Computable Functions
Part I: Non-Computable Functions
Computable and Partially Computable Functions

Computable Function

• Exists a Turing Machine $M$
  -- $M$ Halts on All Input
  -- $M(x) = f(x)$

Partially Computable Function

• Exists a Turing Machine $M$
  -- $M$ Does Not Halt on All Input
  -- If $M$ Halts on $x$, then $M(x) = f(x)$
Examples

Computable

• $f(n) = n + 1$

• Simple Turing Machine in Unary

Partially Computable

• $steps(<M, w>) = \# \text{ operations performed by } M \text{ on } w \text{ before } M \text{ Halts}$

• Simple Turing Machine

  -- Run $M$ on $w$

  -- Count Operations

• Not Computable

  -- Otherwise Could Solve Halting Problem
Non Partially Computable Functions

# Turing Machines is Countable

• # Turing Machines with $N$ States = # Transition Functions

• Transition Functions = 5-tuples $(State, Symbol, State, Symbol, L/R)$

• # Turing Machines with $N$ States = $2 \mid Q \mid ^2 \mid \Sigma \mid ^2$

• Countable Union of Countable Sets is Countable

# Functions $N \rightarrow [0,1]$ is Uncountable

• # Functions $N \rightarrow [0,1] = # Subsets of N$

• # Subset of $N = P(N)$ -- Uncountable

• Most Functions are Not Partially Computable
Busy Beaver Problems

Problems

- \( S(n) = \) Maximum number of operations a Turing Machine with \( n \) states can perform on a Blank tape and then Halt

- \( \Sigma(n) = \) Maximum number of 1’s that a Turing Machine with \( n \) states can write on a Blank tape and then Halt.
Theorem 1:  There is no Turing Machine $B$ that computes $S(n)$.

Proof: Suppose such a Turing Machine $B$ did exist.

Let $M$ be a Turing Machine with $n$ States.

To Determine if $M$ Halts on $\epsilon$, Build the following Turing Machine:

- Using $B$, Compute $S(n)$
- Run $M$ on $\epsilon$
- If $M$ Halts before $S(n)+1$ Steps, Accept
- Otherwise Reject

This Turing Machine Decides $H_\epsilon = \{ <M> | M \text{ Halts on } \epsilon \}$

But $H_\epsilon$ is Undecidable. Hence $B$ Cannot Exist.
Theorem 2: There is no Turing Machine $B$ that computes $\Sigma(n)$ (in binary).

Proof: Suppose such a machine $B$ did exist.

Let $B_n$ be the Turing Machine with $n$ states that starts with a blank tape, writes $\Sigma(n)$ 1’s, and Halts. Now define two new machines:

- $A$ -- writes $n$ in binary on a blank tape
- $C$ -- converts binary to unary

Then

- $|A| \approx \log(n)$ (print 1, move to right, go to next state)
- $|C| = \text{constant}$ (there is an algorithm for converting binary to unary)
- $|B| = \text{constant}$

So the machine $ABC$ takes a blank tape and writes $n$ 1’s on the tape. But for large $n$, we have $|ABC| < |B_n| = n$ Contradiction.

Hence $B$ cannot exist.
**Busy Beaver**

*Values*

- \( B(1) = 1 \)
- \( B(2) = 4 \)
- \( B(3) = 6 \)
- \( B(4) = 13 \)
- \( B(5) \geq 4098 \)

*Observation*

- \( B(n) \neq O(f) \) for any computable function \( f \)
- \( f(B(n)) < B(n+1) \) for all computable \( f \) for arbitrary many values of \( n \)
Busy Beaver and the Halting Problem

Algorithm for Busy Beaver

1. Build all Turing machines with \( n \) states
   -- finite number of lists of 5-tuples
2. Run each machine on a blank tape -- universal Turing machine
3. Take the maximum value of 1’s

Observation

- Algorithm Fails because some Turing machines do not Halt
- Algorithm would work only if we could solve the Halting Problem
Part II: Self Description and Recursion
Theorem: \( \textit{Min} \) is Not Semi-Decidable.

Proof: \( \textit{Min} \) is Semi-Decidable \( \Rightarrow \) Can Enumerate \( \textit{Min} \)

To show \( \textit{Min} \) is not Semi-Decidable:

First build \( M^\# \)

To Run \( M^\# \) on any String \( w \)

1. Construct a Description \(< M^\# >\) of \( M^\# \) (Chapter 25)
2. Find a Turing Machine \( M' \) in the List for \( \textit{Min} \) with \( |< M' >| > |< M^\# >| \).
3. Compute \( M'(w) \).

Then

4. \( M' \) in the List for \( \textit{Min} \) \( \Rightarrow \) \( M' \) is a Minimal Machine
5. \( M^\#(w) = M'(w) \) \( \Rightarrow \) \( M^\# \) is equivalent to \( M' \)
6. \( |< M^\# >| < |< M' >| \) \( \Rightarrow \) \( M' \) NOT a Minimal Machine  CONTRADICTION
Self Description and Recursion

Problem

Construct a Turing Machine $M$ that:

1. Writes a Description of $M$
2. Performs some Operations $W$

Observations

1. We need to know $M$ to write a Description of $M$
2. But ... the Definition of $M$, Depends on $M$!
Attempts at Self Description

Construction of $M$ -- First Try
1. Write $<W>$
2. Perform $W$

No! This Machine Describes $W$, not $M$

Construction of $M$ -- Second Try
1. Write $\langle <W>, W \rangle$
2. Perform $W$

No! This Machine Describes the Machine that
   Writes a Description of the Machine that
   Writes a Description of $W$ and Performs $W$.

Does NOT Describe $M$
Solution: Part I

The Turing Machine B

• On Tape #2, Write a Description of the Turing Machine A that Writes the Symbols Already on Tape #1.
  -- Tape #1: \( s_1 s_2 \cdots s_n \)
  -- Tape #2: \( s_1 R s_2 R \cdots s_n R \)

• Copy the Contents of Tape #2 in Front of the Contents of Tape #1.
  -- Tape #1: \( s_1 R s_2 R \cdots s_n R s_1 \cdots s_n \)

• Observations
  -- The Output of B Depends on the Content of Tape #1
  -- The Program for B is Independent of the Contents of Tape #1
  -- The Description of B is Independent of the Contents of Tape #1.
Solution: Part II

The Turing Machine A

- Write $<B,W>$ on Tape #1
- Observations
  - The Program for $A$ Depends Only on $W$; $B$ is Fixed
  - The Description of $A$ Depends Only on $W$.

The Solution

- Write $<A>, <B>, <W>$
- Perform $W$
Analysis of Solution

Analysis of A, B, W

• A: Writes <B>, <W>
• B: Writes <A> in front of <B>, <W> ⇒ Writes <A>, <B>, <W>
• W: Perform W.

Conclusion

• A, B, W Writes a Description of A, B, W
• Performs W
Virus

Virus Program

• Write a Description of the Virus Program

• For Each Address in the Address Book

  -- Mail a Description of the Virus Program to the Address

  -- Perform Some Malicious Operations
The Recursion Theorem

Let $T$ be a Turing Machine that Computes a Partially Computable Function

$$t(a,b) = T(a,b).$$

Then there is a Turing Machine $R$ that Computes a Partially Computable Function

$$r(x) = T(<R>,x).$$

Proof: Description of $R(x)$:

- Write a Description $<R>$ of $R$
- Perform $r(x) = T(<R>,x)$
The Fixed Point Theorem

Let

• \( S = \{ < M > \mid M \text{ is a Turing Machine} \} \)

• \( f : S \to S \) be a Computable Function

Then there exists a Turing Machine \( F \) such that

• \( f(< F >) = < G > \)

• \( F \Leftrightarrow G \) \((F \text{ and } G \text{ Behave the Same on All Inputs})\)

Proof: Description of \( F(x) \):

• Write \( < F > = \) a Description of \( F \)

• Compute \( < G > = M_f ( < F > ) = f(< F >) \)
  -- \( M_f = \) Turing Machine that Computes \( f \)

• Run \( G \) on \( x \)
# Symbols for Encoding Turing Machines

<table>
<thead>
<tr>
<th>Eleven Symbols</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>Non-Halting State</td>
</tr>
<tr>
<td>$y$</td>
<td>Accepting State</td>
</tr>
<tr>
<td>$n$</td>
<td>Rejecting State</td>
</tr>
<tr>
<td>$a$</td>
<td>String</td>
</tr>
<tr>
<td>$L$</td>
<td>Move Left</td>
</tr>
<tr>
<td>$R$</td>
<td>Move Right</td>
</tr>
<tr>
<td>(</td>
<td>Grouping for Transition Functions</td>
</tr>
<tr>
<td>)</td>
<td>Grouping for Transition Functions</td>
</tr>
<tr>
<td>,</td>
<td>Separator for Transition Functions</td>
</tr>
<tr>
<td>0</td>
<td>Symbol for Encoding States and String</td>
</tr>
<tr>
<td>1</td>
<td>Symbol for Encoding States and Strings</td>
</tr>
</tbody>
</table>
## Godel Numbering for Special Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Godel Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>0000</td>
</tr>
<tr>
<td>$y$</td>
<td>0001</td>
</tr>
<tr>
<td>$n$</td>
<td>0010</td>
</tr>
<tr>
<td>$a$</td>
<td>0011</td>
</tr>
<tr>
<td>$L$</td>
<td>0100</td>
</tr>
<tr>
<td>$R$</td>
<td>0101</td>
</tr>
<tr>
<td>(</td>
<td>0110</td>
</tr>
<tr>
<td>)</td>
<td>0111</td>
</tr>
<tr>
<td>:</td>
<td>1000</td>
</tr>
</tbody>
</table>
Godel Numbering for Turing Machines

Godel Function

- \textit{Godel : Turing Machine} \rightarrow N
  
  -- \textless M \textgreater = \text{Encoding of } M = \text{Encoding of Transition Functions}
  
  -- \textless M \textgreater = \text{Long Binary String (Use Godel Numbering for Special Symbols)}

- \textit{Godel(M)} = \text{Number Represented by Long Binary String Encoding} \textless M \textgreater

- \textless M \textgreater \neq \textless N \textgreater \Rightarrow \textit{Godel(M)} \neq \textit{Godel(N)}
Godel Numbering for Partially Computable Functions

Godel Function

- \textit{Godel} : \textit{Partially Computable Functions} → \textit{N}
  
  -- \textit{F} = Partially Computable Function

  -- \textit{M}_F = Turing Machine with Lowest Godel Number that Computes \textit{F}

- \textit{Godel}(\textit{F}) = \textit{Godel}(\textit{M}_F)

- \phi_k = Partially Computable Function with Godel Number \textit{k}
The s–m–n Theorem

Let \( k = \text{Godel Number of a Partially Computable Function with } m+n \text{ arguments.} \)

Then there Exists a Computable Function \( s_{m,n} \) such that

1. \( j = s_{m,n}(k,u_1,\ldots,u_n) \) is the Godel Number of a Partially Computable Function

2. \( \phi_j(y_1,\ldots,y_n) = \phi_k(u_1,\ldots,u_n,y_1,\ldots,y_n) \)

Proof: Define Turing Machine \( M_{m,n} \) to Compute \( s_{m,n} \):

\[ M_{m,n}(k,u_1,\ldots,u_m) : \]

- Construct \( M_j(w) \):
  -- Write \( u_1,\ldots,u_m \) to the Left of \( w \)
  -- Move Head to Left of \( u_1 \)
  -- Apply \( \phi_k \)
- Return \( j \)