Jeffrey Shallit
School of Computer Science
University of Waterloo
shallit@uwaterloo.ca
https://cs.uwaterloo.ca/~shallit
Now let’s add a new capability to our Turing machine model: **nondeterminism**.

This is just like nondeterminism in finite automata: from every configuration we are allowed to have no possible moves, 1 possible move, or more than one possible move.

Just like in NFA’s, we say that an input is accepted if some **sequence of choices** leads to the accepting state \( q_{\text{acc}} \).
As an example of nondeterminism, let’s build a 4-tape NDTM (nondeterministic Turing machine) to recognize the language \( \{ a^c : c \geq 4 \text{ is not prime} \} \).

The idea is to nondeterministically write some number \( m \) of \( a \)'s, at least 2, on tape 2.

Then we write some number \( n \) of \( a \)'s, at least 2, on tape 3.

Then we use our multiplication algorithm to write \( mn \) \( a \)'s on tape 4.

Then we compare the number of \( a \)'s on tape 4 to the number on tape 1, and accept if they are the same.

How do we nondeterministically write \( m \geq 2 \) \( a \)'s on tape 2?
An example of nondeterminism

On one tape we can do the following:

The idea is that we first write two $a$'s, then in state $q_2$ we have a nondeterministic choice: we can write another $a$ and move right, or we can decide to leave the blank unchanged and move to state $q_3$, where we rewind to the beginning of the tape.
The formal definition of a nondeterministic TM is very much like the formal definition of a deterministic TM.

Namely, it is a 7-tuple \( M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}}) \), where

- \( Q \) is a finite set of states;
- \( \Sigma \) is the finite input alphabet;
- \( \Gamma \) is the finite tape alphabet (and \( \Sigma \subseteq \Gamma \));
- \( q_0 \) is the initial state;
- \( q_{\text{acc}} \) is the accepting state;
- \( q_{\text{rej}} \) is the rejecting state.
- \( \delta \) is the transition function.
The domain of $\delta$ is $(Q - \{q_{\text{acc}}, q_{\text{rej}}\}) \times \Gamma$ and the range of $\delta$ is now $2^{Q \times \Gamma \times \{\leftarrow, \rightarrow\}}$.

Every nondeterministic TM has only finitely many choices from any given configuration.
Does nondeterminism give more power?

Now we come to one basic question about nondeterministic TM’s?

Do they give the same computing power as deterministic TM’s?

That is, do nondeterministic TM’s recognize the same class of languages as deterministic TM’s?

The answer is yes.
Does nondeterminism give more power?

To prove this, we just have to show how to simulate a nondeterministic TM with a deterministic TM.

How can we do this? Using something like the subset construction, which we used for DFA’s, cannot work, because a TM can have an infinite number of different configurations.

Instead we will use the following idea.

Think about a (potentially) infinite tree, that gives the possible sequence of configurations of a nondeterministic TM.

Each node represents a configuration.

The children of a node represent the possible different configurations of the TM that can arise from making the nondeterministic choices.
Does nondeterminism give more power?

For any given TM, the “branching factor” (number of different choices) is bounded by a constant.

We can always make this constant equal to 2, by replacing a state with multiple choices by a sequence of states, each making at most two choices.

In fact, by duplicating configurations when there is only one choice, we can assume that every configuration leads to exactly two configurations.
Does nondeterminism give more power?

Thus the computation of a nondeterministic TM can be viewed as a binary tree:

Here we have labeled each configuration with a binary string.
Does nondeterminism give more power?

This nondeterministic TM accepts an input $x$ iff some configuration in this binary tree is an accepting configuration.

If we are going to simulate a nondeterministic TM, we are going to have to visit each node in this binary tree and see if it is an accepting computation.

How can we do this? (The tree could be infinite.)

Depth-first search will not work! (We could get trapped in an infinite path down the left side.)

Instead, we use *breadth-first search*. This visits all nodes at the top level, then the next level, then the next, in that order: $\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, \ldots$

So we can simulate a nondeterministic TM using a 3-tape deterministic TM as follows:
Simulating NDTM with DTM

(DTM is our abbreviation for “deterministic Turing machine”.)

Tape 1 holds the input.

Tape 2 is a work tape where we simulate the computations of the nondeterministic TM.

Tape 3 holds binary strings of length 0, 1, 2, etc. that will tell us which path of the nondeterministic TM’s computations we will simulate. Initially it will be $\epsilon$. 

Simulating NDTM with DTM

We now do the following:

1. Copy Tape 1 to Tape 2.

2. Execute the steps of the nondeterministic TM on tape 2, choosing at each step which of the two possible next moves based on the next symbol of Tape 3.

3. If at any time the simulations enters \( q_{\text{acc}} \) of the original machine, halt and accept.

4. Otherwise, when the last choice on tape 3 is reached (by reading a blank on tape 3), rewind tape 3 and call an “incrementer” subroutine to replace tape 3 by the next binary string in radix order.

5. Erase tape 2 (using the trick we discussed before with the blank-prime).

Return to step 1.
Thus a nondeterministic choice like

\[ p \xrightarrow{\text{a, b \rightarrow}} q \quad \xleftarrow{\text{a, c \leftarrow}} r \]

is simulated by the two deterministic moves in the DTM below:

\[ (\text{--,a,0,0)(down,rightarrow,rightarrow}) \]

\[ (\text{--,a,1,0)(down,leftarrow,rightarrow}) \]

Here the – is just shorthand for many rules, one for each possibility in the place where the – is.
The binary counter

Let us see how to do the binary counter on tape 3. Each time it is called, it produces the next binary string in radix order.

The following TM carries this out.

State $q_1$ moves to the right end of the current tape contents.

It then moves left in state $q_2$, changing 1’s to 0’s until it sees a 0, which it then changes to a 1 in and goes to state $q_4$.

If there is no 0, then all 1’s are changed to 0’s and one more 0 is added at the right end, in the transition from state $q_3$ to state $q_4$.

Then we rewind the tape in state $q_4$ and reach the halting state.
Putting all these pieces together, we have simulated a NDTM with a DTM.

The simulation works as follows: if the NDTM accepts an input, the DTM also accepts the input.

If the NDTM does not accept an input, the DTM runs forever.
Deciding with a nondeterministic TM

So we have shown how to handle recognition by NDTM in our simulation with a deterministic TM.

But how about decision (instead of just recognition)?

First we have to think about what “deciding with an NDTM” would even mean!

When we decide with a deterministic TM, this means “if we wait long enough, the TM will eventually reach either the reject state or the accept state”.

The analogy for a nondeterministic TM then is “no matter what nondeterministic choices we make, we eventually reach either a reject state or accept state”.

Deciding with a nondeterministic TM

Why is that the right definition?

Because otherwise, there could be some sequence of nondeterministic choices for which no decision is ever made.

That does not correspond nicely to our understanding of what it means to “decide” a language.
Deciding with a nondeterministic TM

So if we adopt this definition of what it means to “decide” a language $L$ with an NDTM $M$, we can now simulate this with a deterministic TM $M'$ that also decides $L$.

We need the following result:

**König’s Lemma.** Every infinite tree (with finite outdegree at each node) has an infinite path.

*Proof.* Let $T$ be an infinite tree with finite outdegree.

Construct an infinite path $P$ as follows: start by putting the root $r$ on $P$.

$r$ has finitely many children, but infinitely many descendants. By the infinite pigeonhole principle, at least one child $c$ has infinitely many descendants. Choose $c$ as the next vertex on the path.

Continue with $c$ as above. This generates $P = (r, c, \ldots)$. 
Deciding with a nondeterministic TM

Let $T$ be a nondeterministic TM where every sequence of nondeterministic choices eventually reaches $q_{\text{acc}}$ or $q_{\text{rej}}$.

Hence the computation tree $T$, representing the configurations corresponding to all nondeterministic choices during the course of the computation, has no infinite path.

So by König’s Lemma, $T$ is finite.

Hence there is some $n$ such that if we consider all nondeterministic choice sequences of length $n$, either at least one reaches $q_{\text{acc}}$ or all of them reach $q_{\text{rej}}$.

In the former case we must accept; in the latter we must reject.
Therefore we can simulate an NDTM $M$ that decides with a DTM $M'$ that decides as follows:

1. Set $n = 1$.
2. flag := true.
3. For each binary string $s$ of length $n$ do steps 3a-3f.
   3a. Use the binary counter to compute $s$ on tape 3.
   3b. Copy tape 1 to tape 2.
   3c. Simulate the NDTM using the nondet. choices $s$ on tape 3.
   3d. If after using the choices in $s$ (or even a prefix) the state $q_{acc}$ was reached by $M$, then $M'$ halts and accepts.
   3e. If after using the choices in $s$ neither $q_{acc}$ nor $q_{rej}$ was reached, set flag := false.
   3f. Erase tape 2.
4. If flag = true then $M'$ halts and rejects. Otherwise, increment $n$ and return to step 2.
Deciding with a nondeterministic TM

This works because the tree of all nondeterministic choices is guaranteed to be finite.

Hence for every input, there must be a number $n$ such that all nondeterministic choices of length $n$ must reach either $q_{\text{acc}}$ or $q_{\text{rej}}$.

This $n$ is eventually found by the simulation.