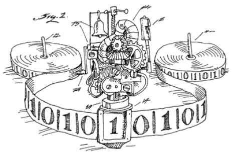
EECS 376: Foundations of Computer Science

Lecture 10 - Diagonalization and Undecidability



Code vs TMs

- Claim: Given any TM, we can simulate it using a Boolean function on string written in C++ code.

```
bool simulateM(string x):
// - hard-coded transition function of TM M
// - copy x into array representing a tape
// - repeatedly read/write/move according M
return accept/reject according to M
```

- **Q:** Given any Boolean function on strings written in C++ code , can we simulate it using a TM?
- **A:** Yes. It is tedious to prove.
- Church-Turing thesis:
  - TM can simulate any model of computation.

Take away: TM ≡ "bool **M**(string x)"

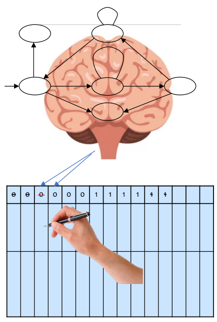
Today's Agenda

- Recap: Turing Machines, Church-Turing thesis
- **Deciders** (vs "loopers") and **decidability**
- **Diagonalization** and an **undecidable language**

Summary So Far

- **General:** Any *finite* object (integer, graph, PDF, C++ code) can be encoded as a finite string. A TM takes a finite string as input.
- **Church–Turing thesis:**
  "Anything that is computable by some physical device (a 'computer') is computable by some **Turing machine**."
- **In short:** Turing Machines = computer programs.
- **Implication:** If a problem *is not* decidable by a TM, it *cannot* be solved by *any* computer! (including future/alien technology)

Turing Machines: Essential Features



1. *Finite* **alphabets**: input & tape
2. *Finite* "brain"/logic/"**code**": state machine
3. *Infinite* **storage:** "tape"
4. Sequence of *local* computations, specified by the code:
   - Read a single symbol
   - Write a single symbol
   - Move a single cell
   - Update active state
5. Runs until it enters a *terminal* state—if ever!

Proving Decidability

Decision Programs

- **Q:** Suppose we run a TM on string. What are the possible outcomes?
  - either: (i) accepts, (ii) rejects, or (iii) "loops" (forever)
- **Definition:** A TM M ***decides*** language L if :
  1. M accepts every string $x \in L$ ("accepts"), and
  2. M rejects every string $x \notin L$ ("rejects").

We say that M is a ***decider*** (for L), and L is ***decidable***.

- **Note:** By definition, M does not loop on any input!

no loop allowed!

Proving Decidability

- **Q:** To prove that a language L is decidable, must we design an *actual* TM?
- **A:** No! *Simulation* lets us write an algorithm in C++ or pseudocode.
- **Example:**
  - $L = \{(n, m) \mid n$ and m are coprime$\}$.

CoPrime(n, m): If $n < m$, swap n and m. If **Euclid**$(n, m) = 1$, return "accept" Else return "reject".	**Euclid**(x, y): // for $x \geq y > 0$ if($x \bmod y = 0$), return y. else return **Euclid**$(y, x \bmod y)$

- **Analysis (Correctness):**
  - By definition, $\gcd(n, m) = 1 \iff$ n and m are coprime.
  - Euclid(x, y) always halts and returns $\gcd(x, y)$.
  - There exists a TM that *simulates* CoPrime, Euclid and therefore decides L.

Undecidable Languages?

- **Question:** Do there exist *undecidable* languages?
  i.e., are there problems that no computer can solve?

- **The goal of today's lecture:** There ***exists*** an undecidable language L .

- **The key idea of today's lecture:**
- Let $\mathcal{L}$ be the set of *all* languages and $\mathcal{M}$ be the set of all TMs, say both over the alphabet $\Sigma = \{0,1\}$.
- If we could show that $|\mathcal{M}| < |\mathcal{L}|$, we would be done!
- **Why:** Each TM M decides at most 1 language.
- **Problem:** Both $|\mathcal{L}|$ and $|\mathcal{M}|$ are infinite! Can we do anything about it?

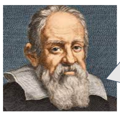
Before Proving Undecidability:

Introduction to Countable and Uncountable Sets

How can we compare the "size" of infinite sets?

---

203 Review: Countability



The attributes "equal," "greater," and "less," are not applicable to infinite, but only to finite, quantities.

Galileo (1638)

Actually yes they are

Georg Cantor (1895)

- **Definition:** A set S is ***countable*** if it is "no larger than" the naturals $\mathbb{N} = \{0,1,2,\ldots\}$, i.e. $|S| \leq |\mathbb{N}|$.
- **Equivalently:** S is countable if there exists a 1-to-1 (injective) function $f: S \to \mathbb{N}$.
- We can also show S is countable by demonstrating how to list all the elements in S such that each element $s \in S$ appears *somewhere* on the list. Why?

Transfinite Cardinal Numbers

- Cardinality of a *finite* set is simply the number of elements in the set.
- Cardinalities of *infinite* sets are not natural numbers, but are special objects called *transfinite cardinal numbers*.
- $\aleph_0$:=|**N**|, is the *first transfinite cardinal* number.
- *continuum hypothesis* claims that |**R**|=$\aleph_1$, the *second transfinite cardinal*.

Ordinals (con.)

- The least ordinal is (vacuously) = ∅. We take:
- 0= ∅
- 1={∅}={0}
- 2 = {∅,{∅}} = {0,1}
- 3 = {∅, {∅},{∅,{∅}}} = {0,1,2}
- 4 = ... = (0,1,2,3}

and so on. In general, we have

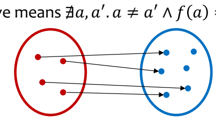
n = {0,1,2, ..., n-1}

Ordinals (con.)

- So, every natural number is an ordinal.
- But the ordinals continue after the natural numbers leave off. If ω denotes the smallest ordinal which is not a natural number, then ω is the set of natural numbers.
- Then the next ordinal after ω is $\omega \cup \{\omega\}$ = ω +1, and so on .

Functions and Set Cardinality

- A function $f: A \to B$ maps each element $x \in A$ to an element $f(x) \in B$.
- A function f is ***injective*** **(1-to-1)** if no two elements in A are mapped to the same element of B.
  - Formally: f injective means $\nexists a, a'. a \neq a' \wedge f(a) = f(a')$.



- If an injective $f: A \to B$ exists, then $|A| \leq |B|$.
- If injective $f: A \to B$ and $g: B \to A$ exist, then $|A| = |B|$.
- This is the ***definition*** of "≤" and "=" for set cardinality.

Warning: properties of "≤" for finite values do not necessarily apply to infinite values

---

Countable Sets

- **Def:** A set S is countable if it is "no larger than" the naturals $\mathbb{N} = \{0,1,2,\ldots\}$, i.e. $|S| \leq |\mathbb{N}|$.
- **Equivalently:** S is countable if $\exists$ an injective function $f: S \to \mathbb{N}$.

- **Claim:** Any finite set is countable.
- **Proof:**
  - Let $S = \{s_1, \ldots, s_n\}$ be a set with n elements.
  - Then $f: S \to \mathbb{N},\ f(s_i) = i$ is an injection from S to $\mathbb{N}$.

- **Q:** Which *infinite* sets are countable ("countably infinite")?

Proving Countability

- **Recall:** S is countable if there is an injective $f: S \to \mathbb{N}$.
- We can prove that a set is countable by explicitly defining such a function.
- Or, we can show how to *list* elements of S (possibly with duplicates) so that each element must appear *at some finite position* in the list.
  - This is enough. It implicitly defines an injective $f: S \to \mathbb{N}$.

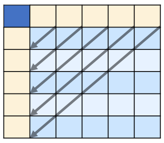
List including all elements in S: f

	s_1	s_2	s_3	s_2	s_5	...
	1	2	3		4	...

Countably Infinite Sets

- **By Definition:** $\mathbb{N} = \{0, 1, 2, 3, 4, \dots\}$.
- Is $\mathbb{Z}$ countable (listable)?
  - First try: $\mathbb{Z} = \{0,1,2,3,4 \dots, -1, -2, -3, -4, \dots\}$.
  - **Does not work!** (What is the exact position of "-1" in the list?)
  - **2nd Try:** List $\mathbb{Z} = \{0,1,-1,2,-2,3,-3,4,-4,5,-5, \dots\}$.
- Is $\mathbb{Q}^+$ countable (listable)?
  - How to list $\mathbb{Q}^+$: First list all x/y with $x + y = 1$, then $x + y = 2$, etc.

	1	2	3	4	...
1	1/1	1/2	1/3	1/4	...
2	2/1	~~2/2~~	2/3	~~2/4~~	...
3	3/1	3/2	~~3/3~~	3/4	...
4	4/1	~~4/2~~	4/3	~~4/4~~	...
...	...	...	...	...	~~...~~



More Countably Infinite Sets: Finite Binary Strings

- Let $S = \{0,1\}^*$ be the set of all *finite* binary strings.
- **Claim:** S is countable.
- **Proof:** List the elements of in *lexicographic order:*
  - by length, and then
  - in sorted order among strings with equal length.

  $S: \epsilon, 0, 1, 00, 01, 10, 11, 000, \dots$
- Every element of S appears in the list.

Diagonalization:

Showing |Countable set| < |Uncountable set|

Proving Uncountability via Diagonalization

- **Q:** How do we show that a set S is *uncountable*?
- **A:** Prove that no injective $f: S \to \mathbb{N}$ exists.
- **How?** Proof by contradiction! Template:
  1. Assume there exists a list of elements of S such that every element $x \in S$ appears somewhere in the list.
  2. Use it to 'construct' some $x^* \notin S$ that is not in the list.
  3. Contradiction! So, no such list can exist.
- ***Diagonalization*** is the usual technique for step 2

First Example:

Set of *infinite-length* binary sequences is uncountable

- Now let S be the set of *infinite-length* binary sequences.
- Suppose there is some list $(s_1, s_2, s_3, \dots)$ of all elements of S.
- Let $s_i[j]$ be the jth bit of s_i.
- Take the 'diagonal' bits and flip them:
  - $x^*[j] = \overline{s_j[j]} = 1 - s_j[j]$.

s_1	0	1	1	0	1	0	...
s_2	1	0	0	0	0	0	...
s_3	0	1	1	1	0	1	...
s_4	0	0	0	0	0	0	...
s_5	1	1	1	1	1	1	...
...	...	...	...	...	...	...	...

x^*	1	1	0	1	0	...	

First Example:

Set of *infinite-length* binary sequences is uncountable

- **Claim:** x^* is not in the list!
- **Proof:** If it were in the list, then for some i, $s_i = x^*$.
- By construction, $x^*[i] = 1 - s_i[i] \neq s_i[i]$, so $x^* \neq s_i$.
- This contradicts the original assumption that it was ***possible*** to list the elements of S. Hence S is not countable.

s_1	0	1	1	0	1	0	...
s_2	1	0	0	0	0	0	...
s_3	0	1	1	1	0	1	...
s_4	0	0	0	0	0	0	...
s_5	1	1	1	1	1	1	...
...	...	...	...	...	...	...	...

x^*	1	1	0	1	0	...	

First Example:

Set of *infinite-length* binary sequences is uncountable

- **Conclusion:** The set of all *infinite* binary sequences is uncountable.
- **Diagonalization Summary:** For any *candidate* list of such sequences, there is a sequence *not* in that list.

Cantor's Theorem

Cantor's Theorem:
For any set X,
$|X| < |P(X)|$.

Proving Undecidability via Diagonalization

Plan: How to show the existence of undecidable language

- **Recall:**
- Let $\mathcal{L}$ be the set of all languages.
- Let $\mathcal{M}$ be the set of all TMs.
- Will show that $|\mathcal{M}| < |\mathcal{L}|$.

- **Q1:** Is $\mathcal{M}$ countable?
- **A:** Yes. (We'll see.)
- **Q2:** Is $\mathcal{L}$ countable?
- **A:** No! (We'll see.)

The set of TMs is countable

- **Claim:** The set of Turing Machines is *countable*.
- **Idea:** Use lexicographical ordering on source code.
  (Every TM has an encoding as a finite-length string!)

易见. $\{0,1\}^* = \{0, 1, 00, 01, \ldots\}$
1 2 3 4
(可列)

或者: 7-tuple是finite的, 因而我们可以用一个binary string去encode它, 因而 $M \leq \{0,1\}^*$

"bool **A**(string x): **return** F"
"bool **A**(string x): **return** T"
"bool **A**(string x): for i=1… x: …"
"bool **A**(string x): let x=x-1 …"

而 {all binary strings} $= \{0,1\}^*$ 也是countable的 $\Rightarrow M$ ctb

The set of Languages is uncountable

$L \subseteq \Sigma^*$ 是 infinite

- **Claim:** Any language L can be represented by an *infinite* binary sequence.
- **Idea:** List all input strings $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$
  - Then L: $x_1 x_2 x_3 \cdots$, where $x_i = 1$ if $s_i \in L$ and 0 otherwise.
- **Example:** Suppose $\Sigma = \{0,1\}$.
  - $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$
  - L: 1 0 0 1 1 1 1 0 0 ⋯

Language L decided by
"bool **M**(string x): **return** (|x| is even)"

Conclude:
There exists an undecidable language

- The set $\mathcal{M}$ of all TMs is countable, $|\mathcal{M}| \leq |\mathbb{N}|$
- The set $\mathcal{L}$ of all languages is uncountable, $|\mathcal{L}| > |\mathbb{N}|$.

- So $|\mathcal{M}| < |\mathcal{L}|$.
- Each TM decides at most one decidable language.
- So there exists an undeciable language.

- Next: Can also prove this directly using a diagonalization argument too.

Another way to show:
There exists an undecidable language

- Construct a table T representing all decidable languages.
  - Columns: list all input strings $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$
  - Rows: list all TMs $\{M_1, M_2, M_3, \ldots\}$
  - $T[i,j] = 1$ iff machine M_i accepts string s_j, 0 otherwise.
- **Claim:** No TM decides the language represented by L^*.

$$L^*[j] = 1 - T[j,j]$$

	s_1	s_2	s_3	s_4	s_5	s_6	…
$L(M_1)$	1	0	0	1	1	0	…
$L(M_2)$	0	1	1	0	0	0	…
$L(M_3)$	1	1	1	1	1	1	…
$L(M_4)$	0	0	0	0	0	0	…
$L(M_5)$	1	0	1	0	0	0	…
…	…	…	…	…	…	…	…

- **Proof.** If L^* is decidable then $L^* = L(M_i)$ for some i. But $s_i \in L(M_i) \Leftrightarrow s_i \notin L^*$, so $L^* \neq L(M_i)$. Contradiction.

L^*	0	0	0	1	1	…

Conclusion

- **Theorem:** There ***exists*** an undecidable language L^*.
- **Interpretation:** There is a "problem" that no computer program can solve correctly (on all inputs).

- **Question:** What problem does L^* represent? Do we care about it? Would it be useful to solve?
- **Answer:** We do not know, since the proof is 'non-constructive': only shows *existence* of L^*!
- **Next time:** Some **explicit** undecidable languages.