Church's Thesis:

- Also known as **Church-Turing Thesis**.
- It is a hypothesis about computable functions.
- It says that any real-world computation can be translated into an equivalent computation involving a Turing machine.
  I.e. It states that a function on the natural numbers is computable by a human following an algorithm iff the function is computable by a Turing Machine.
- **Mathematical evidence for Church's Thesis:**
  1. **Extensions to TMs are equivalent to TMs.**
     We're dealing with different ways of looking at the same problem.
  2. **Competing formalisms for algorithms are equivalent.**
     Again, we're dealing with different ways of looking at the same problem.
     People had not bothered to define algorithms for thousands of years because it was not necessary. Then, in the space of 3 years, 3 different definitions were proposed.
     The first way is **partial recursive functions** and it was done by Gödel and Herbrandt in 1933.
     The second way is **λ-calculus (lambda calculus)** and it was done by Church and his student Kleene in 1936.
     The third way is **Turing Machines** and it was done by Turing in 1936.
  3. **Universality**

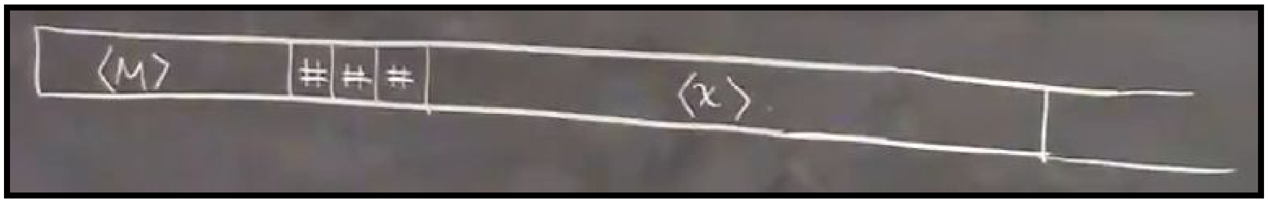
Universality:

- Consider a TM, M_u, that takes in as input a description of any TM, M, and any string, x, that M can work on and simulates the computation M on x. M_u is called the **universal TM**.
- These universal TMs have a self-referential aspect. They can talk about TMs. However, this is the source of some problems. Recall from the first lecture that many of the issues with paradoxes stemmed from the fact that they had a self-referential aspect.
- Going back to M_u, there is an issue. Like all TMs, M_u must have a set of states, an input alphabet, a tape alphabet, etc. The problem lies with its tape alphabet. The tape alphabet consists of a fixed, finite set of symbols, but it's supposed to simulate any TM that could have an entirely different set of symbols. So, I need to be able to find a way to encode an arbitrary TM that has its own states and its own tape alphabet and its own input alphabet using the fixed tape alphabet of M_u. There are many different methods we can use to encode and it doesn't matter which method we use. I will illustrate one possible encoding of TMs using the alphabet {0, 1, #}, where # acts as a separator. We will use an arbitrary TM M, s.t. M = (Q, Σ, Γ, δ, Q0, Qa, Qr). Here's how I'll encode M:
  1. Encode state q by a binary string, denoted as ⟨q⟩, of $\text{ceil}(\log_2^{|Q|})$ bits.
  2. Encode Q by listing the codes of its elements separated by #'s.
     I.e. We can encode Q as ⟨Q0⟩#⟨Qa⟩#⟨Qr⟩#⟨Q1⟩#...
     We first list the initial state, followed by the accept state, then the reject state, and then the rest of the states in order.
  3. Encode symbol a ∊ Γ by a binary string, denoted as ⟨a⟩, of $\text{ceil}(\log_2^{|\Gamma|})$ bits.
  4. Encode Σ by listing the codes of symbols in Σ separated by #'s.
     I.e. We can encode Σ as ⟨a⟩#⟨b⟩#⟨c⟩#...
  5. Encode Γ similarly to Σ, but we start the list with the code for the blank symbol, ⊔.
  6. Encode x = a1a2a3...ak ∊ Γ^* as ⟨a1⟩⟨a2⟩⟨a3⟩…

7. Encode transition δ(q,a) = (p, b, D) as ⟨q⟩#⟨a⟩#⟨p⟩#⟨b⟩#⟨D⟩.
8. Encode δ as ⟨transition1⟩##⟨transition2⟩##⟨transition3⟩…
9. Encode M, denoted as ⟨M⟩, as ⟨Q⟩###⟨Σ⟩###⟨Γ⟩###⟨δ⟩.

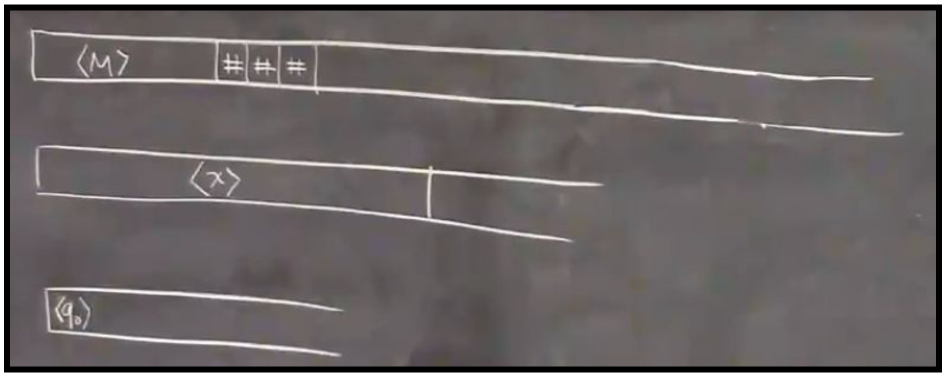
Note: Strings in $\{0,1,\#\}^*$ that don't encode a TM as above, encode TM that rejects all strings.

- The universal TM, M_u, takes as input encoding of a TM, M, and its input x and simulates M on x.
  I.e. The input of M_u is: ⟨M,x⟩
- **Theorem 3.2:** There is a TM M_u called the universal TM that:
  - accepts ⟨M,x⟩ if M accepts x
  - rejects ⟨M,x⟩ if M rejects x
  - loops on ⟨M,x⟩ if M loops on x
- We can describe M_u as a 3-tape TM. This can be converted into a regular 1-tape TM.
  The tape alphabet of M_u is {0,1,#,⊔}.
  Recall that with multi-tape TMs, the input arrives in the leftmost cell(s) of tape 1 and the other tapes are left blank.
  The first tape of M_u is the input tape. The input tape contains an encoding of some TM, M, and an encoding of M's input, x.



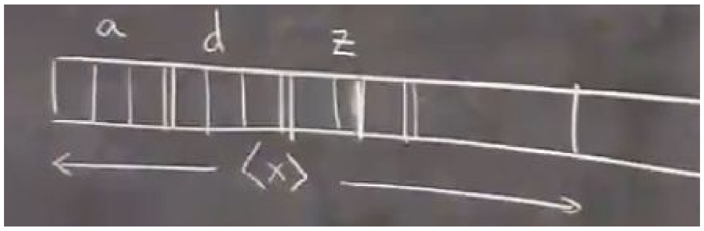
The second tape of M_u is initially blank but it will eventually contain M's encoded tape.
The third tape of M_u is initially blank but it will eventually contain the encoded state of M.

The first thing that M_u does is that it copies ⟨x⟩ onto the second tape, erases ⟨x⟩ from the first tape and puts on the third tape the encoding of the initial state of M.



Recall that every symbol in ⟨x⟩ is represented by ceil($\log_2^{|\Gamma|}$) bits. Suppose that ceil($\log_2^{|\Gamma|}$) is 3.
I.e. I need 3 bits to represent every tape symbol in ⟨x⟩.

Suppose that the first 3 bits represent "a", the second 3 bits represent "d" and the third 3 bits represent "z", as shown below.



Lastly, suppose that each tape head starts out by pointing to the first cell of their respective tape.

This is how M_u works:

1. It says, let me go to tape 3 and see what is the state.
2. Then, it goes to tape 2 and reads the 3 cells (bits) that make up the first symbol. Now, M_u has figured out that it is simulating M in state Q0, scanning symbol "a".
3. Then, it goes to tape 1 and finds the transition function "⟨Q0⟩#⟨a⟩" and looks for should happen next. Suppose that the entire transition function is ⟨Q0⟩#⟨a⟩#⟨p⟩#⟨b⟩#⟨R⟩.
4. Then, M_u changes the state in tape 3 to ⟨p⟩, the 3 bits that represent "a", in tape 2, to the 3 bits that represent "b", and then it moves 3 cells to the right in tape 2. It moves 3 cells to the right because every 3 cells represents one symbol.
5. M_u has now simulated 1 move of M. It then repeats steps 1-4 for the new state and symbol. It keeps repeating until it sees Qa or Qr, at which, it will either accept or reject.

- **Theorem 3.3:** If L is a decidable language, then its complement is also decidable.
  I.e. The set of decidable languages is closed under complementation.

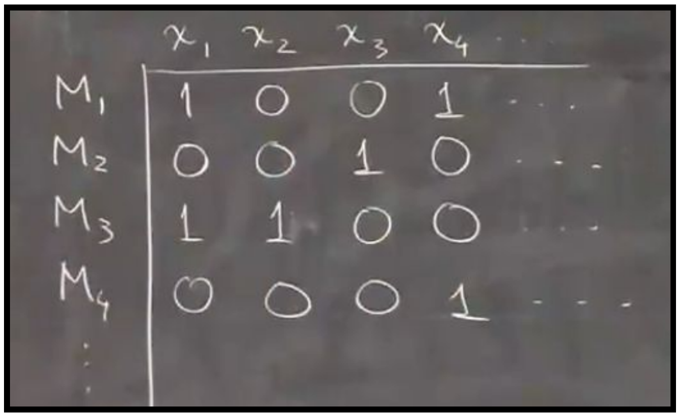
  Proof:
  Suppose that if L is decidable, M is a TM that decides L. Suppose that M takes an input, x. Then, M either accepts x, if $x \in L$ or rejects x, if $x \notin L$. Using M, we can construct another TM, M', that decides the complement of L, L'. If M accepts x, M' rejects x. If M rejects x, M' accepts x.
- Proof that there are languages that are not recognizable:

  Fix Σ to include enough symbols to encode TMs.
  E.g. Σ = {0, 1, #}
  Now, consider all languages over Σ.
  I will define 2 languages, U, the universal language, and D, diagonal.
  U = {⟨M, x⟩ | M accepts x}
  D = {⟨M⟩ | M does not accept ⟨M⟩}

  Theorem 3.4: D is not recognizable.

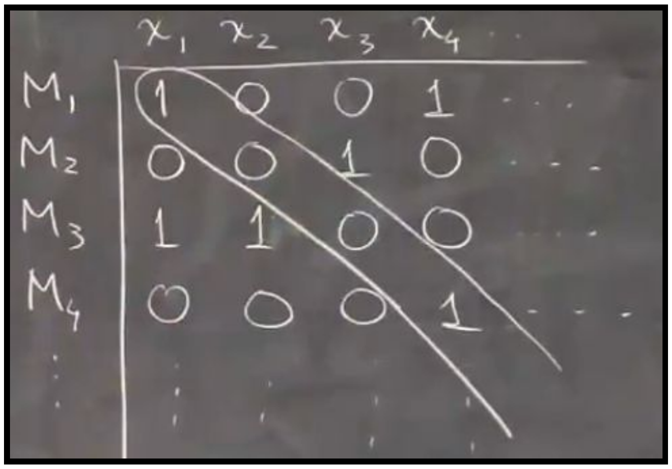
  Proof:
  Suppose for contradiction that D is recognizable. Let M_D be a TM that recognizes D.
  This means that ∀ TM M, M_D accepts ⟨M⟩ iff M does not accept ⟨M⟩.
  Take M = M_D. This means that M_D accepts ⟨M_D⟩ iff M_D does not accept ⟨M_D⟩.
  This is a contradiction. Hence, D is not recognizable.
  Another way to think about the proof:
  Let's enumerate all the TMs M1, M2, M3, …, over all of their inputs.

We put a 1 if Mi accepts Xj and 0 otherwise.
An example table is shown below.



	x_1	x_2	x_3	x_4	…
M_1	1	0	0	1	…
M_2	0	0	1	0	…
M_3	1	1	0	0	…
M_4	0	0	0	1	…
⋮					

Let’s look at the main diagonal of the table, shown below.



	x_1	x_2	x_3	x_4	…
M_1	1	0	0	1	…
M_2	0	0	1	0	…
M_3	1	1	0	0	…
M_4	0	0	0	1	…
⋮	⋮	⋮	⋮	⋮	

I will complement the string along the main diagonal.
I.e. 1001 becomes 0110.
Let D be the complement string (0110…).
If M_D existed, it would be in one of the enumerations, but M_D cannot be any of the TMs enumerated because it differs from every TM by at least 1 input.
Hence, M_D doesn’t exist.

- **Theorem 3.5:** U, the universal language, is
  a) recognizable, but
  b) not decidable.
  U = {⟨M, x⟩ | M accepts x}

  Proof of a):
  M_u recognizes U.

  Proof of b):
  Suppose for contradiction that U is decidable. Let M_1 be a TM that decides it.
  We can use M_1 to decide the complement of D, ¬D.
  ¬D = {⟨M⟩ | M accepts ⟨M⟩}.
  Here’s how we can use M_1 to decide ¬D.
  We can use M_1 to create another TM, M_2.

M_2 on input ⟨M⟩:

1. Construct ⟨M, ⟨M⟩⟩
2. Run M_1 on ⟨M, ⟨M⟩⟩
3. If M_1 accepts, accept
4. Otherwise, reject

If M_2 is a decider for ¬D, then ¬(¬D), the complement of ¬D, or D is also decidable. However, in theorem 3.4, we proved that D is not recognizable. Hence, this is a contradiction. Hence, U is not decidable.

- The proof of theorem 3.5, part b, is a **proof by reduction**.
  Problem P reduces to problem Q iff there is an algorithm to solve P that uses an assumed algorithm for problem Q as a black box.
  I.e. We can't look into the code of the algorithm to see how it works. We just give it an input and it gives back an output.

  For the proof of theorem 3.5, part b, we reduced ¬D to U, which can be written as ¬D ≤ U.
- **Theorem 3.6:** If L and ¬L, the complement of L, are both recognizable, then L and ¬L are decidable.

  Proof:
  Suppose that L and ¬L are both recognizable.
  Let M_1 be a TM that recognizes L and let M_2 be a TM that recognizes ¬L.
  Construct a TM, M, that decides L.
  M simulates M_1 for 1 step and M_2 for 1 step and repeats until either M_1 or M_2 accepts.
  This is because the input must be either in M_1 or M_2. It must either be in L or not be in L.
  If M_1 accepts, then accept.
  If M_2 accepts, then reject.
- **Corollary 3.7:** The complement of U, ¬U, is not recognizable.
  ¬U = {⟨M, x⟩ | M does not accepts x}

  Proof:
  Suppose for contradiction that ¬U is recognizable.
  In theorem 3.5 a), we said that U is recognizable.
  Hence, by theorem 3.6, both U and ¬U are decidable.
  However, in theorem 3.5 b), we said that U is not decidable.
  Hence, this is a contradiction and ¬U is not recognizable.
- **Corollary 3.8:** The set of recognizable languages is not closed under complementation.