What does this do?

```c
(...,____+1,____):...<1?(...,____+1,...
(...,____+1,0):...%==____/____&&!____?(printf("%d\t",____/____),(...,____+1,0)):...%>1&&...%<____/?(...,____+1,...+1/%(____%))<____*?(...,____+1,____):0;main(){(100,0,0);}
```

Turing's Legacy: The Limits Of Computation
Lecture 24 (November 11, 2010)

From the last lecture:

Are all reals describable? NO
Are all reals computable? NO

We saw that computable \(\Rightarrow\) describable
but do we also have describable \(\Rightarrow\) computable?

We'll answer this question today…

Theorem: S can't be put into bijection with \(P(S)\)

Suppose \(f:S\rightarrow P(S)\) is a bijection.
Let \(CONFUSE_f = \{ x | x \in S, x \notin f(x) \}\)
Since f is onto, exists \(y \in S\) such that \(f(y) = CONFUSE_f\).
Is y in \(CONFUSE_f\)?
Let \( S \) be a set such that \( f(S) = \text{CONFUSE} \).

Is \( y \) in \( S \)?

**Theorem:**

A set \( S \) can’t be put into bijection with its power set \( P(S) \).

**Computable Function**

Fix a finite set of symbols, \( \Sigma \).

Fix a precise programming language, e.g., Java.

A program is any finite string of characters that is syntactically valid.

A function \( f : \Sigma^* \rightarrow \Sigma^* \) is computable if there is a program \( P \) that when executed on an ideal computer, computes \( f \).

That is, for all strings \( x \) in \( \Sigma^* \), \( f(x) = P(x) \).

Hence: countably many computable functions!

**Uncountably Many Functions**

The functions \( f : \Sigma^* \rightarrow \{0,1\} \) are in 1-1 onto correspondence with the subsets of \( \Sigma^* \) (the powerset of \( \Sigma^* \)).

**Uncountably many functions from \( \Sigma^* \) to \( \{0,1\} \).**

Thus, most functions from \( \Sigma^* \) to \( \{0,1\} \) are not computable.
Can we explicitly describe an uncomputable function?

The HELLO assignment

Write a JAVA program to output the words “HELLO WORLD” on the screen and halt.

Space and time are not an issue.
The program is for an ideal computer.

PASS for any working HELLO program, no partial credit.

Grading Script

The grading script G must be able to take any Java program P and grade it.

\[
G(P) = \begin{cases} 
\text{Pass, if } P \text{ prints only the words "HELLO WORLD" and halts.} \\
\text{Fail, otherwise.}
\end{cases}
\]

How exactly might such a script work?

What does this do?

\[
\text{main}() \{ \text{printf}("\text{\text{
Hello}}\n") \}
\]

Nasty Program

\begin{verbatim}
n:=0; while (n is not a counter-example to the Riemann Hypothesis) { n++; } print "Hello";
\end{verbatim}

The nasty program is a PASS if and only if the Riemann Hypothesis is false.

A TA nightmare: Despite the simplicity of the HELLO assignment, there is no program to correctly grade it!

And we will prove this.
The theory of what can and can’t be computed by an ideal computer is called Computability Theory or Recursion Theory.

Notation And Conventions
Fix a single programming language (Java)
When we write program P we are talking about the text of the source code for P
P(x) means the output that arises from running program P on input x, assuming that P eventually halts.
P(x) = ⊥ means P did not halt on x

The meaning of P(P)
It follows from our conventions that P(P) means the output obtained when we run P on the text of its own source code

The Halting Problem
Is there a program HALT such that:
HALT(P) = yes, if P(P) halts
HALT(P) = no, if P(P) does not halt

THEOREM: There is no program to solve the halting problem (Alan Turing 1937)
Suppose a program HALT existed that solved the halting problem.
HALT(P) = yes, if P(P) halts
HALT(P) = no, if P(P) does not halt
We will call HALT as a subroutine in a new program called CONFUSE.

CONFUSE
CONFUSE(P)
{ if (HALT(P))
   then loop forever; //i.e., we don't halt
   else exit; //i.e., we halt
   // text of HALT goes here
 }

Does CONFUSE(CONFUSE) halt?
CONFUSE

CONFUSE(P)
{ if (HALT(P))
    then loop forever; \textit{i.e.}, we don't halt
  else exit; \textit{i.e.}, we halt
// text of HALT goes here }

Suppose CONFUSE(CONFUSE) halts:
then HALT(CONFUSE) = TRUE
⇒ CONFUSE will loop forever on input CONFUSE

Suppose CONFUSE(CONFUSE) does not halt
then HALT(CONFUSE) = FALSE

CONTRADICTION

Alan Turing (1912-1954)

Theorem: [1937]
There is no program to solve the halting problem

Turing's argument is essentially the reincarnation of Cantor's Diagonalization argument that we saw in the previous lecture.

Programs (computable functions) are countable, so we can put them in a (countably long) list

All Programs (the input)

\[
\begin{array}{cccccc}
P_0 & P_1 & P_2 & \ldots & P_j & \ldots \\
\hline
P_0 & & & & & \\
P_1 & & & & & \\
\vdots & & & & & \\
P_i & & & & & \\
\vdots & & & & & \\
\end{array}
\]

YES, if \( P_i(P_j) \) halts
No, otherwise

CONFUSE(P_i) halts iff \( d_i = \text{no} \)
(The CONFUSE function is the negation of the diagonal.)
Hence CONFUSE cannot be on this list.
Alan Turing (1912-1954)

Theorem: [1937]
There is no program to solve the halting problem

Is there a real number that can be described, but not computed?

Consider the real number R whose binary expansion has a 1 in the jth position iff the jth program halts on input itself.

Proof that R cannot be computed
Suppose it is, and program Q computes it. Then consider the following program:

MYSTERY(program text P)
for j = 0 to forever do {
    if (P == Pj)
        then use Q to compute jth bit of R
        return YES if (bit == 1), NO if (bit == 0)
}

MYSTERY solves the halting problem!

The Halting Set K

Definition:
K is the set of all programs P such that P(P) halts.

K = \{ Java P | P(P) halts \}

Computability Theory: Vocabulary Lesson

We call a set $S \subseteq \Sigma^*$ decidable or recursive if there is a program P such that:

$P(x) = \text{yes, if } x \in S$
$P(x) = \text{no, if } x \notin S$

Today we saw: the halting set K is undecidable

No program can decide membership in K
Decidable and Computable

Subset $S$ of $\Sigma^*$ $\iff$ Function $f_S$

<table>
<thead>
<tr>
<th>$x$ in $S$</th>
<th>$f_S(x) = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$ not in $S$</td>
<td>$f_S(x) = 0$</td>
</tr>
</tbody>
</table>

Set $S$ is decidable $\iff$ function $f_S$ is computable

Sets are “decidable” (or “undecidable”),
functions are “computable” (or not)

Computable vs. Enumerable

Computability Theory:
Some More Vocabulary

We call a set of strings $S \subseteq \Sigma^*$ enumerable or recursively enumerable (r.e.) if there is a program $P$ such that:

1. $P$ prints an (infinite) list of strings.
2. Any element on the list should be in $S$.
3. Each element in $S$ appears after a finite amount of time.

Can you enumerate all strings in $\Sigma^*$?

for $n = 0$ to infinity do
for all strings $s$ of length $n$ do
print($s$)

Can you enumerate all (syntactically valid) Java programs?

for $n = 0$ to infinity do
for all strings $s$ of length $n$ do
if check-syntax($s$) then
print($s$)

Is the halting set $K$ enumerable?

for $n = 0$ to infinity do
for all strings $s$ of length $n$ do
if check-halting($s$) then
print($s$)
Enumerating $K$

Enumerate-K

for $n = 0$ to forever {
    for $W$ = all strings of length $< n$ do {
        if $W(W)$ halts in $n$ steps then output $W$;
    }
}

$(x,y) = \text{check if program } P_x \text{ halts on } y^{th} \text{ input}$

K is not decidable
but it is enumerable!

Let $K' = \{ \text{Java } P | P(P) \text{ does not halt} \}$

Is $K'$ enumerable?

No! If both $K$ and $K'$ are enumerable, then $K$ is decidable.

Run both enumeration programs in parallel. Every $P$ will be eventually output in one of these, can use to decide in $P$ in $K$.

Oracles and Reductions

Oracle For Set $S$

Is $x \in S$?

YES/NO

Oracle for $S$
Example Oracle
$S = \text{Odd Naturals}$

Oracle for $S$

4?

$\Rightarrow$

No

81?

$\Rightarrow$

Yes

$K_0 = \text{the set of programs that take no input and halt}$

Hey, I ordered an oracle for the famous halting set $K$, but when I opened the package it was an oracle for the different set $K_0$.

But you can use this oracle for $K_0$ to build an oracle for $K$.

GIVEN: Oracle for $K_0$

P = [input I; Q]
Does $P(P)$ halt?

GIVEN: Oracle for $K_0$

BUILD: Oracle for $K$

Does $[I:=P;Q]$ halt?

K_0 = \text{the set of programs that take no input and halt}

We’ve reduced the problem of deciding membership in $K$ to the problem of deciding membership in $K_0$.

Hence, deciding membership for $K_0$ must be at least as hard as deciding membership for $K$.

Thus if $K_0$ were decidable then $K$ would be as well.
We already know $K$ is not decidable, hence $K_0$ is not decidable.

HELLO = \text{the set of programs that print hello and halt}

Let $P'$ be $P$ with all print statements removed.
(assume there are no side effects)

Is $[P';\text{print HELLO}]$ a hello program?

GIVEN: HELLO Oracle

BUILD: Oracle for $K_0$

Does $P$ halt?
Hence, the set HELLO is not decidable.

Halting with input, Halting without input, HELLO, and EQUAL are all undecidable.

Diophantine Equations

Does polynomial \(4X^2Y + XY^2 + 1 = 0\) have an integer root? I.e., does it have a zero at a point where all variables are integers?

\[ D = \{ \text{multivariate integer polynomials } P \text{ s.t. } P \text{ has root where all variables are integers} \} \]

Famous Theorem: \(D\) is undecidable!

[This is the solution to Hilbert’s 10th problem]

Resolution of Hilbert’s 10th Problem

Martin Davis, Julia Robinson, Yuri Matiyasevich (in 1982)

Polynomials can Encode Programs

There is a computable function

\[ F: \text{Java programs that take no input } \rightarrow \text{ Polynomials over the integers} \]

such that

\[ \text{program } P \text{ halts } \iff F(P) \text{ has an integer root} \]
D = the set of all integer polynomials with integer roots

Philosophical Interlude

Church-Turing Thesis
Any well-defined procedure that can be grasped and performed by the human mind and pencil/paper, can be performed on a conventional digital computer with no bound on memory.

Empirical Intuition
No one has ever given a counter-example to the Church-Turing thesis. I.e., no one has given a concrete example of something humans compute in a consistent and well defined way, but that can’t be programmed on a computer. The thesis is true.

Mechanical Intuition
The brain is a machine. The components of the machine obey fixed physical laws. In principle, an entire brain can be simulated step by step on a digital computer. Thus, any thoughts of such a brain can be computed by a simulating computer. The thesis is true.
Quantum Intuition

The brain is a machine, but not a classical one. It is inherently quantum mechanical in nature and does not reduce to simple particles in motion. Thus, there are inherent barriers to being simulated on a digital computer. The thesis is false. However, the thesis is true if we allow quantum computers.