Robustness

We say a computational model is *robust* if small changes to the model don’t change the class of languages recognized.

Robustness is a desirable property of a computational model, and gives us some assurance that our model is a “reasonable” and natural one.

Are DFA’s robust? Yes, because adding extra features like nondeterminism and $\epsilon$-transitions doesn’t change the class of languages recognized: it’s still REG in all cases.
Robustness

Are PDA’s robust?

To some extent.

When we changed the model to allow transitions on multiple letters and multiple stack symbols, we still got the same class of languages.

However, changing whether the model is deterministic or not deterministic actually does change the class of languages accepted. So PDA’s are not as robust a computational model as DFA’s.

TM’s definitely are robust. As we’ll see, pretty much any change to the model still gives the same class of recognized languages, RE.
Variations on the Turing machine model

We can change our basic Turing machine model as presented so far in various ways:

– we can *weaken* the model, by only allowing a tape that extends arbitrarily far to the right (it has a left end)

– we can *strengthen* the model, by adding options such as
  ▶ the ability to remain on the same cell in a move
  ▶ having multiple tracks on the tape (but only one read/write head)
  ▶ having multiple tapes, each with its own independently movable read/write head
  ▶ allowing nondeterminism
  ▶ giving the TM a sun roof and dual exhaust.

It turns out that none of these changes, either by themselves or in combination with others, change the class of languages recognized. So once we prove this, you can use any model of TM that you like.
Variations on the Turing machine model

A steam-powered Turing machine, from the University of Washington:
How do we prove that changes to the TM model do not change the computing power?

We do so the same way we did with DFA and NFA.

For DFA and NFA, we showed that a DFA (the less powerful model) could simulate the NFA (the more powerful model). Since a DFA is an NFA, but one that uses no nondeterminism, together with the previous result, this shows that DFA’s and NFA’s recognize the same class of languages.

So simulation is the key. For each new variation, we need to show the old kind of TM can simulate the new kind, and the new kind can simulate the old kind.
Allowing stationary moves

Here we add a new capability to our basic TM model: the ability to stay on the same cell during a move.

This means our transition function now has range

\[ Q \times \Gamma \times \{←, ↓, →\}, \]

where ↓ is a symbol indicating that we stay in the same cell.

We want to prove that this new model has the same computing power as the ordinary TM.

Because this new model gives TM’s a new capability, but does not change any other aspect of the TM, it suffices to show our basic model can simulate this new model.
Allowing stationary moves

Here the simulation is very easy: if in the new model we have a move like \( \delta(p, a) = (q, b, \downarrow) \), we simulate this in the old model by first moving to the right one cell, then moving back again in the next move.

To carry out the simulation, we first remove the transition \( \delta(p, a) = (q, b, \downarrow) \).

Then we introduce a new state \( q' \) for each \( q \) and add two new moves to the transition function:

- \( \delta(p, a) = (q', b, \rightarrow) \) and
- \( \delta(q', c) = (q, c, \leftarrow) \) for all \( c \in \Gamma \).

The first move rewrites the current cell as \( b \) and moves to the right, entering state \( q' \). Then no matter what is in the next cell (any \( c \) at all), in state \( q' \) we leave it the same and move left. These two moves achieve the simulation of the stationary move.
Multiple tracks

Another extension of a TM allows the tape to be divided into \( k \) tracks, for some fixed integer \( k \geq 1 \). (\( k \) cannot be infinite!)

Illustrated for \( k = 3 \):

Whenever the tape head scans a location on the tape, it can “see” all \( k \) symbols at the location. In the picture, the head is scanning the triple \([k, 2, c]\).
Multiple tracks

We can suppose that the input is given on the first track, and that all other tracks initially have blanks.

It turns out this is not really an extension at all!

Multiple tracks correspond to replacing the tape alphabet $\Gamma$ with the product of tape alphabets for each track: $\Gamma_1 \times \Gamma_2 \times \cdots \times \Gamma_k$.

So this is really just a different way of thinking about the same model.

But it is a very useful conceptual trick, as we will see later.
Now let us consider a TM with \( k \) tapes, for \( k \) a fixed integer \( \geq 1 \). Each tape has its own independent read-write head. The next move can depend on all symbols simultaneously scanned by all heads.

So this means the transition function looks like
\[
\delta(q, a_1, a_2, \ldots, a_k) = (p, b_1, b_2, \ldots, b_k, d_1, d_2, \ldots, d_k),
\]
where the \( a_i \) and \( b_i \) are tape symbols, and each \( d_i \) is either \( \leftarrow \) or \( \rightarrow \) or \( \downarrow \) (stationary move). Here \( a_i \) is the symbol on tape \( i \), and it gets replaced by \( b_i \).

By convention the input is on tape 1, and all the other tapes are initially blank.
Multiple tapes

Here’s a picture for $k = 2$: 

![Diagram of two tapes with labels and arrows indicating movement between them.](image)
As an example of the utility of multiple tapes, let’s design a 3-tape machine to recognize the language \( \{a^{k^2} : k \geq 1\}\).

The idea is that tape 1 will hold the input.

Tape 2 will hold \( j \) X’s, for some \( j \geq 1 \).

Tape 3 will hold \( j^2 \) X’s, for some \( j \geq 1 \).
The 3-tape machine then behaves as follows:

Step 1: (initialization) Start by writing an $X$ on each of tapes 2 and 3. Now $j = 1$.

Step 2: Compare the number of $a$’s on tape 1 to the number of $X$’s on tape 3. If they are equal, accept and halt. If there are more $X$’s than $a$’s, then reject and halt. Otherwise go to Step 3.

(The “compare” part can be done by moving to the right on both tapes provided tape 1 holds $a$ and tape 3 holds $X$. If the next cell on both is a blank, the number of symbols agrees. If the next cell is blank on tape 1 but $X$ on tape 3, then there are more $X$’s than $a$’s.)
Step 3: Copy the contents of tape 2 to the end of tape 3, twice. Then write one more $X$ on tape 3. If tape 3 had $j^2$ $X$’s before, now it has $j^2 + 2j + 1 = (j + 1)^2$ $X$’s.

(Copying can be done by moving to the end of tape 3 and to the beginning of tape 2, then changing blanks on tape 3 to $X$’s while simultaneously moving right on tape 2, and stopping this iteration when both tapes scan blanks.)

Step 4: Move to the end of the tape on tape 2 and write one more $X$. If it had $j$ $X$’s before, now it has $j + 1$ $X$’s.

Return to Step 2.
We can simulate a 2-tape machine with a 5-track TM.

The idea is that tracks 1 and 2 simulate tape 1, tracks 3 and 4 simulate tape 2, and track 5 has a symbol in the cell that represents the furthest point to the left the TM has gone.

How do we use two tracks to simulate a tape?

The idea is to use a “virtual” tape head. For tape 1 the virtual tape head is on track 2; for tape 2 it is on track 4. The virtual tape head is a symbol indicating the symbol currently being scanned on the corresponding tape.
Simulating 2-tape TM with 1-tape TM
At every step, what does our simulating TM do?

It first moves right until it sees the ↑ on track 2. Then it stores the corresponding symbol on track 1 in its state, entering a state like \((p, a_1, -)\).

Next it “rewinds”, moves left until it sees the \(X\) on track 5.

Then it moves right until it sees the ↑ on track 4. Then it stores the corresponding symbol on track 3 in its state, entering a state like \((p, a_1, a_2)\).

Now it knows \(p\), the state the original TM is in, and \(a_1\) and \(a_2\). Now we can carry out the move \(\delta(p, a_1, a_2) = (q, b_1, b_2, d_1, d_2)\). It now enters a state named something like \((p, a_1, a_2, q, b_1, b_2, d_1, d_2)\).
Simulating 2-tape TM with 1-tape TM

We rewind to the \( X \). Next, we move right until we see the \( \uparrow \) on track 2. Now we can replace the \( a_1 \) with \( b_1 \) on track 1. We can also move the \( \uparrow \) on track 2 in the proper direction.

We rewind to the \( X \). Next, we move right until we see the \( \uparrow \) on track 4. Now we can replace the \( a_2 \) with \( b_2 \) on track 3. We can also move the \( \uparrow \) on track 4 in the proper direction.

Finally, we change state to \( q \), rewind, and continue the simulation.

If we ever move left of the cell labeled \( X \), then we have to move the \( X \) one cell.

Thus we can simulate a two-tape TM with a one-tape TM. Obviously the simulation will have many states!

And the same kind of simulation works for any (finite) number of tapes \( k \), using \( 2k + 1 \) tracks.
One-sided tape

In this model, the tape does not extend arbitrarily far to both the left and right, but only arbitrarily far to the right.

Again, the computation starts by scanning a blank cell to the left of the input.

Here’s a picture of this kind of TM, in the initial configuration:
One-sided tape

This model has a new feature that we didn’t see in the basic two-sided model: namely, it’s possible to move off the left edge of the tape. So we have to decide what to do if this instruction is performed.

One way to handle this is as in Prof. Watrous’s notes: if we ever try to move left from the first cell, the “change state” and “rewrite symbol” are done, but we don’t move left.

We now have to show two things: a basic TM with two-sided tape can simulate a one-sided tape, and a machine with a one-sided tape can simulate a two-sided tape.

The first simulation is easiest, so let’s start with that.
One-sided tape

What we need is a way to detect when we’re at the left edge of the tape.

The way we can do this is first, before we do anything else, move left on the blank, write a special symbol $L$ indicating this is the left edge of the tape, and then move back to the blank:

Now we have a way of detecting the left edge.
Suppose we are scanning the $L$. Then the only way we could have gotten there is by moving left from the cell to the right. So we add the transition $\delta(q, L) = (q, L, \rightarrow)$ for each $q$. This has the effect of moving back to the cell to the right in the same state.

Thus we have simulated a one-sided tape with our basic two-sided model.
One-sided tape

The other direction — simulating a two-sided tape with a one-sided tape — is harder.

Here there are two different possible approaches to the simulation.

One simulation works as follows: whenever we would try to move off the left edge of the one-sided tape, we instead shift all the symbols on the tape to the right one cell, then move back all the way to the beginning.

There are two issues with doing this correctly. First, we need to carry out a “rewind” instruction that takes us all the way to the (simulated) left edge. To do this, we need a special symbol (say, $L$ again) that marks the left edge of the tape.

Second, in the “shifting” part, how do we know we’ve gotten to the end of the tape contents? The obvious solution — “move right until we see a blank” — doesn’t work, because the TM we are simulating could have rewritten some cells as blanks.
To solve this, we introduce another useful technique for TM’s. We never write a blank! Whenever the original TM we are simulating writes a blank, we write a new symbol (call it “blank-prime”) instead. A blank-prime is treated exactly as a blank would be; all transitions involving it are the same as for blanks. That way, whenever we move right and see an ordinary blank, we know we are at a cell that has never been visited before, so this must be the “right end” of the tape contents.

Combining these two ideas allow us to simulate a two-sided tape with a one-sided tape.

This isn’t the best possible simulation, though, because every time we try to walk left at the simulated left edge, we have to do a lot of work to shift the whole tape contents down. Would there be a more direct simulation that is more efficient?
The answer is yes, if we use two tracks. We can store the “right side” of the tape in the top track, and the “left side” of the tape in the bottom track, written in the reverse order. You can think of the two-sided tape as being folded into one.

For example the two-sided tape at the top can be simulated by the one-sided tape with two tracks at the bottom.

Notice that we have a special marker $L$ again, to mark where the tape ends.
Now we can add some extra transitions to carry out the simulation.

The basic idea is to keep track of whether we are scanning the symbol on the top track or bottom track.

The basic idea is that if we’re at the left edge (which we can detect with the $L$ in track 2) and scanning the top track, then a left move is interpreted as moving right and switching our attention to the bottom track.

Similarly, if we’re at the left edge and scanning the bottom track, we switch our attention to the top track instead.