We saw that DFAs $M$, nor even NFAs nor GNFAs, cannot recognize simple languages like $\{a^m b^n : m = n\}$. How can we augment the DFA model to give it the needed capability?

1. Allow $M$ to change a character it reads, storing it on its tape.
2. Allow $M$ to move its scanner left L as well as right R (or keep it stationary S).

Capability 1 by itself changes nothing: the DFA would still have to move R past the changed character. Capability 2 by itself also does not allow recognizing any nonregular languages. The proof, that every "two-way DFA" can be simulated by a simple 1-way DFA, is beyond our scope and involves another "exponential explosion" but we will cite it later to say that the class of regular languages equals "constant space" on a Turing machine.

But if we give both capabilities together, then we can do it—and lots more besides. The capabilities add two components to instructions in $\delta$, making them 5-tuples:

$$(p, c / d, D, q) \text{ where } p \text{ and } q \text{ are states, } c \text{ and } d \text{ are chars, and } D \in \{L, R, S\}$$

The meaning is that if $M$ is in state $p$ and scans character $c$, then it can change it to $d$, move its scanning head one position left, right, or keep it stationary, and finally transit to state $q$. The case $(p, c, c, R, q)$ is the same as an ordinary FA instruction $(p, c, q)$ where moving right is automatic. I tend to like to write a slash for the second comma to emphasize that $p, c$ are read and $d, D, q$ are actions taken; it also visually suggests $c$ being changed to $d$. Graphically the instruction looks like:

```
\begin{center}
\begin{tikzpicture}
\node [state] (p) {$p$};
\node [state] (q) [right of=p] {$q$};
\path[->] (p) edge node[above] {$(c / d, L)$} (q);
\end{tikzpicture}
\end{center}
```

or

```
\begin{center}
\begin{tikzpicture}
\node [state] (p) {$p$};
\node [state] (q) [right of=p] {$q$};
\path[->] (p) edge node[above] {$(c / d, D)$} (q);
\path[->,loop above] (q) edge node[above] {} (q);
\end{tikzpicture}
\end{center}
```

for a self-loop.

We also regard the blank as an explicit character. I will represent it as _ in MathCha but in full LaTeX you can get "\text{\textvisiblespace}" which turns up the corners to look like more than just an underscore. My other notes call the blank $B$. The blank belongs not to the input alphabet $\Sigma$ but to the work alphabet $\Gamma$ (capital Gamma) which always includes $\Sigma$ too. We allow going past the right end of the input string $x \in \Sigma^*$ where successive tape cells each initially hold the blank. We can also allow moving leftward of the first char of $x$ where there are likewise blanks on a "two-way infinite tape", or we can stipulate that $x$ is initially left-justified on a "one-way infinite tape" and consider any left move from the first cell to be a "crash." The Turing Kit package shows a two-way infinite tape and this is the default. A compromise is to use a one-way infinite tape but place a special left-endmarker char $\wedge$ in cell 0 with $x$ occupying cells 1, \ldots, $n$ where $n = |x|$. If $x = c$ then the whole tape is initially blank except in the last case it has just $\wedge$ in cell 0. Then $\wedge$, as well as _, belongs to $\Gamma$ but not to $\Sigma$. We will be free to put any other characters we want into $\Gamma$, but the blank (and $\wedge$ if used) are required. With all that said, the definition is crisp:
**Definition:** A Turing machine is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, \_, s, F)$ where $Q, s, F$ and $\Sigma$ are as with a DFA, the work alphabet $\Gamma$ includes $\Sigma$ and the blank $\_$, and

$$\delta \subseteq (Q \times \Gamma) \times (\Gamma \times \{L, R, S\} \times Q).$$

It is deterministic (a DTM) if no two instructions share the same first two components. A DTM is "in normal form" if $F$ consists of one state $q_{\text{acc}}$ and there is only one other state $q_{\text{rej}}$ in which it can halt, so that $\delta$ is a function from $(Q \setminus \{q_{\text{acc}}, q_{\text{rej}}\}) \times \Gamma$ to $(\Gamma \times \{L, R, S\} \times Q)$. The notation then becomes

$$M = (Q, \Sigma, \Gamma, \delta, \_, s, q_{\text{acc}}, q_{\text{rej}}).$$

To define the language $L(M)$ formally, especially when $M$ is properly nondeterministic (an NTM), requires defining configurations (also called IDs for instantaneous descriptions) and computations, but especially with DTMs we can use the informal understanding that $L(M)$ is the set of input strings that cause $M$ to end up in $q_{\text{acc}}$, while seeing some examples first.

**Multi-Tape Turing Machines**

**Definition:** A $k$-tape Turing machine is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, \_, s, F)$ where $Q, s, F$ and $\Sigma$ are as with a DFA, the work alphabet $\Gamma$ includes $\Sigma$ and the blank $\_$, and

$$\delta \subseteq (Q \times \Gamma^k) \times (\Gamma^k \times \{L, R, S\}^k \times Q).$$

It is deterministic (a DTM) if no two instructions share the same first two components. A DTM is "in normal form" if $F$ consists of one state $q_{\text{acc}}$ and there is only one other state $q_{\text{rej}}$ in which it can halt, so that $\delta$ is a function from $(Q \setminus \{q_{\text{acc}}, q_{\text{rej}}\}) \times \Gamma^k$ to $(\Gamma^k \times \{L, R, S\}^k \times Q)$. The notation then becomes

$$M = (Q, \Sigma, \Gamma, \delta, \_, s, q_{\text{acc}}, q_{\text{rej}}).$$

An individual instruction can be notated as:

$$(p, [c_1, c_2, \ldots, c_k]/[d_1, \ldots, d_k], [D_1, \ldots, D_k], q) \quad \text{where} \quad p \text{ and } q \text{ are states, } c_j \text{ and } d_j \text{ are chars, and } D_j \in \{L, R, S\}, \quad j = 1 \text{ to } k$$

**Single Tape Vs. Multiple-Tape TMs---An Example**

$$_{a^m b^n} : \quad n = m.$$  

$x = bbb$ has $m = 0$ but $n = 3 \neq m$ so reject.

By default, $n, m$ are natural numbers, so $n = m = 0$ is allowed, and so $\varepsilon \in L$. Recall that when the input $x$ is $\varepsilon$, the TM tape starts off completely blank. Otherwise, the TM starts in the configuration of scanning the first char of $x$, with the rest of the tape blank. So an initial scan of $\_$ means that $x = \varepsilon$
and we can make $M$ accept right away. And if $x$ starts with $b$ then it cannot be in $L$, so we can make $M$ reject right away. A Turing machine is not required to scan its entire input, though we can impose this requirement (and when we discuss time complexity classes, we will). This gives us a good beginning on how to build $M$ to recognize $L$ step-by-step with goal-oriented reasoning. [Lecture worked on the diagram "interactively"; here we show some stages.]

We've already been able to handle immediate accept and reject conditions in the start state. Now we decide strategy when $x$ begins with $a$. The idea is to $X$-out $a$'s and $b$'s one-by-one in alternation. If we $X$-out always the leftmost $a$ and the rightmost $b$ then the string between (which after the first iteration is $a^{m-1}b^{n-1}$) will belong to $L$ if and only if $x$ does. So we can recurse and keep:

**Tape Invariant:** $X^* a^* b^* X^*$ and after $X$-ing a $b$ the numbers of $X$es on left and right are the same, so the string between them belongs to $L$ if and only if the original $x$ does.

To perform the $X$-ing of one $a$ then the rightmost $b$, add these states and instructions:

Note $\Gamma = \{a, b, \_ X\}$ so we need 4 arcs at each non-halting state. We added an arc on $X$ at the "go right" state because on subsequent iterations the rightmost $b$ will be next to an $X$ not a blank. But what if there is no such $b$? Since we just $X$-ed an $a$, this means there were initially more $a$'s than $b$'s, so we should reject.

Now after $X$-ing the matching $b$ is when we need to talk about what is successful termination. If there is an $X$ to its left then there are no more $a$'s nor $b$'s, so we paired them all, thus an $X$ should mean goto $q_{acc}$. Getting an $a$ once again means not enough $b$'s. On $b$ is when we want to "rewind" to the left end. That is
Note that the input $x$ can belong to $a^* b^*$ without belonging to $L$. Those strings abide by the tape invariant initially, and we can already see that $M$ works correctly on those strings. But what if $x$ is something like $aababb$? Will our $M$ accept when it shouldn't? That's what the footnote is about.

[This is the question where my Wed. 9/27/23 lecture left off. I will pick up here.]