MATH/CSCI 2113, DISCRETE STRUCTURES II, Winter 2010

Handout 7: Lecture Notes on Kleene's Theorem

1 Solving language equations

Let Σ be an alphabet, and recall that Σ^* is the set of words. A *language* is a subset of Σ^*, i.e., an element of $\mathscr{P}(\Sigma^*)$.

Theorem 1.1. *Let K and M be languages over an alphabet Σ, and consider the equation*

$$L = KL \bigm| M. \tag{1}$$

Then the smallest solution of (1) is the language

$$L' = K^*M.$$

Proof. First, we need to show that $L' = K^*M$ is a solution of (1). Indeed, using the laws of regular expressions, we have

$$L' = K^*M = (KK^* \mid \epsilon)M = KK^*M \mid \epsilon M = KL' \mid M,$$

and therefore L' is a solution. Next, we need to show that, if L is any other solution of (1), then $L' \subseteq L$. To prove this, consider an arbitrary element $w \in L'$. Then, by definition of K^*M, we have $w = k_n \dots k_1 m$, for some $n \geqslant 0$, $k_1, \dots, k_n \in K$, and $m \in M$. We prove that $w \in L$ by induction on n. For $n = 0$, we have $w = m \in M \subseteq KL \mid M = L$. For $n > 0$, we know that $w' = k_{n-1} \dots k_1 m \in L$ by induction hypothesis. Then $w = k_n w' \in KL \subseteq KL \mid M = L$, as desired. Since w was arbitrary, this shows that $L' \subseteq L$. Since L was an arbitrary solution of (1), this proves that L' is the least solution. □

Remark 1.2. If K and M are languages such that $\epsilon \notin K$, then the equation (1) has a *unique* solution, which is given by $L' = K^*M$.

Proof. We already know that $L' = K^*M$ is the least solution of (1). Let L be some other solution, and assume that $L' \neq L$. Since $L' \subseteq L$, this means that there exists some $w \in L - L'$. Let w be such a word of shortest length. We will derive a contradiction.

By assumption, $w \in L = KL \mid M$. It cannot be the case that $w \in M$, or else we would have $w \in L'$. Therefore, we must have $w \in KL$. It follows that $w = kl$, where $k \in K$ and $l \in L$. By assumption, $\epsilon \notin K$, therefore $k \neq \epsilon$. It follows that l is of shorter length than w. Since w was the shortest element of $L - L'$, it follows that $l \in L'$. But then $w = kl \in KL' = KK^*M \subseteq K^*M = L'$, which is the desired contradiction. □

2 Finite state automata

Definition. Let Σ be an alphabet. A *(deterministic) finite-state automaton* A over Σ is a labelled directed graph whose vertices are called *states* and whose edges are labelled by elements of Σ, together with

- a distinguished vertex s_0, called the *initial state*;
- a distinguished set of vertices T, called the *accepting states*;

such that the following condition holds:

- Determinism: for every vertex s and symbol $a \in \Sigma$, there exists exactly one edge labelled a with source s.

We write S for the set of states. The edges are also called *transitions*. The *next-state function* $N : S \times \Sigma \to S$ is defined so that $N(s, a)$ is the unique state s' for which there exists an edge $s \xrightarrow{a} s'$.

Given a finite-state automaton, the *eventual-state function* $N^* : S \times \Sigma^* \to S$ is defined recursively as:

$$\begin{array}{rcl} N^*(s, \epsilon) & = & s, \\ N^*(s, aw) & = & N^*(N(s, a), w). \end{array}$$

In other words, for a word $w = a_1a_2 \dots a_n \in \Sigma^*$, $N^*(s, w)$ is defined to be the unique state s' such that there exists a sequence of edges

$$s \xrightarrow{a_1} \xrightarrow{a_2} \cdots \xrightarrow{a_n} s'.$$

The *language accepted by A* (in the alphabet Σ) is defined as

$$L(A) = \{w \mid N^*(s_0, w) \in T\}.$$

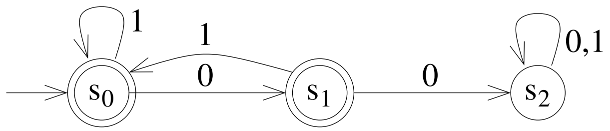
3 Translation from finite-state automata to regular expressions

Theorem 3.1 (Kleene's theorem, part 1). *Let L be the language accepted by some finite-state automaton A. Then L is defined by some regular expression.*

3.1 An example

Converting a finite-state automaton into a regular expression amounts to solving a system of equations. We will illustrate how this works in a few examples. It should then be clear that this can be done in general.

Consider the following finite-state automaton, which accepts all binary strings that do not contain repeated zeros:



Let $N^* : S \times \Sigma^* \to S$ be the eventual-state function. For each state s_i, let L_i be the language accepted by the state s_i, which is defined as:

$$L_i = \{w \mid N^*(s_i, w) \in T\}$$

Then from the description of the automaton, it is immediately clear that L_0, L_1, and L_2 satisfy the following equations:

$$\begin{aligned} L_0 &= 0L_1 \mid 1L_0 \mid \epsilon && (2)\\ L_1 &= 0L_2 \mid 1L_0 \mid \epsilon && (3)\\ L_2 &= 0L_2 \mid 1L_2. && (4) \end{aligned}$$

Note that these equations essentially tabulate the next-state function, and that we have added ϵ to the equation for L_i if and only if s_i is an accepting state.

Note that the equations are of the form of Remark 1.2, and we can solve them explicitly to obtain a regular expression for $L_0 = L(A)$.

We rewrite (4) as

$$L_2 = (0 \mid 1)L_2 \mid \emptyset,$$

and solve it:

$$L_2 = (0 \mid 1)^*\emptyset = \emptyset. \tag{5}$$

Substituting (5) into (3), we obtain

$$L_1 = 0\emptyset \mid 1L_0 \mid \epsilon = 1L_0 \mid \epsilon. \tag{6}$$

Substituting (6) into (2), we obtain

$$L_0 = 0(1L_0 \mid \epsilon) \mid 1L_0 \mid \epsilon,$$

which can be rewritten by the laws of regular expressions as

$$\begin{aligned} L_0 &= 01L_0 \mid 0\epsilon \mid 1L_0 \mid \epsilon\\ &= 01L_0 \mid 1L_0 \mid 0 \mid \epsilon\\ &= (01 \mid 1)L_0 \mid (0 \mid \epsilon). \end{aligned}$$

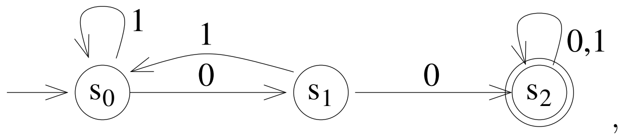
This has solution

$$L_0 = (01 \mid 1)^*(0 \mid \epsilon). \tag{7}$$

And indeed, this is the desired regular expression for the language of binary strings containing no repeated zeros.

3.2 Another example

Consider the automaton



,

which is the complement of the automaton of the previous example (i.e., it accepts exactly those binary strings that do contain a repeated zero). The system of equation then becomes

$$\begin{array}{rcl} L_0 & = & 0L_1 \mid 1L_0 \\ L_1 & = & 0L_2 \mid 1L_0 \\ L_2 & = & 0L_2 \mid 1L_2 \mid \epsilon. \end{array}$$

Notice that the only change is that we have added ϵ the last equation, instead of the first two. Solving the last equation for L_2, we get

$$L_2 = (0 \mid 1)^* \mid \epsilon = (0 \mid 1)^*.$$

Substituting this into the second equation, we get

$$L_1 = 0(0 \mid 1)^* \mid 1L_0.$$

Substituting this into the first equation, we get

$$\begin{array}{rcl} L_0 & = & 0(0(0 \mid 1)^* \mid 1L_0) \mid 1L_0 \\ & = & 00(0 \mid 1)^* \mid (01 \mid 1)L_0, \end{array}$$

which we solve as

$$L_0 = (01 \mid 1)^* 00(0 \mid 1)^*.$$

4 Non-deterministic finite state automata

A non-deterministic finite state automaton is defined similarly to a deterministic one, with the following exceptions:

- Edges are labelled by elements of $\Sigma \cup \{\epsilon\}$, where ϵ is a special symbol not contained in the alphabet Σ. An edge that is labelled by ϵ is called an *ϵ-transition* or an *ϵ-edge*.
- We drop the condition of determinism. Therefore, there could be more than one edge labelled a from a given state, or none.
- We allow a set of initial states, instead of just one.

More formally:

Definition. Let Σ be an alphabet and let ϵ be a symbol that is different from all elements of Σ. A *non-deterministic finite-state automaton* A over Σ is a labelled directed graph whose vertices are called *states* and whose edges are labelled by elements of $\Sigma \cup \{\epsilon\}$, together with

- a distinguished set of vertices I, called the *initial states*;
- a distinguished set of vertices T, called the *accepting states*.

As before, we write S for the set of states. We write $s \xrightarrow{a} s'$ if there exists an a-labelled edge from s to s'. We write $s \Rightarrow s'$ if s' can be reached from s by following zero or more ϵ-edges.

For a word $w = a_1 a_2 \ldots a_n \in \Sigma^*$, we write $s \overset{w}{\Rightarrow} s'$ if there exists a sequence of edges

$$s \Rightarrow \xrightarrow{a_1} \Rightarrow \xrightarrow{a_2} \Rightarrow \ldots \Rightarrow \xrightarrow{a_n} \Rightarrow s'.$$

We write $N^*(s, w) = \{s' \mid s \overset{w}{\Rightarrow} s'\}$. Note that this is a set of states, so the eventual-state function of a non-deterministic automaton is a function $N^* : S \times \Sigma^* \to \mathscr{P}S$.

A word $w \in \Sigma^*$ is *accepted* by A if there exists some initial state $s \in I$ and some accepting state $s' \in T$ such that $s \overset{w}{\Rightarrow} s'$. We define $L(A)$, the *language accepted by* A, to be the set of all $w \in \Sigma^*$ accepted by A.

5 Translation from non-deterministic finite-state automata to deterministic finite-state automata

If X is a set of states of a non-deterministic finite state automaton, we write $\bar{X} = \{s' \mid \exists s \in X. s \Rightarrow s'\}$. In other words, $\bar{X}$ is the set of all states reachable from X by zero or more ϵ-transitions. We say that X is ϵ-closed if $X = \bar{X}$.

Definition. Suppose we are given a non-deterministic finite state automaton A with state set S, initial states I, and accepting states T. We define a deterministic finite state automaton $\det(A)$ as follows:

- The states of $\det(A)$ are the ϵ-closed sets of states of A.
- The initial state of $\det(A)$ is $\bar{I}$.
- A state X is accepting if and only if $X \cap T \neq \emptyset$.
- For any $a \in \Sigma$, and any state X is $\det(A)$, there is an edge $X \xrightarrow{a} X'$ if and only if $X' = N^*(X, a)$. This means that X' is the set of all states of A that can be reached from a state in X by means of a single a-transition and zero or more ϵ-transitions.

Proposition 5.1. *The automata A and $\det(A)$ accept the same language. Moreover, $\det(A)$ is a deterministic finite state automaton.*

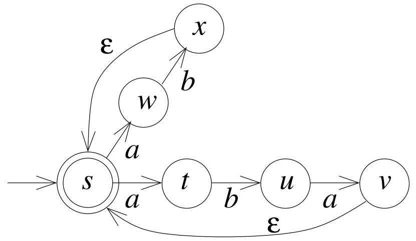
Corollary 5.2. *A language is accepted by some non-deterministic finite state automaton if and only if it is accepted by some deterministic finite state automaton.*

Proof. If L is accepted by some non-deterministic finite state automaton A, then it is also accepted by the deterministic finite state automaton $\det(A)$ by Proposition 5.1. Conversely, every deterministic finite state automaton can be regarded as a non-deterministic finite state automaton, which happens to have a single initial state and no ϵ-transitions. $\square$

5.1 An example

In theory, if A is a non-deterministic finite state automaton with n states, then $\det(A)$ has up to 2^n states. However, in practice, it suffices to enumerate the states of $\det(A)$ that can actually be reached from the initial state, and these are often much fewer than 2^n.

Consider the following non-deterministic finite state automaton A, which accepts the language $(ab|aba)^*$.



We can represent this automaton by its state transition table. At first, let's ignore the ϵ-transitions:

	a	b	
s	t, w	$\emptyset$	accepting, initial
t	$\emptyset$	u	
u	v	$\emptyset$	
v	$\emptyset$	$\emptyset$	
w	$\emptyset$	x	
x	$\emptyset$	$\emptyset$	

Next, we ϵ-close each entry in the table. For example, any state that can reach v can also reach s.

	a	b	
s	t, w	$\emptyset$	accepting, initial
t	$\emptyset$	u	
u	v, s	$\emptyset$	
v	$\emptyset$	$\emptyset$	
w	$\emptyset$	x, s	
x	$\emptyset$	$\emptyset$	

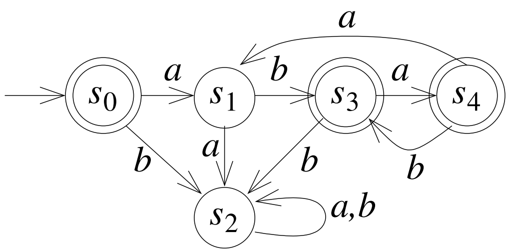
Now the states of $\det(A)$ are ϵ-closed sets of states of A, and the transitions of $\det(A)$ are calculated as unions of rows of the transition table of A. We start from the initial state s, and enumerate only states that occur in the columns for a or b in a previous row.

	a	b	
s	t, w	$\emptyset$	accepting, initial
t, w	$\emptyset$	u, x, s	
$\emptyset$	$\emptyset$	$\emptyset$	
u, x, s	v, s, t, w	$\emptyset$	accepting
v, s, t, w	t, w	u, x, s	accepting

The process ends after 5 states (of the $2^6 = 64$ possible) have been enumerated. Renaming these states $\{s\} = s_0$, $\{t, w\} = s_1$, $\emptyset = s_2$, $\{u, x, s\} = s_3$, $\{v, s, t, w\} = s_4$, we can rewrite the transition table of the deterministic FSA as follows:

	a	b	
s_0	s_1	s_2	accepting, initial
s_1	s_2	s_3	
s_2	s_2	s_2	
s_3	s_4	s_2	accepting
s_4	s_1	s_3	accepting

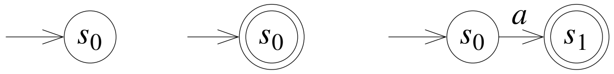
Here is a picture of the reachable states of $\det(A)$:



6 Translation from regular expressions to non-deterministic finite-state automata

We will translate each regular expression as a non-deterministic automaton.

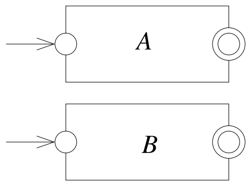
The base-case regular expressions $\emptyset$, ϵ, and a are easy to express as non-deterministic finite state automata. The are, respectively:



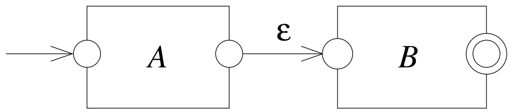
Given non-deterministic finite state automata A and B, we will define automata $A|B$, AB, and A^*, such that

$$L(A|B) = L(A) \cup L(B), \quad L(AB) = L(A)L(B), \quad L(A^*) = L(A)^*.$$

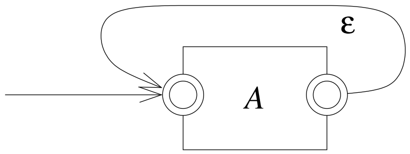
Definition (Union). The automaton $A|B$ is defined as the disjoint union of A and B, with their original transitions, initial states, and accepting states. In pictures:



Definition (Concatenation). The automaton AB is defined as follows: take the disjoint union A and B, with their original transitions. Keep the initial states of A initial, and keep the accepting states of B accepting. Add an ϵ-transition from each old accepting state of A to each old initial state of B. In pictures:



Definition (Iteration). The automaton A^* is defined as follows: take the same states, initial states, accepting states, and transitions as A, but add an ϵ-transition from each accepting state to each initial state, and make all initial states accepting. In pictures:



Lemma 6.1. *The following hold:*

$$L(A|B) = L(A) \cup L(B), \quad L(AB) = L(A)L(B), \quad L(A^*) = L(A)^*.$$

7 Kleene's theorem, part 2

Theorem 7.1 (Kleene's theorem, part 2). *Let L be the language defined by some regular expression. Then L is accepted by some deterministic finite state automaton.*

Proof. First, by induction on the size of the regular expression, and using the constructions of Section 6, we can construct a non-deterministic finite state automaton A that accepts the language L. Second, by Proposition 5.1, $\det(A)$ is a deterministic finite state automaton that accepts L. $\square$

Remark. The number of states of the non-deterministic automaton A is proportional to the size of the regular expression. The number of states of the deterministic automaton $\det(A)$ is exponentially larger in the worst case. However, in practice, the size of the deterministic automaton can be reduced in two ways: first, by removing non-reachable states (as discussed in Section 5.1), and second, by identifying $*$-equivalent states (as discussed in Chapter 12.3 of the textbook).