Today's lecture: Regular Languages

I. Finite Automata and Regular Languages
II. Closure Properties

Schedule:

Today: Regular Languages.
The rest of *Sipser* 1.1 (i.e. pages 40–47).

September 15: Non-Determinism – Read: *Sipser* 1.2.
Lecture will cover through Example 1.35 (i.e. pages 47–52).
Homework 1 goes out (due Sept. 25).

September 18: NFAs
Lecture will cover through Example 1.38 (i.e. pages 53-54).
Homework 0 due. item**September 20:** Equivalence of DFAs and NFAs.
The rest of *Sipser* 1.2 (i.e. pages 54–63).

September 22: Regular Expressions – Read: *Sipser* 1.3.
Lecture will cover through Example 1.58 (i.e. pages 63-59). Homework 2 goes out (due Oct. 2).

September 25: Equivalence of DFAa and Regular Expressions.
The rest of *Sipser* 1.3 (i.e. pages 63–76).
Homework 1 due.

September 27 and beyond: see Sept. 6 notes.

I. Finite Automata and Regular Languages

A. An example automaton

1. The diagram for this example, see Figure 1.

2. Processing the string `abcaabc`

previously processed input	current input symbol	pending input	current state	next state
ϵ	a	bcaabc	0	1
a	b	caabc	1	0
ab	c	aabc	0	0
abc	a	abc	0	1
abca	a	bc	1	1
abcaa	b	c	1	0
abcaab	c	ϵ	0	0
abcaabc	–	ϵ	0	

The string is accepted. ☺

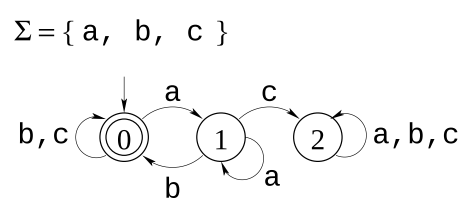


Figure 1: A Finite Automaton

3. Processing the string `bacab`

previously processed input	current input symbol	pending input	current state	next state
ϵ	b	acab	0	0
b	a	cab	0	1
ba	c	ab	1	2
bac	a	b	2	2
baca	b	ϵ	2	2
bacab	–	ϵ	2	

The string is rejected. ☹

B. Formally Defining Finite Automata

1. The ingredients of a finite automaton

a. A set of states, Q.

b. An input alphabet, Σ, a set of symbols.

c. A transition function: $\delta : Q \times \Sigma \to Q$.

d. An initial state, q_0.

e. A set of accepting states: $F \subseteq Q$.

2. We can combine these to make a formal, mathematical description of a finite automaton

a. The combination is a "tuple"

i.. that means lump them all together

ii.. the order of the elements in the tuple matters.

Thus, $(Q, \Sigma, \delta, q_0, F)$ is a finite automaton, but $(q_0, \delta, Q, F, Sigma)$ is not.

b. The tuple is: $(Q, \Sigma, \delta, q_0, F)$.

3. We say that this is a formal, mathematical definition because everything in the definition has a well-defined mathematical meaning: sets, functions, sequences, and tuples.

4. Our example machine as a tuple:

- $M = (Q, \Sigma, \delta, q_0, F)$, where
- $Q = \{0, 1, 2\}$
- $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$
- I'll define $\delta(q, c)$ with the table below:

	$\delta(q,c)$	a	b	c
		c		
q	0	1	0	0
q	1	1	0	2
q	2	2	2	2

- $q_0 = 0$
- $F = \{0\}$

C. How can we decide if the formally defined machine accepts as string?

1. Generalize δ to work on strings

$$\begin{array}{rcl} \delta(q,\epsilon) & = & q \\ \delta(q, w \cdot c) & = & \delta(\delta(q,w),c) \end{array}$$

Note how our definition of δ (for strings) parallels the inductive definition of strings from the Sept. 08 lecture:

A string of elements of Σ is either

$$\begin{array}{rll} & \epsilon, & \text{the empty string} \\ \text{or} & w \cdot c, & \text{where } c \in \Sigma \text{, and } w \text{ is a string.} \end{array}$$

2. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton, and let $w \in \Sigma^*$ be a string. M accepts w iff $\delta(q_0, w) \in F$.

3. Our examples again

a. Processing the string `abcaabc`.

$$\begin{array}{lll}
\multicolumn{2}{l}{\delta(q_0, \texttt{abcaabc})} & \\
= & \delta(0, \texttt{abcaabc}), & q_0 = 0 \\
= & \delta(\delta(0, \texttt{abcaab}), \texttt{c}) & \\
= & \delta(\delta(\delta(0, \texttt{abcaa}), \texttt{b}), \texttt{c}) & \\
= & \delta(\delta(\delta(\delta(0, \texttt{abca}), \texttt{a}), \texttt{b}), \texttt{c}) & \\
= & \delta(\delta(\delta(\delta(\delta(0, \texttt{abc}), \texttt{a}), \texttt{a}), \texttt{b}), \texttt{c}) & \\
= & \delta(\delta(\delta(\delta(\delta(\delta(0, \texttt{ab}), \texttt{c}), \texttt{a}), \texttt{a}), \texttt{b}), \texttt{c}) & \\
= & \delta(\delta(\delta(\delta(\delta(\delta(\delta(0, \texttt{a}), \texttt{b}), \texttt{c}), \texttt{a}), \texttt{a}), \texttt{b}), \texttt{c}) & \\
= & \delta(\delta(\delta(\delta(\delta(\delta(\delta(\delta(0, \epsilon), \texttt{a}), \texttt{b}), \texttt{c}), \texttt{a}), \texttt{a}), \texttt{b}), \texttt{c}), & \text{now we can simplify!} \\
= & \delta(\delta(\delta(\delta(\delta(\delta(\delta(0, \texttt{a}), \texttt{b}), \texttt{c}), \texttt{a}), \texttt{a}), \texttt{b}), \texttt{c}) & \\
= & \delta(\delta(\delta(\delta(\delta(\delta(1, \texttt{b}), \texttt{c}), \texttt{a}), \texttt{a}), \texttt{b}), \texttt{c}) & \\
= & \delta(\delta(\delta(\delta(\delta(0, \texttt{c}), \texttt{a}), \texttt{a}), \texttt{b}), \texttt{c}) & \\
= & \delta(\delta(\delta(\delta(0, \texttt{a}), \texttt{a}), \texttt{b}), \texttt{c}) & \\
= & \delta(\delta(\delta(1, \texttt{a}), \texttt{b}), \texttt{c}) & \\
= & \delta(\delta(1, \texttt{b}), \texttt{c}) & \\
= & \delta(0, \texttt{c}) & \\
= & 0 &
\end{array}$$

Thus, $\delta(q_0, \texttt{abcaabc}) \in F$, and the string is accepted.

b. Processing the string `bacab`.

Left as an exercise for the industrious (or bored) reader.

D. Definition of a regular language

1. A regular language is a language that is accepted by a finite automaton.

a. The automaton must accept every string that is in the language, and

b. the automaton must reject every string that is not in the language.

c. We have given precise, mathematical definitions for a finite automaton (i.e. a 5-tuple, ...) and what it means for a finite automaton to accept a string. Thus, we have formally defined the regular langauges.

2. Keep in mind that languages are sets of strings:

a. There is a finite automaton that accepts every string. It's language is Σ^*. This machine **does not** accept accept all languages. That's because to accept language L, the machine must reject all strings that *are not* in L.

b. Likewise, there is a finite automaton that rejects eery string. It's language is $\emptyset$. This machine **does not** reject all languages.

c. Every finite automaton recognizes exactly one language.

II. Closure Properties

A. What is a closure property?

1. Formal definition

a. Let A be a set, and $\circ$ be an operation on elements of A.

b. We say that A is closed under $\circ$ iff for all $x, y \in A$, it is the case that $x \circ y$ is defined, and $x \circ y$ is an element of A.

2. Examples

a. The natural numbers are closed under addition and multiplication.

b. The natural numbers are not closed under subtraction.

c. The integers are closed under subtraction.

d. The non-zero rational numbers are closed under division.

e. The non-zero rational numbers are not closed under square root.

3. Why we care.

a. Let's say we have some z that we want to prove to be an element of some set A. If we can find $x, y \in A$ such that $z = x \circ y$, and A is closed under $\circ$, then we've shown that $z \in A$. I'll give an example shortly.

b. Conversely, we might have some z that we know is *not* an element of A, some x that we know is an element of A, and some y for which we are unsure. If we can show that $x \circ y = z$, then we have shown that $y \notin A$. Here's an example:

- Show that $(\sqrt{2} + \sqrt{3})$ is not rational.
- Proof:

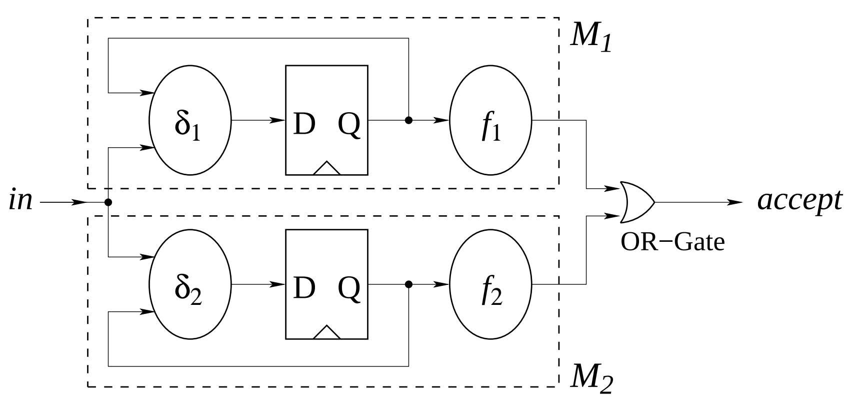
1. Let $\mathbb{Q}$ denote the rational numbers.
2. $\sqrt{2} \notin \mathbb{Q}$
   Sipser Thm. 0.24, p. 22.
3. Given an natural number, n, either $\sqrt{n}$ is either a natural number or it is irrational.
   A generalization of Thm. 0.24 from *Sipser*.

4.		$((\sqrt{2} + \sqrt{3}) \in \mathbb{Q})$	
	$\Leftrightarrow$	$((\sqrt{2} + \sqrt{3})^2 \in \mathbb{Q})$,	rationals closed under $*$
5.	$\Leftrightarrow$	$(2 + 2\sqrt{2}\sqrt{3} + 3) \in \mathbb{Q})$,	rewrite $((\sqrt{2} + \sqrt{3})^2$
6.	$\Leftrightarrow$	$2\sqrt{2}\sqrt{3} \in \mathbb{Q})$,	rationals closed under $+$
7.	$\Leftrightarrow$	$\sqrt{2}\sqrt{3} \in \mathbb{Q})$,	rationals closed under $*$
8.	$\Leftrightarrow$	$\sqrt{6} \in \mathbb{Q})$,	rewrite $\sqrt{2}\sqrt{3}$
9.	$\Leftrightarrow$	False,	$\sqrt{6} \notin \mathbb{Q}$, see step 3

B. Closure properties of the regular languages:

1. The regular languages are closed under union

a. Proof:



b. Another proof:

We can construct the 5-tuple for the finite automaton corresponding to the circuit above.

Let L_1 and L_2 be regular languages. Because L_1 is regular, it is recognized by some finite automaton. Let $M_1 = (Q_1, \Sigma, \delta_1, q_{0,1}, F_1)$ be an automaton that recognizes L_1. Likewise, let $M_2 = (Q_2, \Sigma, \delta_2, q_{0,2}, F_2)$. Note that both machines must have the same alphabet, because δ_1 and δ_2 must be defined over the same alphabet (and I have no idea why Sipser claimed that the alphabets could be different (p. 46, step 2 of his construction – he may have had a proof with NFAs in mind, but we haven't seen NFAs yet). I'll now construct a machine $M = (Q, \Sigma, \delta, q_0, F)$ such that $L(M) = L_1 \cup L_2$. Because M is a finite automaton, $L(M)$ is regular, therefore $L_1 \cup L_2$ is regular.

Q: Each state of M consists of a state from M_1 and a state from M_2. We represent such combinations with a Cartesian product: $Q = Q_1 \times Q_2$. Note that the number of states of M is the product of the number of states of M_1 and the number of states of M_2. This allows M to be in any state-pair from M_1 and M_2.

Σ: The same alphabet as for M_1 and M_2.

δ: M makes moves that correspond to both M_1 and M_2 moving in parallel. Thus, the Q_1 component of M's state changes according to δ_1, and the Q_2 component changes according to δ_2. We can write this as the formula:

$$\delta((q_1, q_2), c) \quad = \quad (\delta_1(q_1, c), \delta_2(q_2, c))$$

q_0: M starts in the initial state for each machine:

$$q_0 \quad = \quad (q_{0,1}, q_{0,2})$$

F: M accepts if either M_1 or M_2 accepts:

$$F \quad = \quad \{(q_1, q_2) \in Q \mid (q_1 \in F_1) \vee (q_2 \in F_2)\}$$

Sipser states that the correctness of this construction is obvious. That's good enough for me. If you want a more formal proof, we can prove by induction that for any string $w \in \Sigma^*$:

$$\delta(q_0, w) \quad = \quad (\delta_1(q_{0,1}, w), \delta_2(q_{0,2}, 2))$$

Here we go:

Case $w = \epsilon$:

$$\begin{array}{rcll} \delta(q_0, w) & = & \delta(q_0, \epsilon), & w = \epsilon \\ & = & q_0, & \text{def. } \delta(-, \epsilon) \\ & = & (q_{0,1}, q_{0,2}), & \text{def. } q_0 \\ & = & (\delta_1(q_{0,1}, \epsilon), \delta_2(q_{0,2}, \epsilon)), & \text{def. } \delta_i(-, \epsilon) \end{array}$$

Case $w = x \cdot c$:

$$\begin{array}{rcll} \delta(q_0, w) & = & \delta(q_0, x \cdot c), & w = x \cdot c \\ & = & \delta(\delta(q_0, x), c), & \text{def. } \delta(-, x \cdot c) \\ & = & \delta(\delta((q_{0,1}, q_{0,2}), x), c), & \text{def. } q_0 \\ & = & \delta((\delta_1(q_{0,1}, x), \delta_2(q_{0,2}, x)), c), & \text{induction hypothesis} \\ & = & (\delta_1(\delta_1(q_{0,1}, x), c), \delta_2(\delta_2(q_{0,2}, x), c)), & \text{def. } \delta \\ & = & (\delta_1(q_{0,1}, x \cdot c), \delta_2(q_{0,2}, x \cdot c)), & \text{def. } \delta_i(-, x \cdot c) \\ & = & (\delta_1(q_{0,1}, w), \delta_2(q_{0,2}, w)), & w = x \cdot c \end{array}$$

Now we can show that $L(M) = L_1 \cup L_2$.

$L(M) \subseteq L_1 \cup L_2$: Let $w \in L(M)$. That means $\delta(q_0, w) \in F$. Using the result of our induction proof above, we get $(\delta_1(q_{0,1}, w), \delta_2(q_{0.2}, w)) \in F$. From the definition of F, we conclude that either $\delta_1(q_{0,1}, w) \in F_1$, or $\delta_2(q_{0,2}, w) \in F_2$. In the first case, $w \in L(M_1) = L_1$, and in the second case, $w \in L(M_2) = L_2$. Either way, $w \in L_1 \cup L_2$ as required.

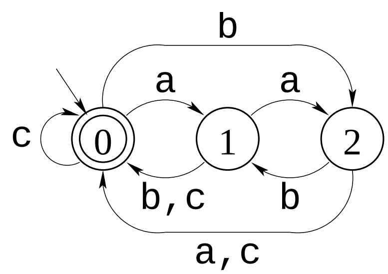


Figure 2: M_2: another finite automaton

$L_1 \cup L_2 \subseteq L(M)$: Let $w \in L_1 \cup L_2$. Assume that $w \in L_1$, the other case is equivalent. This means that $w \in L(M_1)$ which means that $\delta_1(q_{0,1}, w) \in F_1$. From the induction proof above, we have $\delta(q_0, w) = (\delta_1(q_{0,1}, w), \delta_2(q_{0,2}, w))$. We just showed that $\delta_1(q_{0,1}, w) \in F_1$; therefore, $(\delta_1(q_{0,1}, w), \delta_2(q_{0,2}, w)) \in F$. This means that M accepts w. In other words, $w \in L(M)$ as required.

c. An example:

Let L_1 be the language recognized by the finite automaton shown in Figure 1. Likewise, let M_2 be the finite automaton shown in Figure 2, and let $L_2 = L(M_2)$. Languages L_1 and L_2 are regular because they are recognized by finite automata. Thus, we know that $L_1 \cup L_2$ is regular because the regular languages are closed under union. We don't have to figure out how to draw an automaton for $L_1 \cup L_2$, we know that it exists. Thus, this saves us a bunch of work.

2. The regular languages are closed under concatenation

a. What is concatenation?

Let L_1 and L_2 be two languages. We write $L_1 \circ L_2$ for the *concatenation* of languages L_1 and L_2. A string, w is in $L_1 \circ L_2$ iff there are strings x and y (possibly empty) such that $w = xy$, $x \in L_1$, and $y \in L_2$.

b. Showing that the regular languages are closed under concatenation.

This is a bit more involved than the proof for union. Basically, we have to construct a machine that finds a prefix of w that is in L_1 such that the rest of the string is in L_2. The problem is that there may be more than one such choice. For example, let L_1 and L_2 be defined as in the example for union. Let $w =$ ababcbabbabbabba. If we let $x =$ ababcb and $y =$ abbabbabba, we can show that $x \in L_1$ and $y \in L_2$. On the other hand, if we try $x =$ ab and $y =$ abcbabbabbabba we'll find that $x \in L_1$ but $y \notin L_2$. You can find other ways to break up w that work and others that don't work.

At first it might seem that a machine to recognize $L_1 \circ L_2$ must keep track of all possible places to break w. For an aribitrarily long w, this means keeping track of an arbitrary amount of information, which isn't longer finite state.

The solution is to make a machine whose state keeps track of what state M_1 would be in if we are still reading x and all the possible states that M_2 could be in if we've switched to reading y. Because M_2 has a finite set of states, there are only a finite set of possible combinations. Of course, if M_2 has n_2 states, then there are 2^{n_2} possible combinations, but that is still finite. So, we could build a the following machine:

$Q = Q_1 \times 2^{Q_2}$.

Σ the same alphabet as for M_1 and M_2.

$\delta((q_1 P_2), c) = (\delta_1(q_1, c), \{p' \| (\exists p \in P_2.\ p' = \delta_2(p, c)) \vee ((p' = q_{0,2}) \wedge (\delta_1(q_1, c) \in F_1))\})$.

That's a big formula. The $\delta_1(q_1, c)$ part keeps track of the state of M_1. if M_1 reaches an accepting state (i.e. $(\delta_1(q_1, c) \in F_1)$), then we include $q_{0,2}$ in the set of states of Q_2 that we are tracking. For each state of Q_2 that we are tracking, we include its successor according to δ_2 (thats what the $p' = \delta_2(p, c)$ stuff is about.

$q_0 = (q_{0,1}, \{q_{0,2} \mid \text{ if } q_{0,1} \in F_1\})$.

That means that we start M_1 in its initial state. If the initial state of M_1 is an accepting state, we start M_2 right away. Otherwise, we set the initial possible states of M_2 to the empty set.

$F = Q_1 \times 2_2^F$.

We accept if there's anyway that M_2 could be in an accepting state.

We could go on and prove this construction correct, but I won't bother. The idea of keeping track of the states that a finite automaton could be in is the central idea behind NFAs (non-deterministic finite automata). We'll start on NFAs on Friday. Once we've introduced NFAs, showing closure under concatention is straightforward.

3. The regular languages are closed under Kleene star. If L is a language, then $w \in L^*$ iff there is some $k \geq 0$ and strings $x_1, x_2, \ldots x_k$ such that $w = x_1 \cdot x_2 \cdots x_k$, and all of the x_i's are in L. Note that L^* always contains the empty string.