Lecture 7 - Undecidability and the Church-Turing Thesis

Eric A. Autry

Course Office Hours

Monday 9-11 am	Zhe Wang	Physics 154
Monday 2-4 pm	Zhe Wang	LSRC B105
Monday 5:30-7:30 pm	**Eric Autry** Zicheng Yuan	**Fishbowl CIEMAS 3602**
Tuesday 1-3 pm	Yuanyuan Yu	Hudson Hall 232
Wednesday 8 am - noon	Yu Cao	Fishbowl CIEMAS 3602
Wednesday 2-4 pm	Siyuan Liu	LSRC D106
Wednesday 4-8 pm	**Eric Autry** Yuanyuan Yu Zicheng Yuan	**Gross Hall 304B**
Thursday 12-2 pm	Siyuan Liu	Fishbowl CIEMAS 3602

Last time: Turing Machines and Infinity

This time: Undecidability and the Church-Turing Thesis

Next time: The Halting Problem and Reductions

Homework 2 solutions are posted on Sakai.

Homework 3 is posted on Sakai and due **this Thursday the 27th.**

- Skip problem 4(c).
- Typo in problem 5 (new version uploaded). Should be all subsets of $\mathbb{N}$.

Midterm is **next Tuesday the 2nd.**

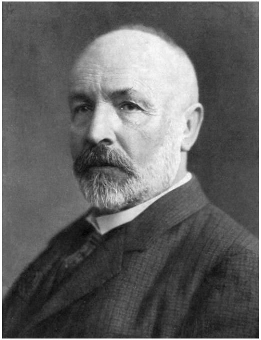
Real Numbers

Theorem

The real numbers $\mathbb{R}$ are uncountable.

Proof:

Diagonalization proof developed by Cantor in 1891.



Real Numbers

Theorem

The real numbers $\mathbb{R}$ *are uncountable.*

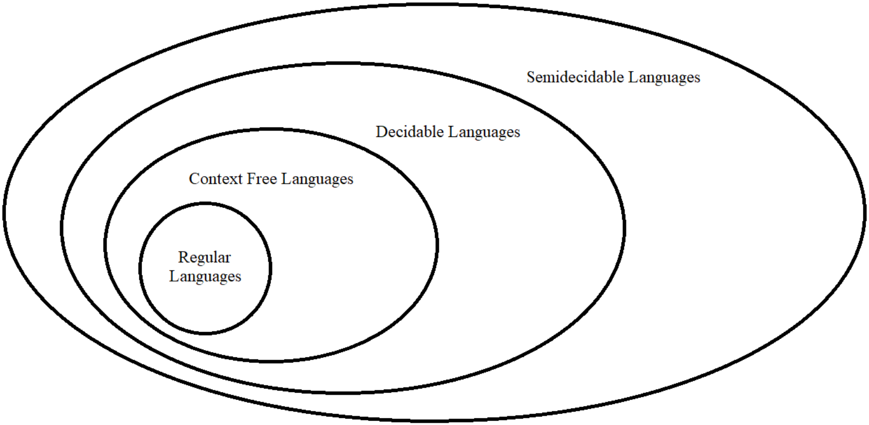
Proof:

This new number cannot be in the original list.

1	0	.	5	0	0	0	0	...	
2	3	.	1	4	1	5	9	...	
3	2	.	7	1	8	2	8	...	
4	1	.	4	1	4	2	1	...	
5	1	.	7	3	2	0	5	...	
.	.	.	.	.	.	.	.		
.	.	.	.	.	.	.	.		
.	.	.	.	.	.	.	.		

0 . 4 7 6 1 9

Context Free vs Decidable vs Semidecidable



Question: are all languages decidable?

Decidable Languages

If a Turing machine halts on all inputs and either accepts or rejects, the language it recognizes is called a **decidable** language. (These are also known as recursive languages.)

When this happens, we say that the Turing machine **decides** the language.

What if there is an input that causes the Turing Machine to never halt?

Semidecidable Languages

If a Turing machine

- halts and accepts all strings in a language A,

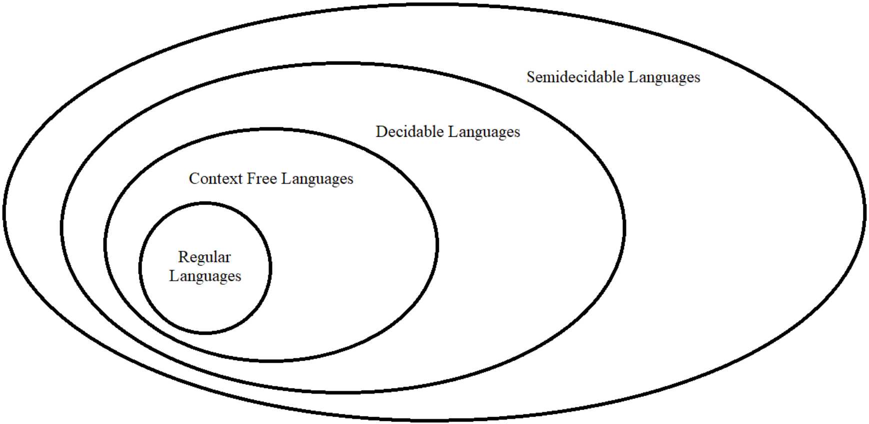
and for strings that are not in A, **either**:

- halts and rejects, **or**
- loops forever,

we say the Turing machine **recognizes** the language A and call the language **semidecidable.** (These are also known as Turing-recognizable or recursively enumerable languages.)

Note: for these semidecidable languages, we are treating infinite loops as a form of rejection.

Context Free vs Decidable vs Semidecidable



Question: can we find a language that is not decidable?

Acceptance Problem for DFAs

A **decidable language about DFAs**:

$$A_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$$

1. Simulate machine B on input w.
   (a) Mark the start state and the first input of w.
   (b) Read the marked input of w and the marked state of B.
   (c) Follow the corresponding transition, marking the new state of B and the next symbol of w.
   (d) If the next symbol is not black, return to Step 1(b), otherwise continue to Step 2.
2. If B ends in an accepting state, accept. If it ends in a nonaccepting state, reject.

Note: we can do this for NFAs, Regular Expressions, PDAs, and CFGs too!

Acceptance Problem for Turing Machines

Question: can we do the same thing with Turing Machines?

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts input string } w\}$$

What happens when we try to simulate machine M?

Clearly if it accepts, we accept. And if it rejects, we reject.

But what if machine M enters an infinite loop?

Then our new machine will also loop. **Therefore A_{TM} is semidecidable, but is not decidable!**

Can't we try to detect whether the machine is infinitely looping?

- This is called the Halting Problem, and we will prove later that it cannot be done.

(Note, a Turing Machine which simulates other Turing Machines is often labeled as U, and called the universal Turing Machine.)

Undecidable Languages

So, we have now found a language that was semidecidable but not decidable.

The next natural question is then: are there languages that are not even semidecidable?

Theorem

Some languages cannot be recognized by a Turing Machine.

Proof:

There are a countable number of Turing Machines. Each machine can be encoded as a finite string, meaning that the set of all Turing Machines corresponds to a subset of all finite strings. The set of all finite strings is countable (homework problem), and so the set of all Turing Machines is countable.

Undecidable Languages

Theorem

Some languages cannot be recognized by a Turing Machine.

Proof:

There are a countable number of Turing Machines.

How many languages are there?

Key idea: a language is a set of strings. We often use some rule to define a language (like all strings that end in 0), but a rule is not required.

Let's use a diagonalization argument to show that the set of all languages is uncountable.

Undecidable Languages

Theorem

Some languages cannot be recognized by a Turing Machine.

Proof:

Let's use a diagonalization argument to show that the set of all languages is uncountable.

$$S = \{L \mid L \text{ is a language made up of binary strings}\}$$

Note: the set of all finite length binary strings is countable, i.e., we can write $b_1, b_2, \ldots, b_k, \ldots$

$$0, \quad 00, \quad 01, \quad 10, \quad 11, \quad 000, \quad 001, \quad 010, \quad \ldots$$

Undecidable Languages

Theorem

Some languages cannot be recognized by a Turing Machine.

Proof:

$$S = \{L \mid L \text{ is a language made up of binary strings}\}$$

Assume by way of contradiction that S is countable.

Then a list of these binary languages must exist:

$$L_1, \quad L_2, \quad L_3, \quad \ldots$$

Let's construct a binary language that is not in that list...

Undecidable Languages

Theorem

Some languages cannot be recognized by a Turing Machine.

Proof:

Let's construct a binary language that is not in that list...

1. Look at language L_1.
    - If the string b_1 **is** in L_1, then it **is not** in our new language.
    - If the string b_1 **is not** in L_1, then it **is** in our new language.
2. Look at language L_2.
    - If the string b_2 **is** in L_2, then it **is not** in our new language.
    - If the string b_2 **is not** in L_2, then it **is** in our new language.
3. Look at language L_3.
    - If the string b_3 **is** in L_3, then it **is not** in our new language.
    - If the string b_3 **is not** in L_3, then it **is** in our new language.
4. etc

Undecidable Languages

Theorem

Some languages cannot be recognized by a Turing Machine.

Proof:

We've defined a binary language, because all of the string in the language are binary strings.

But, our new binary language could not have been in S because it was different than every language in S. So S was incomplete and we have reached our contradiction.

There are a countable number of Turing Machines, but an uncountable number of languages!

Undecidable Languages

There are a countable number of Turing Machines, but an uncountable number of languages!

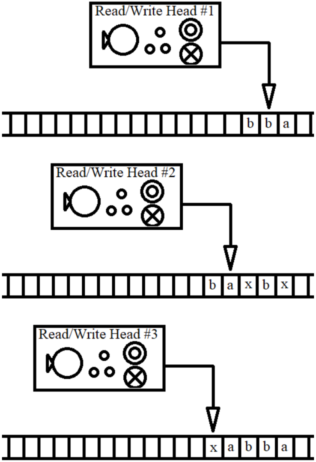
But, we've had problems before...

- When we saw that DFAs were limited, we added an infinite stack to get the more powerful PDAs.
- When we saw that PDAs were limited, we added an infinite tape to get the even more powerful Turing Machines.
- Now we see that Turing Machines are limited...

Can we build a better Turing Machine?

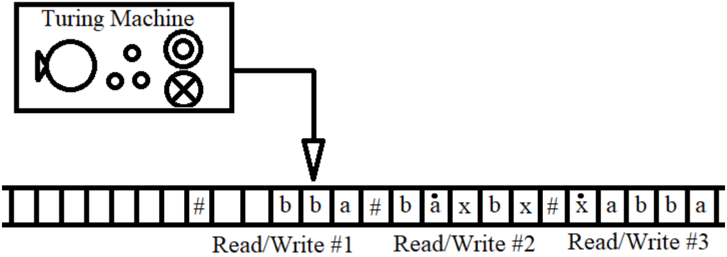
Turing Machine Variant #1

What if we gave the machine multiple infinite tapes, and each tape had its own read/write head?



Turing Machine Variant $\#1$

Nope.



- Just simulate each separate tape/head one at a time.
- If one of the tapes runs out of space, pause and shift everything to give it more space.

Turing Machine Variant #2

What if we allowed nondeterministic behavior?

Nope. Let's build a 3-tape Turing Machine that is equivalent to a nondeterministic Turing Machine.

- Note that we just showed that a 3-tape TM is equivalent to a 1-tape TM.

Define the 3 tapes as:

Tape 1: The input tape. Stores the input and never changes it.

Tape 2: The simulation tape.

Tape 3: The nondeterminism tape.

Turing Machine Variant #2

Tape 1: The input tape. Stores the input and never changes it.

Tape 2: The simulation tape.

Tape 3: The nondeterminism tape.

- Copy the input to the simulation tape and run the machine.
- If there is a choice, mark both options on the third tape.
- For each **step** we take, reset the simulation tape and rerun the machine following the branch specified by the third tape until we reach the **next step** of computation.
- Increment the third tape to look at the next branch and repeat until we've made a single step for each branch. Then perform another **single step** of computation.
- If a branch rejects, remove it from the third tape.
- If we ever reach the accept state, accept. If all branches on the third tape reject, then reject.

Weaker Turing Machines?

It turns out, we can put limitations on a Turing Machine that do not decrease its power:

- What if we cut the tape at one end so it is only infinite in one direction?
  - Still equivalent.
- What if we don't allow the machine to stand still, forcing it to always move?
  - Still equivalent.
- What if we don't allow the machine to move left, instead forcing it to move right or reset to its initial position?
  - Still equivalent.

Church-Turing Thesis

Since 1936, many variations of machines have been proposed, but all have ended up being equivalent to (or less powerful than) a Turing Machine.

This lead to the Church-Turing Thesis:

- It basically states that the intuitive notion of an algorithm is equivalent to Turing Machine algorithms.

i.e.,

- If a solution to a problem is calculable, a Turing Machine can compute it.

This hasn't been proven, but so far nobody has come up with a better machine...

What about quantum computing?

Proving Undecidability

So, there exist undecidable problems...

How do we show if a problem is undecidable?

We can either prove it by directly considering the problem,
or **reduce** the problem to another undecidable problem.

Famous undecidable problem used in many proofs: The Halting Problem.

Proving Undecidability

Let's prove that A_{TM} is undecidable:

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts input string } w\}$$

Proof:
Let's assume by way of contradiction that A_{TM} is decidable.

Then there exists a Turing Machine H that decided the language, i.e.,

$$H(\langle M, w\rangle) = \begin{cases} accept & \text{if } M \text{ accepts } w, \\ reject & \text{if } M \text{ rejects } w. \end{cases}$$

Proving Undecidability

Let's prove that A_{TM} is undecidable:

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts input string } w\}$$

Proof:

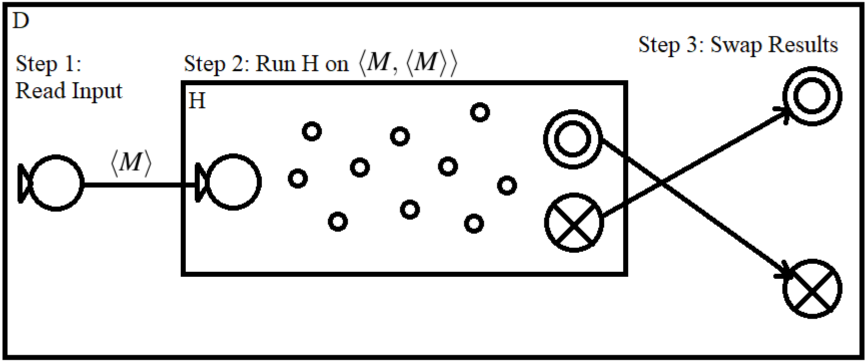
Let's create a Turing Machine D that takes in another Turing Machine M, runs H on M with the string representation of M as the input, then returns the opposite:

1. Run H on the input $\langle M, \langle M\rangle\rangle$.
2. If H accepts, then return reject. If H rejects, then return accept.

$$D(\langle M\rangle) = \begin{cases} accept & \text{if } M \text{ does not accept string } \langle M\rangle, \\ reject & \text{if } M \text{ accepts string } \langle M\rangle. \end{cases}$$

Proving Undecidability

$$D(\langle M \rangle) = \begin{cases} \mathit{accept} & \text{if } M \text{ does not accept string } \langle M \rangle, \\ \mathit{reject} & \text{if } M \text{ accepts string } \langle M \rangle. \end{cases}$$



Proving Undecidability

Let's prove that A_{TM} is undecidable:

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts input string } w\}$$

Proof:

What happens when we run D on itself?

$$D(\langle D\rangle) = \begin{cases} accept & \text{if } D \text{ does not accept string } \langle D\rangle, \\ reject & \text{if } D \text{ accepts string } \langle D\rangle. \end{cases}$$

Wait... D can only accept if it rejects, and only reject if it accepts?

D cannot exists. But, we built D using only the machine H.

Therefore H cannot exist and there is no Turing Machine that can decide A_{TM}.