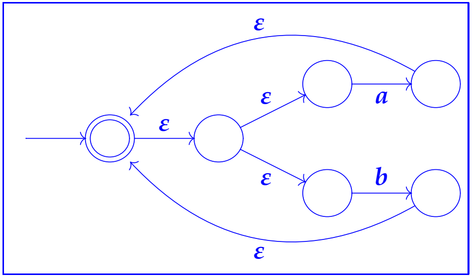
(N.B. doing without q_0 by just looping back to s
and making that accepting won't work – Exercise 4.1.)

For example,

if $M =$



then $Star(M) =$



Induction step for r^*

Given an NFA$^\varepsilon$ $M = (Q, \Sigma, \Delta, s, T)$, the construction of $Star(M)$ is pictured on Slide 74. The states of $Star(M)$ are all those of M together with a new state, called q_0 say. The start state of $Star(M)$ is q_0 and this is also the only accepting state of $Star(M)$. Finally, the transitions of $Star(M)$ are all those of M together with new ε-transitions from q_0 to the start state of M and from each accepting state of M to q_0 (only one of this latter kind of transition is shown in the picture).

Clearly, $Star(M)$ accepts ε (since its start state is accepting) and any concatenation of one or more strings accepted by M. Conversely, if v is accepted by $Star(M)$, the occurrences of q_0 in a transition sequence witnessing this fact allow us to split v into the concatenation of zero or more strings, each of which is accepted by M. So we do indeed have

$$L(Star(M)) = \{u_1 u_2 \ldots u_n \mid n \geq 0 \text{ and each } u_i \in L(M)\}$$

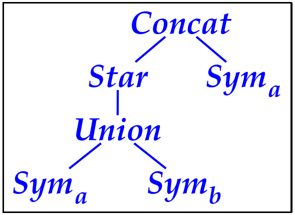
$\square$

This completes the proof of part (a) of Kleene's Theorem (Slide 64). Slide 77 shows how the step-by-step construction applies in the case of the regular expression $(a|b)^*a$ to produce an NFA$^\varepsilon$ M satisfying $L(M) = L((a|b)^*a)$. Of course an automaton with fewer states and ε-transitions doing the same job can be crafted by hand. The point of the construction is that it provides an automatic way of producing automata for any given regular expression.

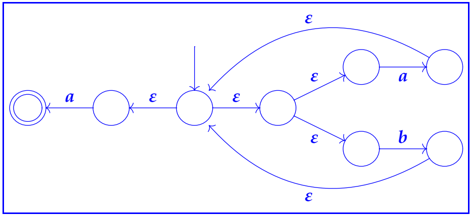
Example

Regular expression $(a|b)^* a$

whose abstract syntax tree is



is mapped to the NFA$^\varepsilon$ $Concat(Star(Union(M_a, M_b)), M_a) =$



(*cf.* Slides 69, 72 and 75).

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 Σ^*) of the form $L(r)$ for some r?

Decidability of matching

We now have a positive answer to question (a) on Slide 38. Given string u and regular expression r:

- construct an NFA$^{\varepsilon}$ M satisfying $L(M) = L(r)$;
- in PM (the DFA obtained by the subset construction, Slide 60) carry out the sequence of transitions corresponding to u from the start state to some state q (because PM is deterministic, there is a unique such transition sequence);
- check whether q is accepting or not: if it is, then $u \in L(PM) = L(M) = L(r)$, so u matches r; otherwise $u \notin L(PM) = L(M) = L(r)$, so u does not match r.

(The subset construction produces an exponential blow-up of the number of states: PM has 2^n states if M has n. This makes the method described above potentially inefficient – more efficient algorithms exist that don't construct the whole of PM.)

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 of the form $L(r)$ for some regular expression r.

Example of a regular language

Recall the example DFA we used earlier:

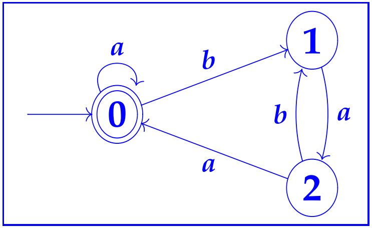
$$M \triangleq \begin{tikzcd} {} \arrow[r] & q_0 \arrow[r, "a"] \arrow[loop below, "b"] & q_1 \arrow[r, "a"] \arrow[l, bend right, "b"'] & q_2 \arrow[r, "a"] \arrow[ll, bend left, "b"] & \boxed{q_3} \arrow[loop above, "a"] \arrow[loop below, "b"] \end{tikzcd}$$

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

WRONG! since $baabaa \in L(M)$
but $baabaa \notin L(a^*|a^*b(ab)^*aaa^*)$

We need an algorithm for constructing a suitable r for each M (plus a proof that it is correct).

Kleene part (b): from automata to regular expressions

Given any DFA $M = (Q, \Sigma, \delta, s, F)$, we have to find a regular expression r (over the alphabet Σ of input symbols of M) satisfying $L(r) = L(M)$. In fact we do something more general than this, as described in the Lemma on Slide 84. Note that if we can find the regular expressions $r^{S}_{q,q'}$ mentioned in the lemma (for any choice of $S \subseteq Q$ and $q, q' \in Q$), then the problem is solved. For taking S to be the whole of Q and q to be the start state s, then by definition of $r^{Q}_{s,q'}$, a string u matches this regular expression iff there is a transition sequence $s \xrightarrow{u}{}^{*} q'$ in M. As q' ranges over the finitely many accepting states, $q_1, \ldots, q_k$ say, then we match exactly all the strings accepted by M. In other words the regular expression $r^{Q}_{s,q_1} | \cdots | r^{Q}_{s,q_k}$ has the property we want for part (b) of Kleene's Theorem. (In case $k = 0$, i.e. there are *no* accepting states in M, then $L(M)$ is empty and so we can use the regular expression $\emptyset$.)

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 $r \triangleq r_1 | \cdots | r_k$ where

$$r_i = r^Q_{s,q_i} \qquad (i = 1, \ldots, k)$$

(in case $k = 0$, we take r to be the regular expression $\emptyset$).

$r := r_1 | r_2 | \cdots$

Proof of the Lemma on Slide 84

The regular expression $r^S_{q,q'}$ can be constructed by induction on the number of elements in the subset S.

Base case, S is empty. In this case, for each pair of states q, q', we are looking for a regular expression to describe the set of strings

$$\{u \mid q \xrightarrow{u}{}^* q' \text{ with } \textit{no} \text{ intermediate states in the sequence of transitions}\}.$$

So each element of this set is either a single input symbol a (if $q \xrightarrow{a} q'$ holds in M) or possibly ε, in case $q = q'$. If there are no input symbols that take us from q to q' in M, we can simply take

$$r^{\emptyset}_{q,q'} \triangleq \begin{cases} \emptyset & \text{if } q \neq q' \\ \varepsilon & \text{if } q = q' \end{cases}$$

On the other hand, if there are some such input symbols, $a_1, \ldots, a_k$ say, we can take

$$r^{\emptyset}_{q,q'} \triangleq \begin{cases} a_1|\cdots|a_k & \text{if } q \neq q' \\ a_1|\cdots|a_k|\varepsilon & \text{if } q = q' \end{cases}$$

[**Notation:** given sets X and Y, we write $X \setminus Y$ for the set $\{x \in X \mid x \notin Y\}$ of elements of X that are not in Y and call it the *relative complement* of X by Y.]

Induction step. Suppose we have defined the required regular expressions for all subsets of states with n elements. If S is a subset with $n+1$ elements, choose some element $q_0 \in S$ and consider the n-element set $S \setminus \{q_0\} = \{q \in S \mid q \neq q_0\}$. Then for any pair of states $q, q' \in States_M$, by inductive hypothesis we have already constructed the regular expressions

$$r_1 \triangleq r^{S\setminus\{q_0\}}_{q,q'} \qquad r_2 \triangleq r^{S\setminus\{q_0\}}_{q,q_0} \qquad r_3 \triangleq r^{S\setminus\{q_0\}}_{q_0,q_0}, \qquad r_4 \triangleq r^{S\setminus\{q_0\}}_{q_0,q'}$$

Consider the regular expression

$$r \triangleq r_1 | r_2 (r_3)^* r_4$$

Clearly every string matching r is in the set

$$\{u \mid q \xrightarrow{u}{}^* q' \text{ with all intermediate states in the sequence of transitions in } S\}.$$

Conversely, if u is in this set, consider the number of times the sequence of transitions $q \xrightarrow{u}{}^* q'$ passes through state q_0. If this number is zero then $u \in L(r_1)$ (by definition of r_1). Otherwise this number is $k \geq 1$ and the sequence splits into $k+1$ pieces: the first piece is in $L(r_2)$ (as the sequence goes from q to the first occurrence of q_0), the next $k-1$ pieces are in $L(r_3)$ (as the sequence goes from one occurrence of q_0 to the next), and the last piece is in $L(r_4)$ (as the sequence goes from the last occurrence of q_0 to q'). So in this case u is in $L(r_2(r_3)^* r_4)$. So in either case u is in $L(r)$. So to complete the induction step we can define $r^S_{q,q'}$ to be this regular expression $r = r_1 | r_2 (r_3)^* r_4$. $\square$

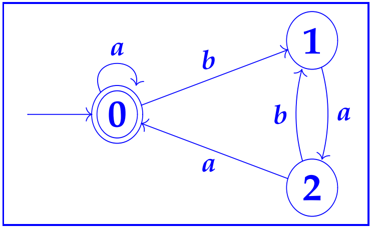
An example

We give an example to illustrate the construction of regular expressions from automata that is inherent in the above proof of part (b) of Kleene's Theorem. The example also demonstrates that we do not have to pursue the inductive construction of the regular expression to the bitter end (the base case $S = \emptyset$): often it is possible to find some of the regular expressions $r^S_{q,q'}$ one needs by *ad hoc* arguments – but if in doubt, *use the algorithm*.

Note also that at the inductive steps in the construction of a regular expression for M, we are free to choose which state q_0 to remove from the current state set S. A good rule of thumb is: *choose a state that disconnects the automaton as much as possible.*

$r^{\{0,1,2\}}_{0,0} = r^{\{0,2\}}_{0,0} \mid r^{\{0,2\}}_{0,1} \left(r^{\{0,2\}}_{1,1}\right)^* r^{\{0,2\}}_{1,0}$

$M \triangleq$



By direct inspection we have:

$r^{\{0\}}_{i,j}$	0	1	2
0			
1	$\emptyset$	ε	a
2	aa^*	a^*b	ε

$r^{\{0,2\}}_{i,j}$	0	1	2
0	a^*	a^*b	
1			
2			

(we don't need the unfilled entries in the tables)

Consider the NFA shown on Slide 88. Since the start state is $\mathbf{0}$ and this is also the only accepting state, the language of accepted strings is that determined by the regular expression $r_{0,0}^{\{0,1,2\}}$. Choosing to remove state $\mathbf{1}$ from the state set, we have

$$L(r_{0,0}^{\{0,1,2\}}) = L(r_{0,0}^{\{0,2\}} | r_{0,1}^{\{0,2\}} (r_{1,1}^{\{0,2\}})^* r_{1,0}^{\{0,2\}}) \tag{3}$$

Direct inspection shows that $L(r_{0,0}^{\{0,2\}}) = L(a^*)$ and $L(r_{0,1}^{\{0,2\}}) = L(a^*b)$. To calculate $L(r_{1,1}^{\{0,2\}})$, and $L(r_{1,0}^{\{0,2\}})$, we choose to remove state $\mathbf{2}$:

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

These regular expressions can all be determined by inspection, as shown on Slide 88. Thus $L(r_{1,1}^{\{0,2\}}) = L(\epsilon | a(\epsilon)^*(a^*b))$ and it's not hard to see that this is equal to $L(\epsilon | aa^*b)$. Similarly $L(r_{1,0}^{\{0,2\}}) = L(\emptyset | a(\epsilon)^*(aa^*))$ which is equal to $L(aaa^*)$. Substituting all these values into (3), we get

$$L(r_{0,0}^{\{0,1,2\}}) = L(a^* | a^*b(\epsilon | aa^*b)^* aaa^*)$$

So $a^* | a^*b(\epsilon | aa^*b)^* aaa^*$ is a regular expression whose matching strings comprise the language accepted by the NFA on Slide 88. (Clearly, one could simplify this to a smaller, but equivalent regular expression, but we do not bother to do so.)

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 Σ^*) of the form $L(r)$ for some r?

Regular languages are closed under complementation

Lemma. If L is a regular language over alphabet Σ, then its complement $\{u \in \Sigma^* \mid u \notin L\}$ is also regular.

Proof. Since L is regular, by definition there is a DFA M such that $L = L(M)$. Let $Not(M)$ be the DFA constructed from M as indicated on Slide 92. Then $\{u \in \Sigma^* \mid u \notin L\}$ is the set of strings accepted by $Not(M)$ and hence is regular. □

[**N.B.** If one applies the construction on Slide 92 (interchanging the role of accepting & non-accepting states) to a *non-deterministic* finite automaton N, then in general $L(Not(N))$ is not equal to $\{u \in \Sigma^* \mid u \notin L(N)\}$ – see Exercise 4.5.]

We saw on slide 79 that part (a) of Kleene's Theorem allows us to answer question (a) on Slide 38. Now that we have proved the other half of the theorem, we can say more about question (b) on that slide. In particular, it is a consequence of Kleene's Theorem plus the above lemma that for each regular expression r over an alphabet Σ, there is a regular expression $\sim r$ that determines via matching the complement of the language determined by r:

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

To see this, given a regular expression r, by part (a) of Kleene's Theorem there is a DFA M such that $L(r) = L(M)$. Then by part (b) of the theorem applied to the DFA $Not(M)$, we can find a regular expression $\sim r$ so that $L(\sim r) = L(Not(M))$. Since $L(Not(M)) = \{u \in \Sigma^* \mid u \notin L(M)\} = \{u \in \Sigma^* \mid u \notin L(r)\}$, this $\sim r$ is the regular expression we need for the complement of r.

$Not(M)$ Stopped Here

Given DFA $M = (Q, \Sigma, \delta, s, F)$,
then $Not(M)$ is the DFA with

- set of states $= Q$
- input alphabet $= \Sigma$
- next-state function $= \delta$
- start state $= s$
- accepting states $= \{q \in Q \mid q \notin F\}$.

(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 $Not(M)$ iff it is not accepted by M:

$$L(Not(M)) = \{u \in \Sigma^* \mid u \notin L(M)\}$$

Regular languages are closed under intersection

Theorem. If L_1 and L_2 are regular languages over an alphabet Σ, then their intersection $L_1 \cap L_2 = \{u \in \Sigma^* \mid u \in L_1 \,\&\, u \in L_2\}$ 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: $p \,\&\, q = \neg(\neg p \vee \neg q)$).

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

[It is not hard to directly construct a DFA $And(M_1, M_2)$ from M_1 and M_2 such that $L(And(M_1, M_2)) = L(M_1) \cap L(M_2)$ – 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 $r_1 \mathbin{\&} r_2$, such that

a string u matches $r_1 \mathbin{\&} r_2$ 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 $r_1 \mathbin{\&} r_2$ with $L(r_1 \mathbin{\&} r_2) = L(r_1) \cap L(r_2)$. □

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 Σ^*) of the form $L(r)$ for some r?

Equivalent regular expressions

Definition. Two regular expressions r and s are said to be **equivalent** if $L(r) = L(s)$, that is, they determine exactly the same sets of strings via matching.

For example, are $b^* a (b^* a)^*$ and $(a|b)^* a$ equivalent?

Answer: yes (Exercise 2.3)

How can we decide all such questions?

Note that $L(r) = L(s)$

iff $L(r) \subseteq L(s)$ and $L(s) \subseteq L(r)$
iff $(\Sigma^* \setminus L(r)) \cap L(s) = \emptyset = (\Sigma^* \setminus L(s)) \cap L(r)$
iff $L((\sim r) \,\&\, s) = \emptyset = L((\sim s) \,\&\, r)$
iff $L(M) = \emptyset = L(N)$

where M and N are DFAs accepting the sets of strings matched by the regular expressions $(\sim r) \,\&\, s$ and $(\sim s) \,\&\, r$ respectively.

So to decide equivalence for regular expressions it suffices to

check, given any given DFA M, whether or not it accepts some string.

Note that the number of transitions needed to reach an accepting state in a finite automaton is bounded by the number of states (we can remove loops from longer paths). So we only have to check finitely many strings to see whether or not $L(M)$ is empty.