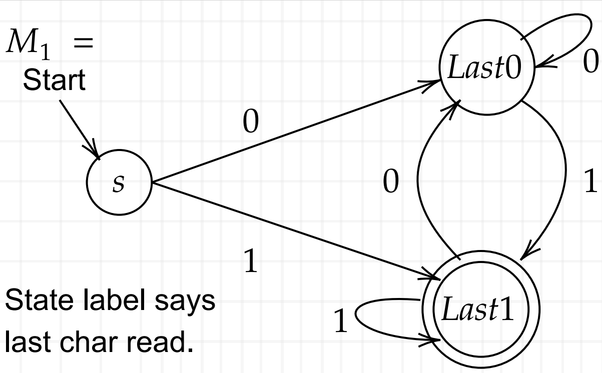
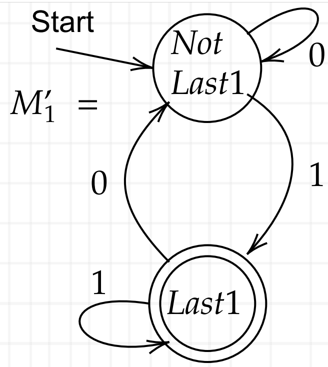
CSE396 Lecture Tue. 2/9/21: Finite Automata and Languages

Suppose we want to accept only those binary strings x that end in 1. We have $\Sigma = \{0,1\}$.
Is that the same as saying x does not end in 0? No: the empty string ϵ does not end in 0 but that doesn't mean it ends in 1.

Designing a finite automaton is sometimes like playing "Musical Chairs". Any char that we read might be the end of the string. If the char is a 1, we have to be at the accepting "chair". So we make two states, one saying the previous char read was a 1, the other a 0. We will also tentatively make the start state separate, saying no char has been read yet.



In fact, the start state and Last0 are equivalent in that both are not in F and both go to the same state on the same char. Fusing them gives:



By popular demand, the table for M_1: $M_1 = (Q, \Sigma, \delta, s, F)$ where $Q = \{s, Last0, Last1\}$, $\Sigma = \{0,1\}$, the start state is literally called s, $F = \{Last1\}$, and $\delta: Q \times \Sigma \to Q$ is defined by

$\delta(s,0) = Last0, \delta(s,1) = Last1, \delta(Last0,0) = Last0,\ \delta(Last0,1) = Last1,\ \delta(Last1,0) = Last0$, and $\delta(Last1,1) = Last1$.

Or in my own preferred style as a set of instructions,
$\delta = \{(s,0,Last0),(s,1,Last1),(Last0,0,Last0),(Last0,1,Last1),(Last1,0,Last0),(Last1,1,Last1)\}$

But on homeworks, it is much more important to give a **well-commented arc-node diagram** than to give the tables like the text does (without comments!). One thing that also helps is to re-state the target language in various ways. So how else can we express "strings that end in 1"?

$$L_1 = \left\{w1 : w \in \{0,1\}^*\right\}.$$

What does "$\{0,1\}^*$" mean? The superscript star * means "zero or more". Zero or more of what? Chars. What does "zero chars" mean? It means the empty string ϵ. So what this says is that w can be any binary string whatever, which makes $w1$ stand for any binary string that ends in a 1.

We could also just write directly $L_1 = \{0,1\}^* \cdot 1$. The comma is then read "or". But more often in programming, especially scripting, we write a vertical bar (or two) to mean "or": $L_1 = (0|1)^*1$. Well, the text writes $\cup$, which corresponds to "OR" the way $\cap$ is a way of expressing AND logic in sets. So the text would write $L_1 = (0 \cup 1)^* \cdot 1$. That looks fine when typed, but in handwriting the $\cup$ tends to close up and look like 0, while $|$ always looks like 1. So I will use a third style one can find online and write + for "or", so $L_1 = (0+1)^*1$. (But a superscript $^+$ instead of * will mean "one or more.") Once the choice and understanding are settled, all of these are visually clear: it must end in 1 and what comes beforehand can be anything.

A Second Language

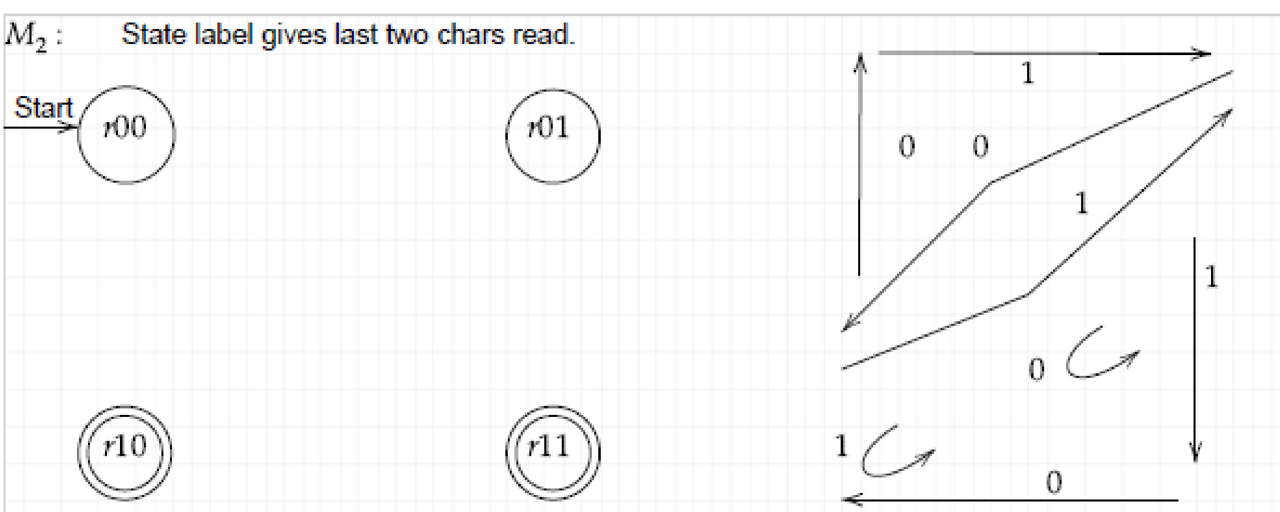
Now, how about $L_2 = \{x :$ the second from last char in x is a $1\}$? How can we express this more compactly and visually? We could say $\{0,1\}^*10$. But that leaves out strings that end in 11, which are good too. Now, by the way, the string "1" is no longer good: it needs at least 2 chars. So we can write (using the text's $\cup$ style):

$L_2 = \{0,1\}^*10 \cup \{0,1\}^*11$. Or we can group it as

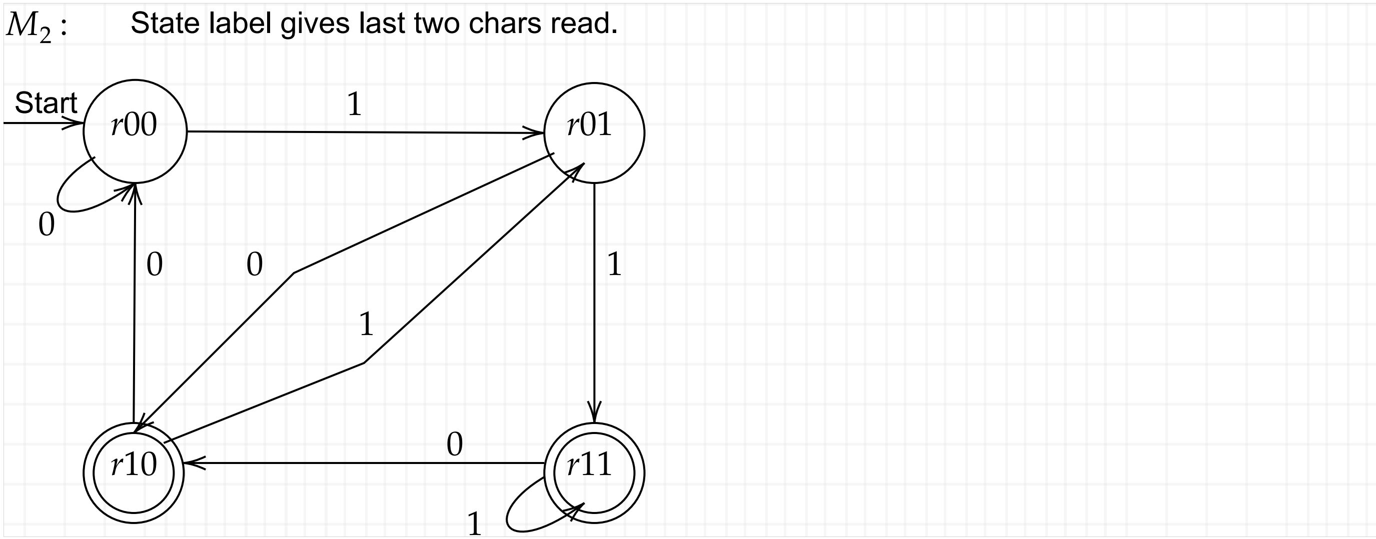
$L_2 = \{0,1\}^*(10 \cup 11)$.

We can even group it as $\{0,1\}^*1(0 \cup 1)$ but maybe that is "too cute". If we don't want to mix braces and parens, we can write it as $L_2 = (0 \cup 1)^* \cdot (10 \cup 11)$. "My way": $(0+1)^*(10+11)$.

How about a DFA now? Can we do it with a 2-state machine, since after all the language is conceptually almost as simple as L_1 is? Ummm...no. We have to track the last 2 chars read. We can say something up-front about the starting condition: If the last two chars read were both 0, they give us no help toward a 1 (if the "music stops" now or after the next char, we lose). Hence, that is really the same condition as starting from scratch. Starting with a 0 gives no help, while starting with 1 is just like the last two chars being 01. Thus we can make "Last00" the start state and proceed accordingly. Let's abbreviate that to $r00$ where r means "read" and label the other states $r01$, $r10$, and $r11$. The latter two are our accepting states. Once we lay down the states and the starting and final conditions, the arcs should be easy to fill in:



In lecture I did so:



Moral: The left-hand side is well-commented enough that it would be *full credit*. Whereas, I've seen people write down a table like the following without even saying what the states in F are:

State \ char	0	1
1	1	2
2	3	4
3	1	2
4	3	4

Just from that, I have no idea what is going on.

Third From Last Char

Now how about $L_3 = \left\{x \in \{0,1\}^* : \textit{the third char from the right end is a } 1\right\}$? Among many ways to write this more symbolically but visually we can give:

$$L_3 = (0 \cup 1)^*(100 \cup 101 \cup 110 \cup 111) \text{, which equals } (0 \cup 1)^* 1 (0 \cup 1)^2 .$$

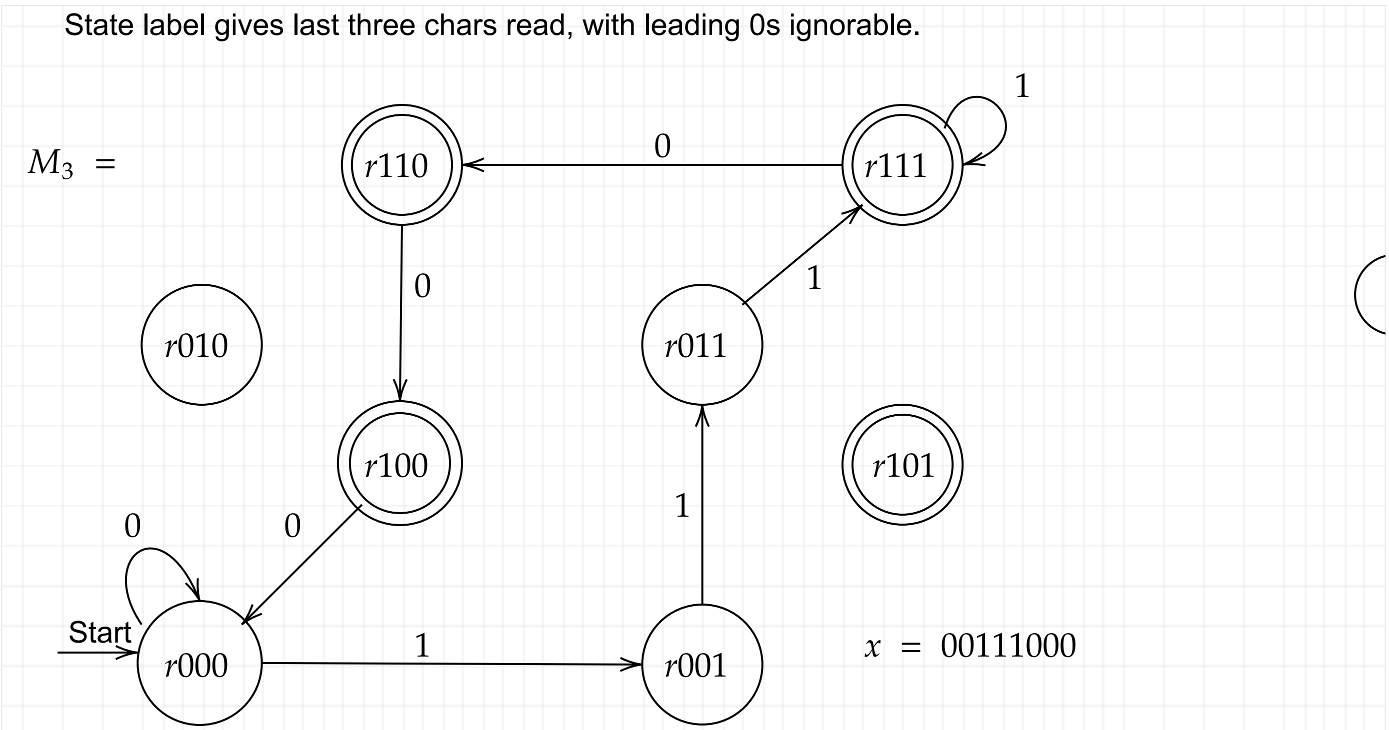
The superscript ^{2} doesn't mean squaring. It means *exactly two* occurrences of 0 or 1. If I wrote it as $L_3 = (0+1)^*1(0+1)^2$, the $+$ and ^{2} still wouldn't be numerical. There is, however, a symbolic resemblance to the numerical operations. For one, we can imitate how $(0+1)^2$ multiplies out:

$$(0+1)^2 = 0 \cdot 0 + 0 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 = 00 + 01 + 10 + 11 .$$

So long as you realize that the concatenation $\cdot$ is not commutative, and that $0 \cdot 1$ doesn't equal zero, you can use analogies with rules of ordinary algebra. Chief among them is the distributive law. That's what allows one to write, e.g.,

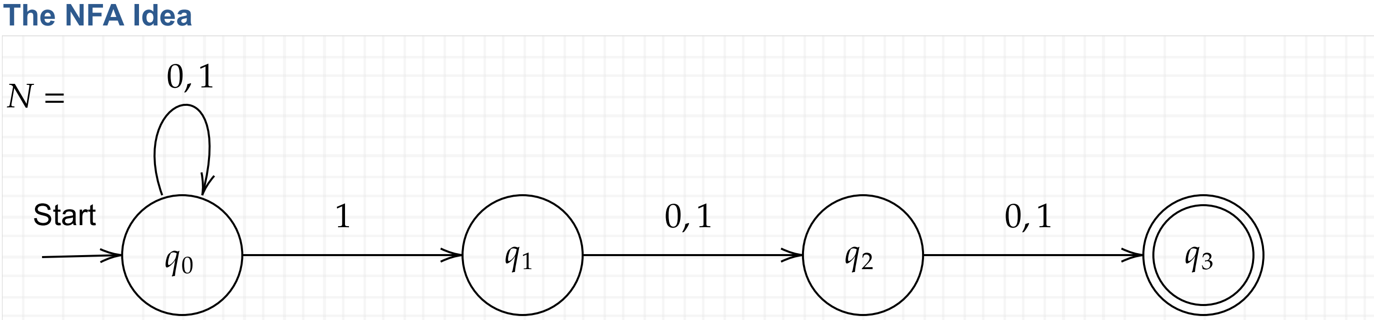
$$(1 \cdot 00 + 1 \cdot 01 + 1 \cdot 10 + 1 \cdot 11) \;=\; 1 \cdot (00 + 01 + 10 + 11) \;=\; 1(0+1)^2 \quad .$$

Now for the machine. Alas, we will prove in a few weeks that it cannot be built with fewer than 8 states---that one really needs to track the $2^3 \;=\; 8$ possibilities for the last 3 chars read. So:



The arcs filled in may make you think this is going to be a nice cube, but after these it gets pretty messy. The fact that this is not a nice cube also hints that this is not really a Cartesian product situation. It is also somehow lacking the clean visual impact of the expression $(0 \cup 1)^* 1 (0 \cup 1)^2$, or in my terms, $(0+1)^* 1 (0+1)^2$. Is there a kind of machine to reflect this?

The NFA Idea



Note that if you are in state q_3 and the music doesn't stop---that is, you get another char---then you can't go anywhere. The computation "crashes" and you lose---even though q_3 is an accepting state. The major story is what goes down at the start state if you get a 1. You have the option to stay at start or make a "leap of faith" by going to q_1: banking on there being exactly 2 more chars. This is *nondeterminism* **at** state q_0 **on** char 1. We have $N = (Q, \Sigma, \delta, s, F)$ where:

- $Q = \{q_0, q_1, q_2, q_3\}$
- $s = q_0$,
- $F = \{q_3\}$, and
- $\delta = \{(q_0, 0, q_0), (q_0, 1, q_0), (q_0, 1, q_1), (q_1, 0, q_2), (q_1, 1, q_2), (q_2, 0, q_3), (q_2, 1, q_3)\}$.

The two highlighted tuples have the same source state and char but different destination states. Thus the δ relation does **not** define a function from $Q \times \Sigma$ to Q. For this reason, we cannot unambiguously execute the machine like we could before. But as a specification, it makes visual sense of what the language is---maybe more sense than the crazy twisty half-finished cube M_3.

[If time allows, define NFAs formally, but otherwise, this sets up the reading of sections 1.2 and 1.3 for Thursday.]