Today's lecture: Non-Determinism

I. Closure Properties, continued **II.** Two-Tape Finite Automata **III.** Nondeterministic, Finite Automata (NFAs)

Schedule:

Today: 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.

$\Sigma = \{$ a, b, c $\}$

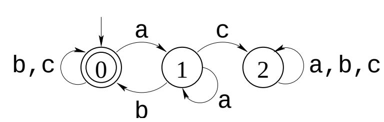

Figure 1: M_1: a finite automaton

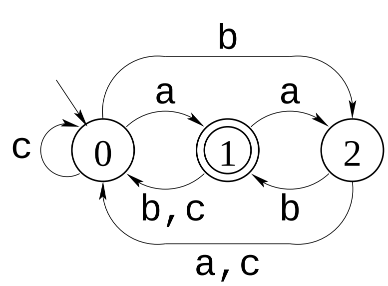

Figure 2: M_2: another finite automaton

I. Closure properties of the regular languages:

A. Machines and languages for our examples (see Figures **??** and **??**.

1. We showed in the Sept. 13 lecture that M_1 accepts all strings where every `a` is followed eventually by a `b` without an intervening `c`.

2. What language does M_2 recognize?

3. Let $L_1 = L(M_1)$ and $L_2 = L(M_2)$.

B. The regular languages are closed under unionfig:xxx.

1. We proved this in the Sept. 13 lecture and notes.

2. Consider the strings `a`, `ab`, `bb`, and `ac`:

$$\begin{array}{llll} \texttt{a} \notin L_1, & \texttt{ab} \in L_1, & \texttt{bb} \in L_1, & \texttt{ac} \notin L_1, \\ \texttt{a} \in L_2, & \texttt{ab} \notin L_2, & \texttt{bb} \in L_2, & \texttt{ac} \notin L_2. \end{array}$$

We conclude that `a`, `ab` and `bb` are in $L_1 \cup L_2$, but that `cc` is not. (There are an infinite number of strings in $L_1 \cup L_2$ and an infinite number of strings that are not in $L_1 \cup L_2$).

C. The regular languages are closed under concatenation

1. 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 $y \in L_2$.

For example, `abb` $\in L_1$ and `a` $\in L_2$. Therefore `abba` $\in L_1 \circ L_2$ even though `abba` is in neither L_1 nor L_2.

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

This is a bit more involved than the proof for union. The hard part is figuring out where to break the string. Given a string, w, it may have many prefixes that are in L_1, but only some of them may have corresponding suffixes in L_2.

For example, if we break `abba` into `ab` and `ba`, we see that `ab` $\in L_1$, but `ba` $\notin L_2$. How do we find the "right" place to break the string?

We will see later this lecture that this is an example of where we can use *nondeterminism*. We'll try to sneak up on nondeterminism gently by looking at "two-tape" machines first.

D. The regular languages are closed under Kleene star.

1. What is "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.

2. An example:

a. The strings `a`, `aba` and `acbb` are in L_2.

b. The following strings are in L_2^*: ϵ, `a`, `aaba`, `acbbacbbacbbaba`.

c. Show that any string that ends with two or more consecutive `a`'s is in L_2^*.

II. Two-Tape Machines

A. Continuing with Concatenation

1. Let's modify M_1 and M_2 to work with the alphabet $\{\texttt{a}, \texttt{b}, \texttt{c}\} \times \{\texttt{0}, \texttt{1}\}$.

2. Figure 3 shows how we combine these machine using this extended alphabet.

a. The second component of each symbol says whether it should be treated as part of the string for M_1 or part of the string for M_2.

b. Figure 3 shows a finite automaton that recognizes the concatenation language modified by using this extended alphabet. Therefore, this version of concatenation produces a regular language.

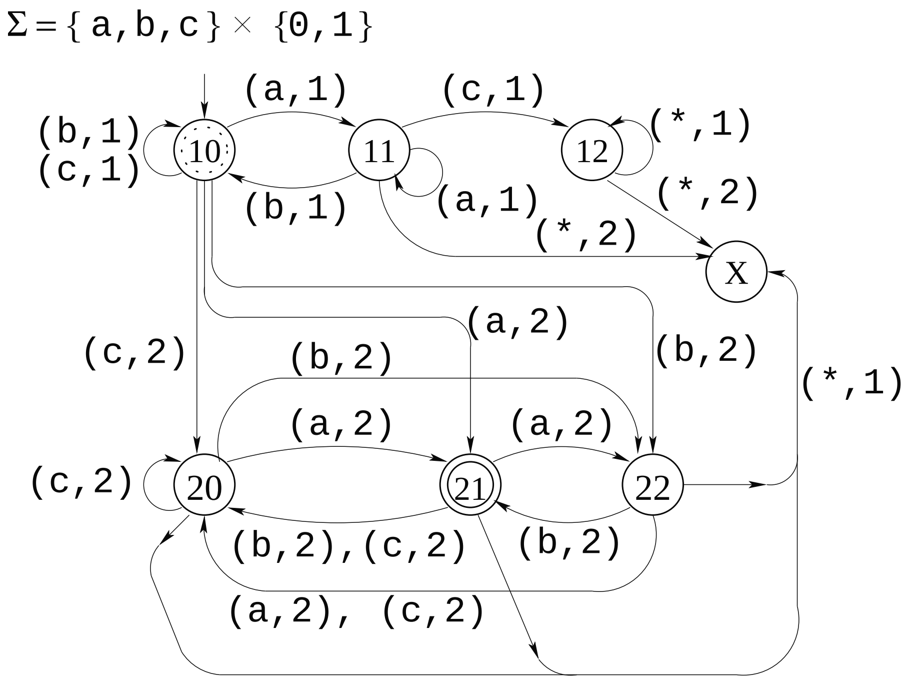

Figure 3: M_1 and M_2 "concatenated" by using an extended alphabet

B. Two-tape machines

1. Let's split the two parts of the input into separate strings.

a. Defining the *weave* function:

$$\begin{aligned} \mathit{weave}(\epsilon, \epsilon) &= \epsilon \\ \mathit{weave}(x_1 \cdot c_1, x_2 \cdot c_2) &= \mathit{weave}(x_1, x_2) \cdot (c_1, c_2) \end{aligned}$$

2. A two-tape finite automaton is a 6-tuple: $M = (Q, \Sigma_1, \Sigma_2, \delta, q_0, F)$ where

Q is a finite set of states.

Σ_1 and Σ_2 are finite sets of symbols.

$\delta : Q \times (\Sigma_1 \times \Sigma_2) \rightarrow Q$ is the state transition function.

$q_0 \in Q$ is the initial state.

$F \subseteq Q$ is the set of accepting states.

We say that M accepts (w_1, w_2) with $w_1 \in \Sigma_1^*$ and $w_2 \in \Sigma_2^*$ iff $\mathit{length}(w_1) = \mathit{length}(w_2)$ and the machine $(Q, \Sigma_1 \times \Sigma_2, \delta, q_0, F)$ accepts $\mathit{weave}(w_1, w_2)$.

3. So far, our two-tape machines are just a slight variation on ordinary finite automata.

4. Hiding the second string.

a. Think about our concatenation example. A string w is in $L_1 \circ L_2$ iff there is some string v such that $\mathit{weave}(w, v)$ is accepted by the machine from figure 3.

b. We can generalize this idea. Let $M = (Q, \Sigma_1, \Sigma_2, \delta, q_0, F)$ be a two-tape finite automaton. We'll say that M *existentially accepts* $w \in \Sigma_1^*$ iff there exists some string, $v \in \Sigma_2^*$ such that M accepts $\mathit{weave}(w, v)$.

- We're not saying how we find the v string. We could imagine writing a program that tries all possible strings of the right length. We might be able to come up with more efficient schemes. However, it doesn't matter. We've got a perfectly precise definition, even if we're not sure how we could best build the hardware or write the software to implement it.

5. Simplifying our two-tape machines.

a. If the user doesn't have to provide the second input string, why should we ask them to include it in their description. We can just have multiple arcs out of the same state with the same label. We can think of these arcs as needing "advice" as to which one to take. Figure 4 shows our example machine with this simplification.

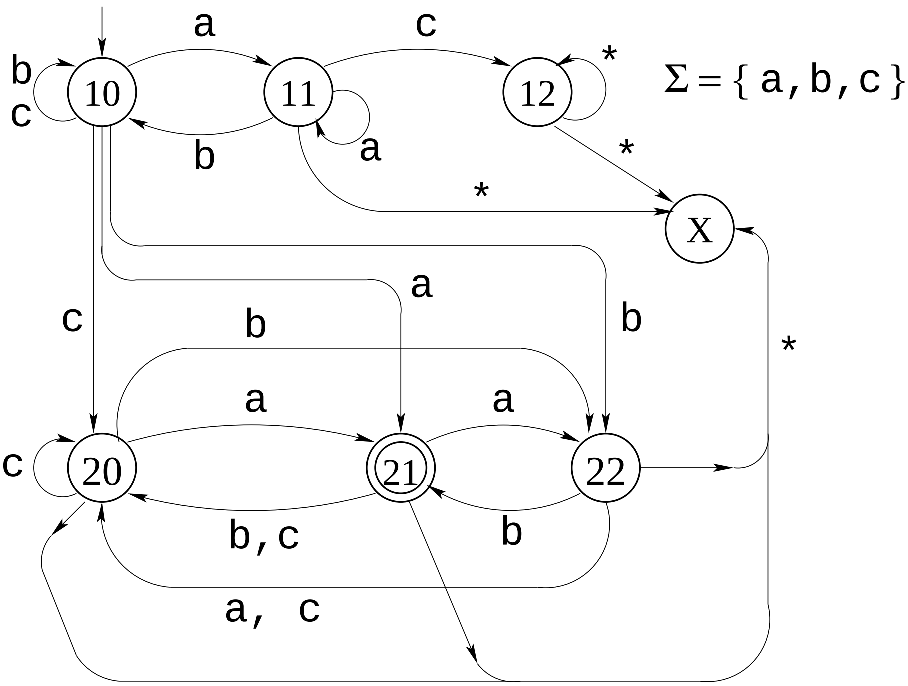

Figure 4: The machine from Figure 3 with the second tape"hidden"

b. If we allow multiple arcs out of a state for the same symbol, can we allow a state to have no arcs out for some symbol?

i.. Where should they go?

ii.. The default destination should make sense for any machine. This suggests that they go to a permanently accepting state or a permanently rejecting one. Noting all of the clutter in Figures 3 and 4 for arcs to permanently rejecting states, we'll make these the default. *If a state has no outgoing arc for symbol c, then if the machine reads a c from that state, the machine rejects its input.*

iii.. Figure 5 shows our example machine with this simplification.

c. Now, we notice that state 10 has three arcs that go to states from M_2: states 20, 21 and 22.

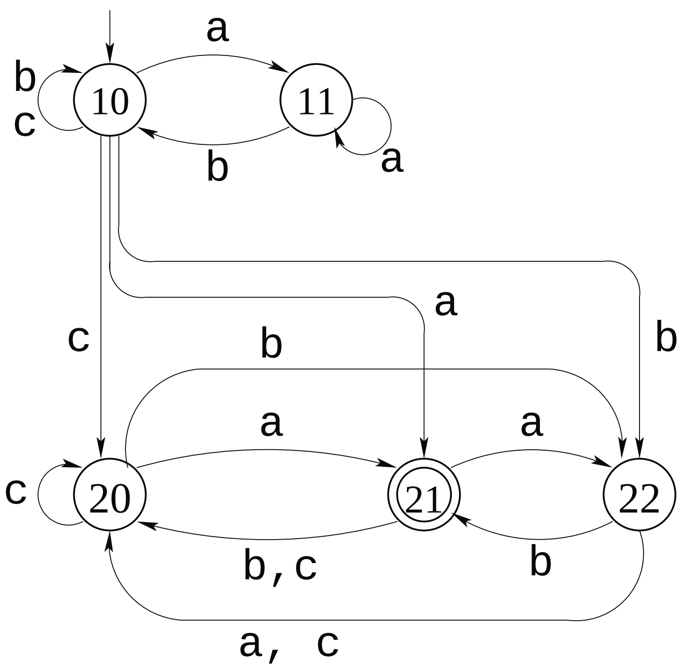

Figure 5: The machine from Figure 4 with arcs to permanently rejecting states omitted