Illinois Institute of Technology
Department of Computer Science

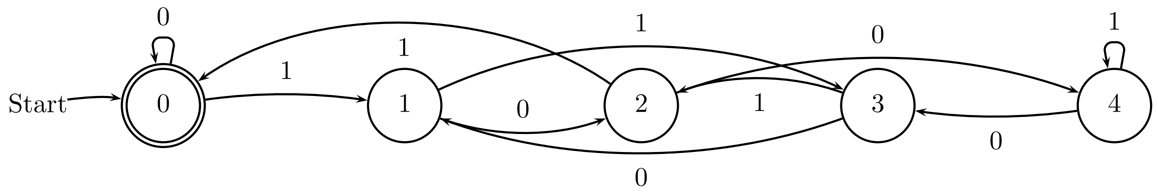
Lecture 23: April 17, 2019

CS 330 Discrete Structures
Spring Semester, 2019

1 Finite state machines (continued)

1.1 Finite state machines as a computational model

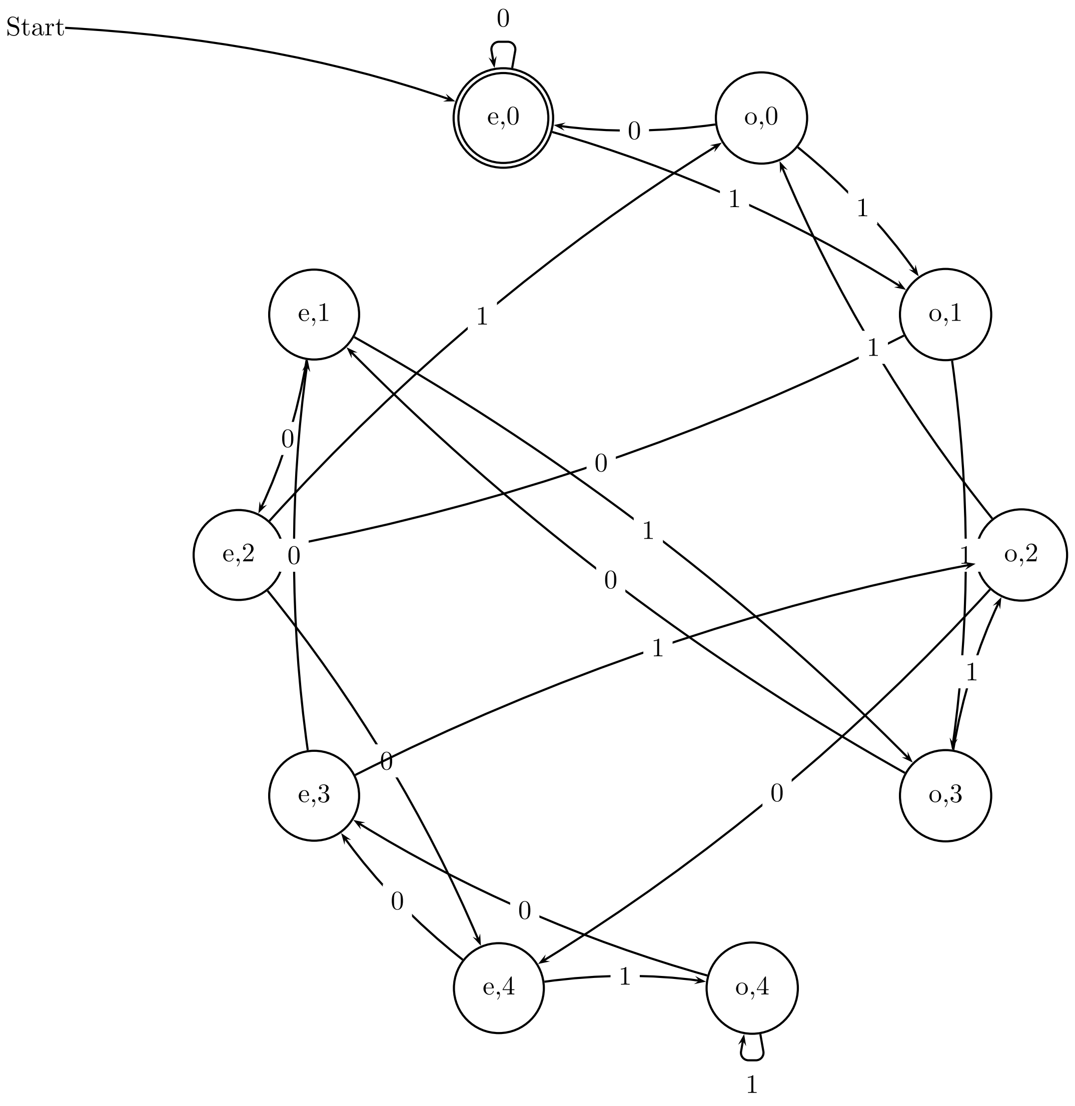
Consider a multiple of five. If a 0 is appended to the end, the number is doubled and is still a multiple of five. If a 1 is appended to the end, the number is doubled and 1 is added, so we should move to state 1. Similar reasoning for the other four states gives us:



What about multiples of ten? We can follow a similar approach and find the transitions for ten states labeled 0–9. Alternatively, since the multiples of ten are the numbers that are both even and multiples of five, and we have machines that recognize even numbers and multiples of five, it would be nice if we could somehow combine these two machines. In essence, we want to run these two machines in parallel, applying the input to both and accepting the input if both machines accept.

We can collapse these two machines into one by keeping track of the state of each machine in our new machine. This gives us the (somewhat messier) machine:[1]

[1]In fact, in this case, this is the identical machine, with the states relabeled, that we would derive if we used the same approach we did for the multiples of five on this problem. There is a simple way to reduce this machine to one with only six states: do you see how to do it?



2 Closure properties of regular languages

We can prove that a language L is regular by constructing a finite state machine that recognizes L. We use this observation to prove a number of results.

Lemma: The set of regular languages is closed under intersection.[2]

Proof:[3] Let L_1 and L_2 be regular languages. Say that L_1 is recognized by $M_1 = (\Sigma, S, s_0, F, \delta)$ and L_2 is recognized by $M_2 = (\Sigma, T, t_0, G, \lambda)$. Then $L_1 \cap L_2$ is recognized by $M = (\Sigma, S \times T, (s_0, t_0), F \times G, \alpha)$, where $\alpha : \Sigma \times S \times T \to S \times T$ is defined as $\alpha(x, s, t) = (\delta(x, s), \lambda(x, t))$.

[2]That is, the intersection of two regular languages is still a regular language.

[3]This is precisely what we did to solve the problem of multiples of ten.

Lemma: The set of regular languages is closed under union.

Proof: Let L_1 and L_2 be regular languages. Say that L_1 is recognized by $M_1 = (\Sigma, S, s_0, F, \delta)$ and L_2 is recognized by $M_2 = (\Sigma, T, t_0, G, \lambda)$. Then $L_1 \cup L_2$ is recognized by $M = (\Sigma, S \times T, (s_0, t_0), (F \times T) \cup (S \times G), \alpha)$, where $\alpha : \Sigma \times S \times T \to S \times T$ is defined as $\alpha(x, s, t) = (\delta(x, s), \lambda(x, t))$.

Lemma: The set of regular languages is closed under complementation.

Proof: Let L be a regular language. Say that L is recognized by $M = (\Sigma, S, s_0, F, \delta)$. Then $\overline{L}$ is recognized by $M' = (\Sigma, S, s_0, S - F, \delta)$.

These three lemmas directly lead to the following closure theorem:

Theorem: The set of regular languages is closed under intersection, union, and complement.

We move to other properties of regular languages.

Theorem: All finite sets are regular.

Proof sketch: Consider the language $L = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$. We prove this by induction on k.

Base case: Here $k = 1$. Assume σ_1 is not empty;[4] let $\sigma_1 = a_1 a_2 \cdots a_n$. We can construct a finite state machine that recognizes σ_1. Σ is the appropriate alphabet. Let $S = \{0, 1, 2, \ldots, n, \text{FAIL}\}$, $s_0 = 0$, and $F = \{n\}$.

Define the transition function δ as follows:

$$\begin{aligned} \delta(\text{FAIL}, a_j) &= \text{FAIL} \\ \delta(i, a_j) &= \begin{cases} j & \text{if } i + 1 = j \\ \text{FAIL} & \text{otherwise} \end{cases} \end{aligned}$$

Then $M = (\Sigma, S, s_0, F, \delta)$ recognizes σ_1 and no other strings.

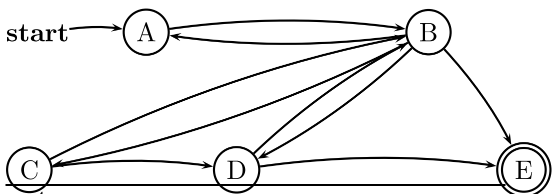
Inductive step: For $k > 1$, build a machine M_k that recognizes σ_k. By induction there is a machine $M_{1\ldots k-1}$ that recognizes the set $\{\sigma_1, \sigma_2, \ldots, \sigma_{k-1}\}$. By the "union lemma" above, the union of $\{\sigma_k\}$ and $\{\sigma_1, \sigma_2, \ldots, \sigma_{k-1}\}$ is therefore regular.

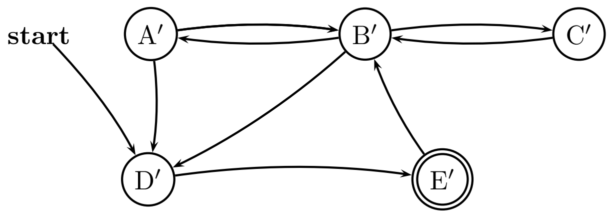
We now consider some slightly more complicated questions of regularity. Is concatenation closed over regular languages? The concatenation of languages L_1 and L_2 is defined as

$$L_3 = L_1 L_2 = \{l_1 l_2 \mid l_1 \in L_1,\ l_2 \in L_2\}$$

For example, if $L_1 =\{$ all strings with at least two zeroes$\}$ and $L_2 =$ all strings with at least two ones then $L_3 = L_1 L_2 = \{$all strings that can be divided into a first section with at least two zeroes and a last section with at least two ones$\}$.

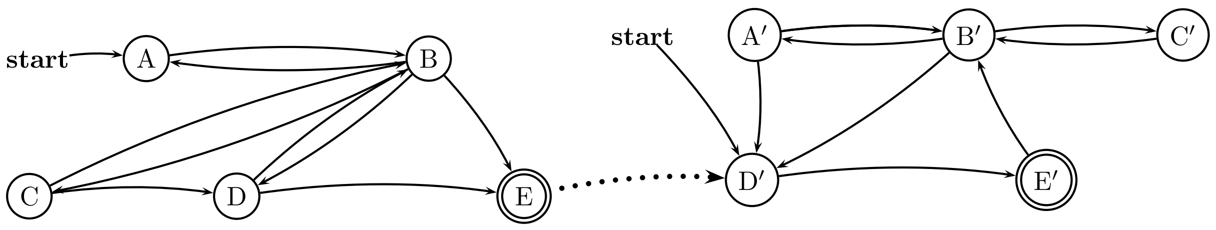
So far we have connected machines in "parallel", meaning that we simulate running two machines at the same time, For concatenation we want to hook up machines in "series". Thus, suppose we have 2 arbitrary machines M_1 and M_2 accepting L_1 and L_2:





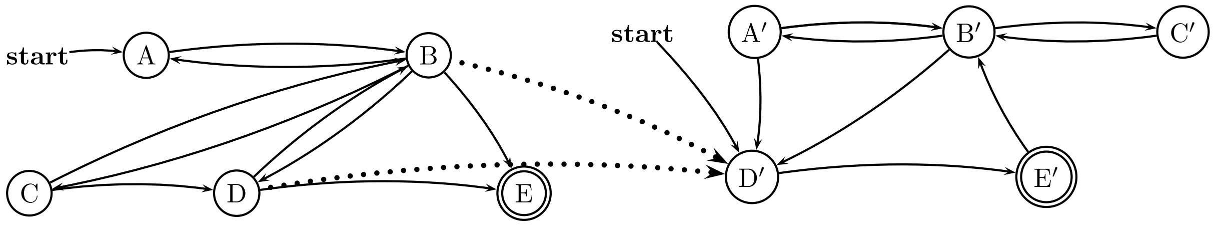
[4]Exercise: Design a finite state machine that recognizes only the empty string.

One attempt at making a machine M_3 to accept L_3 involves linking the final states of M_1 to the start state of M_2:



However, we have some problems here. What is the boundary between the two machines? If we are given a string in L_3, how do we decide which part of it should be checked for membership in L_1 and which part should be checked for membership in L_2. Moreover, the transition from machine M_1 to M_2 does not use up an input character, which violates the definition of a finite state machine.

We can fix the second problem by connecting any transitions into an accepting states of M_1 directly to the start state of the second machine as follows:



The first problem with our construction, however, is more complicated. In some sense, we want to break up a string in all possible ways $x = uv$, checking if $u \in L_1$ and $v \in L_2$. The basis for such a technique is nondeterminism.

Now we have

Theorem: If L_1 and L_2 are regular, then L_1L_2 is regular.

We define L^* to be $\{\varepsilon\} \cup L \cup LL \cup LLL \cup \cdots$. With the above theorem and the fact that regular language is closed under union operation, it would wrong to conclude that if L is regular, then L^* is regular because that is an infinite union and the union closure property holds only for finite unions. However L^* is regular if L is; the next section shows us how to approach the problem.

3 Non-determinism

When you come to a fork in the road, take it.

—Yogi Berra

A non-deterministic finite state machine transitions from one state to a whole set of states. In other words, it tries out several possibilities at the same time, and accepts if any of the possibilities lead to an accepting state.

More precisely, a non-deterministic finite state machine is a quintuple $(\Sigma, S, s_0, F, \delta)$ where:

Σ is the alphabet of the machine

S is the finite set of states of the machine

s_0 is the start state

F is the set of final states of the machine

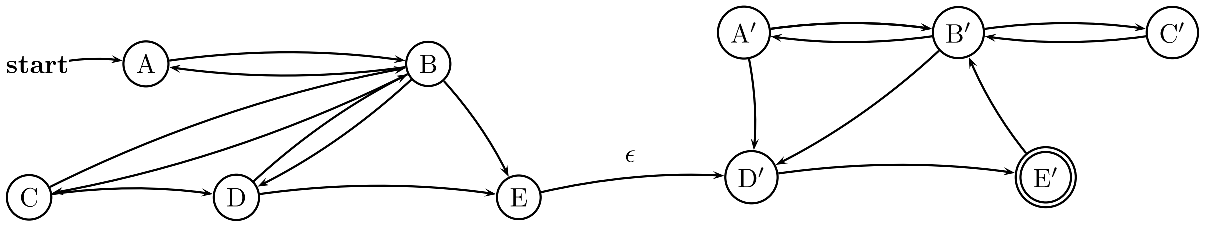
δ is a mapping from $S \times \Sigma \longrightarrow 2^S$. Where 2^S is the set of all subsets of S; it is sometimes called the power set of S and denoted $P(S)$. For example;

$$2^{\{1,2,3\}} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{2,3\}, \{1,3\}, \{1,2,3\}\}$$

Thus, this transition function maps states and input characters to a set of states.

A non-deterministic finite state machine accepts if any of the states in which it ends up at the end of the input is an accepting state.

In a non-deterministic machine, we may also allow transitions from state to state without gobbling up input; these transitions are called epsilon transitions (ϵ is the null string). Neither nondeterminism nor epsilon transitions really increase the power of our finite state machines, because they can all be modeled with a regular deterministic finite state machine. Nevertheless, they do make life easier. For example, we may now represent the concatenation of two finite state machines as the following nondeterministic machine:



Using nondeterminism, we can also show that for any given a regular language L, the L^{reverse} is also regular. L^{reverse} is simple the language consisting of the reverses of all the strings in L. For example, the reverse of the string "I love CS330" is "033SC evol I".

To prove closure under reversing, we start with the machine M that accepts L. Then we reverse the direction of all the transitions, and switch starting states with final states. If M has more than one final state, then we add a new starting state to the new machine, and add epsilon transitions from this new state to each of the final states of M.

Note that reversing the directions of transitions might mean that we have two transitions on the same symbol coming out of a state, so that the new machine needs to be nondeterministic.

Now, as an exercise, show that L^* is regular when L is by connecting final states to the start state with an epsilon move.

3.1 An example of non-regular language

Consider the language L=$\{0^n1^n | n = 0, 1, 2, \cdots\}$. This is not a regular language. If it was, we should have an finite state automaton which accepts L. Suppose this automaton has N states, then if we input a string 0^i1^i of length $2i, i > N$, by the pigeon hole principle, when we move around and reach the first "1", we move i times and must hit a certain state at least twice. So there will be a loop starting and ending at this state. Cutting the substring which labels the loop or repeating this substring for any times will give us another string which is also accepted by the automaton. But the string can't satify that we have the same number of "0"s and "1"s!

You can find a more detailed proof in Rosen. The same analysis gives us

pumping lemma: Let L be a regular set. Then there is a constant n such that if z is any word in L, and $|z| \geq n$, we may write z = uvw in such a way that $|uv| \leq n$, $|v| \geq 1$, and for all $i \geq 0$, uv^iw is in L.

Pumping lemma is a very useful tool to prove that a language is *not* regular.