Decidability

Turing Machines Coded as Binary Strings
Diagonalizing over Turing Machines
Problems as Languages
Undecidable Problems

Integers, Strings, and Other Things

◆ Data types have become very important as a programming tool.
◆ But at another level, there is only one type, which you may think of as integers or strings.
◆ Key point: Strings that are programs are just another way to think about the same one data type.

Example: Text
◆ Strings of ASCII or Unicode characters can be thought of as binary strings, with 8 or 16 bits/character.
◆ Binary strings can be thought of as integers.
◆ It makes sense to talk about “the i-th string.”

Binary Strings to Integers
◆ There’s a small glitch:
  ▷ If you think simply of binary integers, then strings like 101, 0101, 00101,… all appear to be “the fifth string.”
◆ Fix by prepending a “1” to the string before converting to an integer.
  ▷ Thus, 101, 0101, and 00101 are the 13th, 21st, and 37th strings, respectively.
  ▷ Also handles empty string

Example: Images
◆ Represent an image in (say) GIF.
◆ The GIF file is an ASCII string.
◆ Convert string to binary.
◆ Convert binary string to integer.
◆ Now we have a notion of “the i-th image.”

Example: Proofs
◆ A formal proof is a sequence of logical expressions, each of which follows from the ones before it.
◆ Encode mathematical expressions of any kind in Unicode.
◆ Convert expression to a binary string and then an integer.
Proofs – (2)

- But a proof is a sequence of expressions, so we need a way to separate them.
- Also, we need to indicate which expressions are given.

Proofs – (3)

- Quick-and-dirty way to introduce new symbols into binary strings:
  1. Given a binary string, precede each bit by 0.
     - Example: 101 becomes 01001.
  2. Use strings of two or more 1's as the special symbols.
     - Example: 111 = "the following expression is given"; 11 = "end of expression."

Example: Encoding Proofs

010001111110000101110101

A given expression follows
Expression
End

Example: Programs

- Programs are just another kind of data.
- Represent a program in ASCII.
- Convert to a binary string, then to an integer.
- Thus, it makes sense to talk about "the i-th program."
- Hmm...There aren't all that many programs.

Finite Sets

- Intuitively, a finite set is a set for which there is a particular integer that is the count of the number of members.
- Example: \{a, b, c\} is a finite set; its cardinality is 3.
- It is impossible to find a 1-1 mapping between a finite set and a proper subset of itself.

Infinite Sets

- Formally, an infinite set is a set for which there is a 1-1 correspondence between itself and a proper subset of itself.
- Example: the positive integers \{1, 2, 3,...\} is an infinite set.
  - There is a 1-1 correspondence 1<->2, 2<->4, 3<->6,... between this set and a proper subset (the set of even integers).
Countable Sets

- A countable set is a set with a 1-1 correspondence with the positive integers.
  - Hence, all countable sets are infinite.
  - **Example**: All integers.
    - $0 \leftrightarrow 1; -i \leftrightarrow 2i; +i \leftrightarrow 2i+1$.
    - Thus, order is $0, -1, 1, -2, 2, -3, 3, \ldots$
  - **Examples**: set of binary strings, set of Java programs.

Example: Pairs of Integers

- Order the pairs of positive integers first by sum, then by first component:
  - $[1,1], [2,1], [1,2], [3,1], [2,2], [1,3], [4,1], [3,2], \ldots, [1,4], [5,1], \ldots$
- **Interesting exercise**: figure out the function $f(i,j)$ such that the pair $[i,j]$ corresponds to the integer $f(i,j)$ in this order.

Enumerations

- An enumeration of a set is a 1-1 correspondence between the set and the positive integers.
- Thus, we have seen enumerations for strings, programs, proofs, and pairs of integers.

How Many Languages?

- Are the languages over $\{0,1\}^*$ countable?
  - No; here's a proof.
- Suppose we could enumerate all languages over $\{0,1\}^*$ and talk about “the $i$-th language.”
- Consider the language $L = \{ w \mid w$ is the $i$-th binary string and $w$ is not in the $i$-th language.$\}$

Proof – Continued

- Clearly, $L$ is a language over $\{0,1\}^*$.
- Thus, it is the $j$-th language for some particular $j$.
- Let $x$ be the $j$-th string.
- Is $x$ in $L$?
  - If so, $x$ is not in $L$ by definition of $L$.
  - If not, then $x$ is in $L$ by definition of $L$.

Diagonalization Picture

**Strings**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>...</td>
</tr>
</tbody>
</table>

**Languages**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Proof – Concluded

- We have a contradiction: \( x \) is neither in \( L \) nor not in \( L \), so our sole assumption (that there was an enumeration of the languages) is wrong.
- Comment: This is really bad; there are more languages than programs.
- E.g., there are languages with no membership algorithm.

Non-Constructive Arguments

- We have shown the existence of a language with no algorithm to test for membership, but we have no way to exhibit a particular language with that property.
- A proof by counting the things that fail and claiming they are fewer than all things is called a Hungarian argument.

Binary-Strings from TM’s

- We shall restrict ourselves to TM’s with input alphabet \( \{0, 1\} \).
- Assign positive integers to the three classes of elements involved in moves:
  1. States: \( q_1 \) (start state), \( q_2 \) (final state), \( q_3 \), …
  2. Symbols \( X_1 \) (0), \( X_2 \) (1), \( X_3 \) (blank), \( X_4 \), …
  3. Directions \( D_1 \) (L) and \( D_2 \) (R).

Binary Strings from TM’s – (2)

- Suppose \( \delta(q_i, X_j) = (q_k, X_l, D_m) \).
- Represent this rule by string \( 0^i10^j10^k10^l10^m \).
- Key point: since integers \( i, j, \ldots \) are all > 0, there cannot be two consecutive 1’s in these strings.

Binary Strings from TM’s – (2)

- Represent a TM by concatenating the codes for each of its moves, separated by 11 as punctuation.
  - That is: Code_11Code_11Code_11 ...
Enumerating TM’s and Binary Strings

- Recall we can convert binary strings to integers by prepending a 1 and treating the resulting string as a base-2 integer.
- Thus, it makes sense to talk about “the i-th binary string” and about “the i-th Turing machine.”
- Note: if i makes no sense as a TM, assume the i-th TM accepts nothing.

Table of Acceptance

<table>
<thead>
<tr>
<th>String j</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>TM 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TM 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TM 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TM 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TM 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TM 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( x = 0 \) means the i-th TM does not accept the j-th string; 1 means it does.

Diagonalization Again

- Whenever we have a table like the one on the previous slide, we can diagonalize it.
  - That is, construct a sequence D by complementing each bit along the major diagonal.
  - Formally, \( D = a_1a_2... \), where \( a_i = 0 \) if the \((i, i)\) table entry is 1, and vice-versa.

The Diagonalization Argument

- Could D be a row (representing the language accepted by a TM) of the table?
  - Suppose it were the j-th row.
  - But D disagrees with the j-th row at the j-th column.
  - Thus D is not a row.

Diagonalization – (2)

- Consider the diagonalization language \( L_d = \{ w \mid w \) is the i-th string, and the i-th TM does not accept w\}.
- We have shown that \( L_d \) is not a recursively enumerable language; i.e., it has no TM.

Problems

- Informally, a “problem” is a yes/no question about an infinite set of possible instances.
- Example: “Does graph G have a Hamilton cycle (cycle that touches each node exactly once)?”
  - Each undirected graph is an instance of the “Hamilton-cycle problem.”
Problems – (2)

Formally, a problem is a language.
Each string encodes some instance.
The string is in the language if and only if the answer to this instance of the problem is “yes.”

Example: A Problem About Turing Machines

We can think of the language $L_d$ as a problem.
“Does this TM not accept its own code?”
Aside: We could also think of it as a problem about binary strings.
Do you see how to phrase it?

Decidable Problems

A problem is **decidable** if there is an algorithm to answer it.

- Recall: An “algorithm,” formally, is a TM that halts on all inputs, accepted or not.
- Put another way, “decidable problem” = “recursive language.”
- Otherwise, the problem is **undecidable**.

Bullseye Picture

Decidable problems = Recursive languages
Recursively enumerable languages
Not recursively enumerable languages

From the Abstract to the Real

While the fact that $L_d$ is undecidable is interesting intellectually, it doesn’t impact the real world directly.
We first shall develop some TM-related problems that are undecidable, but our goal is to use the theory to show some real problems are undecidable.

Examples: Undecidable Problems

- Can a particular line of code in a program ever be executed?
- Is a given context-free grammar ambiguous?
- Do two given CFG’s generate the same language?
The Universal Language

- An example of a recursively enumerable, but not recursive language is the language $L_u$ of a universal Turing machine.
- That is, the UTM takes as input the code for some TM $M$ and some binary string $w$ and accepts if and only if $M$ accepts $w$.

Designing the UTM

- Inputs are of the form: Code for $M$ 111 $w$
- Note: A valid TM code never has 111, so we can split $M$ from $w$.
- The UTM must accept its input if and only if $M$ is a valid TM code and that TM accepts $w$.

The UTM – (2)

- The UTM will have several tapes.
  - Tape 1 holds the input $M$ 111 $w$
  - Tape 2 holds the tape of $M$.
    - Mark the current head position of $M$.
  - Tape 3 holds the state of $M$.

The UTM – (3)

- Step 1: The UTM checks that $M$ is a valid code for a TM.
  - E.g., all moves have five components, no two moves have the same state/symbol as first two components.
  - If $M$ is not valid, its language is empty, so the UTM immediately halts without accepting.

The UTM – (4)

- Step 2: The UTM examines $M$ to see how many of its own tape squares it needs to represent one symbol of $M$.
- Step 3: Initialize Tape 2 to represent the tape of $M$ with input $w$, and initialize Tape 3 to hold the start state.

The UTM – (5)

- Step 4: Simulate $M$.
  - Look for a move on Tape 1 that matches the state on Tape 3 and the tape symbol under the head on Tape 2.
  - If found, change the symbol and move the head marker on Tape 2 and change the State on Tape 3.
  - If $M$ accepts, the UTM also accepts.
A Question

- Do we see anything like universal Turing machines in real life?

**Proof** That $L_u$ is Recursively Enumerable, but not Recursive

- We designed a TM for $L_u$, so it is surely RE.
- Suppose it were recursive; that is, we could design a UTM $U$ that always halted.
- Then we could also design an algorithm for $L_u$, as follows.

**Proof** – (2)

- Given input $w$, we can decide if it is in $L_u$ by the following steps.
  1. Check that $w$ is a valid TM code.
     - If not, then its language is empty, so $w$ is in $L_u$.
  2. If valid, use the hypothetical algorithm to decide whether $w111w$ is in $L_u$.
  3. If so, then $w$ is not in $L_u$; else it is.

**Proof** – (3)

- But we already know there is no algorithm for $L_u$.
- Thus, our assumption that there was an algorithm for $L_u$ is wrong.
- $L_u$ is RE, but not recursive.

Bullseye Picture

- Decidable problems = Recursive languages
- All these are decidable
- Not recursively enumerable languages
- Recursively enumerable languages
- $L_d$
- $L_u$