What Is Computability Theory?

Rebecca Weber, Dartmouth College
WIM at MIT, March 3, 2009
Computability

We call a function computable if there is a computer program that executes it.

What are the limits of computational power? We need to abstract the essentials.

Also:

- Want to be independent of hardware advances.
- Don’t want to set limits in advance on time and memory use.
Essential Components

- memory that can be read from and written to
- arithmetic
- if...then
- looping (for, while)

More than this list is purely to make it easier for humans to use.
BASIC

10 REM LANDING
20 REM A flying saucer coming in for a landing.
30 FOR FREQ% = 600 TO 50 STEP -25
40    SOUND FREQ%, 2
50    SOUND 32767, .5
60 NEXT FREQ%
70 END
```cpp
#include <iostream>
using namespace std;

int main () {
    for (int n=10; n>0; n--) {
        cout << n << " , " ;
    }
    cout << "FIRE!\n" ;
    return 0 ;
}
```
FALSE
(factorial program)

[$1=~[$1-f;!*]?]f:
"calculate the factorial of [1..8]:"
β^β’0-$$0>~\8>|$
"result:"
~[\f;!].?
["illegal input!"]?"
"
Commonalities

- Finite sequence of symbols out of a finite alphabet (usually letters, numbers, and standard punctuation).

- Could be ordered somehow and each assigned a number.

Fix some programming language.
**Enumeration (Listing)**

Sequences on alphabet \{a,b\} may be enumerated as follows:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>6</td>
<td>bb</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>7</td>
<td>aaa</td>
</tr>
<tr>
<td>3</td>
<td>aa</td>
<td>8</td>
<td>aab</td>
</tr>
<tr>
<td>4</td>
<td>ab</td>
<td>9</td>
<td>aba</td>
</tr>
<tr>
<td>5</td>
<td>ba</td>
<td>10</td>
<td>abb</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>baa</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>bab</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>bba</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>bbb</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Presumably not all will be valid programs, but that’s okay. We’ll just consider those to be programs that do nothing.
A Counting Argument

- There are as many programs in a given language as there are natural numbers (*countably many*).
- There are as many functions on the natural numbers as there are real numbers (*uncountably many*).

The latter set is strictly larger!

In fact, there are uncountably many noncomputable functions.
The Halting Problem

Fix an enumeration of programs $P_0, P_1, \ldots$. We’ll define a function $f$ that is not computed by any program on the list.

$$f(x) = \begin{cases} 
1 & \text{if } P_x(x) \text{ halts (gives an output)} \\
0 & \text{if } P_x(x) \text{ goes into an infinite loop}
\end{cases}$$

Proof by contradiction: from $f$, define $g$:

$$g(x) = \begin{cases} 
P_x(x) + 1 & \text{if } f(x) = 1 \\
0 & \text{if } f(x) = 0
\end{cases}$$

If we can write $f$ as a program we can also write $g$ as a program, meaning $g = P_e$ for some $e$. But then if $P_e(e)$ is defined, $g(e) = P_e(e) + 1 \neq P_e(e)$, and if $P_e(e)$ is not defined, $g(e) = 0$, again $\neq P_e(e)$. 
Not Just Functions

It is useful to work in terms of subsets of the natural numbers for our continuing exploration.

Notation:

- $\mathbb{N} = \{0, 1, 2, \ldots\}$, the natural numbers.
- $x \in A$ is read “$x$ is in $A$” or “$x$ is a member of $A$” or “$x$ is an element of $A$”, where $A$ is a set.
- $A \subseteq B$ is read “$A$ is a subset of $B$” and means all elements of $A$ are also elements of $B$; $B$ may or may not have additional elements not in $A$. 
Sets, Sequences, Functions

We are very loose with these objects and blur them together.

- The function \( f : \mathbb{N} \rightarrow \{0, 1\} \) is associated with the sequence with entries \( f(0), f(1), f(2), \ldots \), in order.

- The sequence \( S \) is associated with the set \( A \) where \( n \in A \) if the \( n^{th} \) entry of \( S \) is 1, and \( n \notin A \) otherwise.

Example

Define \( f \) by \( f(n) = \) the remainder of \( n \) upon division by 2. This function is associated with the sequence 010101010101\ldots and the set of odd numbers.
Comparing Noncomputability

If we choose some set $A$ and allow our programs to include statements of the form “if $n \in A$, then...”, we are working with oracle programs. If $A$ is noncomputable, we can now compute more sets than we could before (e.g., $A$ itself).

[If $A$ is computable we’ve added nothing. Why?]

Notation: if $B$ can be computed by a program with oracle $A$, we say $B$ is Turing reducible to $A$ and write $B \leq_T A$. 
It Goes Up and Up

Call the set associated with the Halting Problem $H$, and give it to every program in our enumeration as an oracle (the list is now written $P^H_0, P^H_1, \ldots$). Define a new $f$:

$$f(x) = \begin{cases} 1 & \text{if } P^H_x(x) \text{ halts (gives an output)} \\ 0 & \text{if } P^H_x(x) \text{ goes into an infinite loop} \end{cases}$$

*The same proof as before* shows $f$ is not computable by any $P^H_e$, and this proof is not dependent on $H$. No matter what oracle $A$ we choose, there are sets $B \not\leq_T A$ — in fact, uncountably many.

This $f$ is the Halting Problem *relativized* to $H$. 
Turing Degrees

If we call the set associated with our new halting function $H'$, we have $H \subseteq_T H'$. Iterating we can get $H' \subseteq_T H'' \subseteq_T H''' \subseteq_T \ldots$ forever, because we always have uncountably many sets left.

The relation $A \equiv_T B$ defined as $A \leq_T B \& B \leq_T A$ partitions the subsets of $\mathbb{N}$ into boxes (equivalence classes) called Turing degrees.

Each Turing degree contains countably many sets.

There are uncountably many Turing degrees.
Computably Enumerable Sets

A natural collection of sets to consider to be “next-larger” than the computable sets is the *computably enumerable* (c.e.) sets.

A is c.e. if its elements may be listed out computably, but not necessarily in order.

Each program $P_e$ is associated with two c.e. sets: its domain and its range. When taken for all programs, either one covers all the c.e. sets, and traditionally we use the domain.

We denote $\text{dom}(P_e)$ by $W_e$, and call it the $e^{th}$ c.e. set.
Facts About C.E. Sets

- A set is computable if and only if its elements may be enumerated \textit{in order}.

- A set $A$ is computable if and only if both $A$ and $\bar{A}$ are c.e.
  
  ($\bar{A} = \text{complement of } A = \text{everything in } \mathbb{N} \text{ but not } A$)

- All c.e. sets are Turing reducible to the Halting Set.

- The Halting Set is c.e. itself.

- There are non-c.e. sets $A$ such that $A \leq_T H$ as well.
Things We Study I
LUB and GLB for Degrees

We say \( \deg(A) \leq \deg(B) \) if \( A \leq_T B \). This makes the degrees a partially ordered set that we can study.

- Every pair of degrees has a least upper bound.
- *Not* every pair of degrees has a greatest lower bound.
- For some but not all \( A \) there is \( B \) so that \( \text{glb} (\deg(A), \deg(B)) \) exists and equals \( \deg(\emptyset) \).
- For some but not all \( A \) there is \( B \) so that \( \text{lub}(\deg(A), \deg(B)) = \deg(H) \).
If we relativize the halting set to a computable set, its Turing degree remains $\deg(H)$.

- If $A \leq_T B$, $H^A \leq_T H^B$, but $\prec_T$ can change to $\equiv_T$.

- There are noncomputable sets $A$ such that $H^A \equiv_T H$ ($A$ is called low).

- There are sets $A \prec_T H$ such that $H^A \equiv_T H'$ ($A$ is called high).
Tools We Use (I and Only)
Priority Constructions

(A very sketchy look at constructing a noncomputable low set $A$.)

<table>
<thead>
<tr>
<th>Goal</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$ is c.e.</td>
<td>Computable construction that puts things in $A$ and never takes them out</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>$\overline{A}$ is not c.e.</td>
<td>Make every infinite $W_e$ intersect $A$</td>
</tr>
</tbody>
</table>

$H^A \equiv_T H$
Keep $A$ and hence $H^A$ from “changing too much” during construction
Conflicts and Resolutions I

On the one hand, we want to put things into $A$ when we see the opportunity to make $A$ intersect $W_e$. On the other hand, putting things into $A$ might change $H^A$.

Give a *priority ordering* to construction requirements:

- **Pos0**: Make $W_0$ intersect $A$
- **Neg0**: Keep $H^A$’s value on 0 constant
- **Pos1**: Make $W_1$ intersect $A$
- **Neg1**: Keep $H^A$’s value on 1 constant

… and so forth.
Conflicts and Resolutions II

▶ The Neg requirements forbid enumerating certain elements into $A$ (set restraint on $A$): namely, Neg22 restrains the elements that tell us whether 22 is in $H^A$ or not.

▶ If Neg22 says “nothing below 140 can enter $A$”, Pos23, Pos24 and beyond must obey it. Pos22, Pos21 and up can ignore it.

▶ If $W_{23}$ is infinite, Pos23 will still find a number in $W_{23}$ it’s allowed to put in $A$. If $W_{23}$ is finite we don’t care about it.

▶ Each Pos requirement puts at most one number in $A$, so Neg can make sure its value of $H^A$ changes only finitely often.
In a Nutshell

Priority arguments allow us to cope with information that is being given gradually and may be incomplete or even incorrect during the course of the construction.

▶ We may act wrongly, but not acting might be just as wrong.

If we set things up so errors can be overcome, we can keep our construction at a known level of computability and hence make assertions about the computability of the set we’re constructing.
Takehome Message

Computable functions are great, but noncomputable ones are more interesting.

THANK YOU!