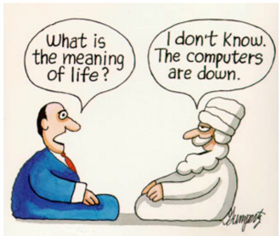
Lecture 19: Universality and Computability



Fundamental Questions

Universality. What is a general purpose computer?

Computability. Are there problems that no machine can solve?

Church-Turing thesis. Are there limits on the power of machines that we can build?

Pioneering work in the 1930's.

- (Princeton == center of universe).
- Hilbert, Gödel, Turing, Church, von Neumann.
- Automata, languages, computability, universality, complexity, logic.

Universality

Q. Which one of the following does not belong?


Cray


Dell PC

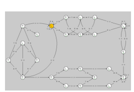

iMac

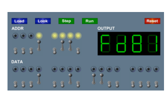

Espresso maker


Palm Pilot

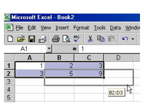

Xbox


Tivo

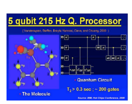

Turing machine


TOY

Java language

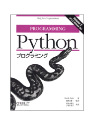

MS Excel


Java cell phone



Quantum computer


DNA computer



Python language

Turing Machine: Components

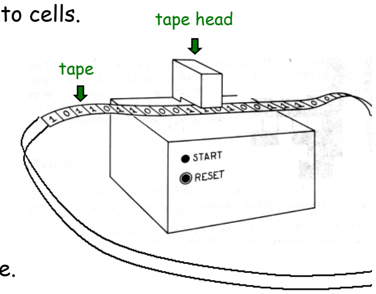
Alan Turing sought the most primitive model of a computing device.

Tape.

- Stores input, output, and intermediate results.
- One arbitrarily long strip, divided into cells.
- Finite alphabet of symbols.

Tape head.

- Points to one cell of tape.
- Reads a symbol from active cell.
- Writes a symbol to active cell.
- Moves left or right one cell at a time.



Java: As Powerful As Turing Machine

Turing machines are equivalent in power to TOY and Java.

- Can use Java to solve any problem that can be solved with a TM.
- Can use TM to solve any problem that can be solved with a TOY.
- Can use TOY to solve any problem that can be solved with Java.

Java simulator for Turing machines.

```
State state = start;
while (true) {
   char c = tape.readSymbol();
   tape.write(state.symbolToWrite(c));
   state = state.next(c);
   if      (state.isLeft())  tape.moveLeft();
   else if (state.isRight()) tape.moveRight();
   else if (state.isHalt())  break;
}
```


Turing Machine: As Powerful As TOY Machine

Turing machines are equivalent in power to TOY and Java.

- Can use Java to solve any problem that can be solved with a TM.
- Can use TM to solve any problem that can be solved with a TOY.
- Can use TOY to solve any problem that can be solved with Java.

Turing machine simulator for TOY programs.

- Encode state of memory, registers, pc, onto Turing tape.
- Design TM states for each instruction.
- Can do because all instructions:
  - examine current state
  - make well-defined changes depending on current state

TOY: As Powerful As Java

Turing machines are equivalent in power to TOY and Java.

- Can use Java to solve any problem that can be solved with a TM.
- Can use TM to solve any problem that can be solved with a TOY.
- Can use TOY to solve any problem that can be solved with Java.

TOY simulator for Java programs.

- Variables, loops, arrays, functions, linked lists, ...
- In principle, can write a Java-to-TOY compiler!

Java, Turing Machines, and TOY

Turing machines are equivalent in power to TOY and Java.

- Can use Java to solve any problem that can be solved with a TM.
- Can use TM to solve any problem that can be solved with a TOY.
- Can use TOY to solve any problem that can be solved with Java.

Also works for:

- C, C++, Python, Perl, Excel, Outlook, ...
- Mac, PC, Cray, Palm pilot, ...
- TiVo, Xbox, Java cell phone, ...

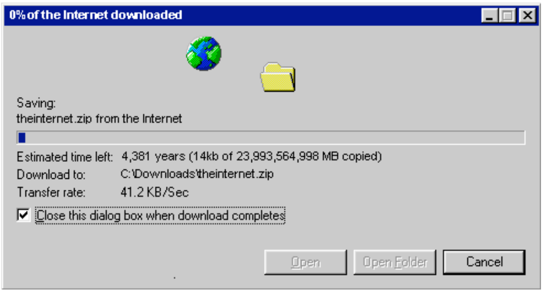
Does not work:

- DFA or regular expressions.
- Gaggia espresso maker.

Not Enough Storage?

Implicit assumption.

- TOY machine and Java program have unbounded amount of memory.
- Otherwise Turing machine is strictly more powerful.
- Is this assumption reasonable?



Universal Turing Machine

Java program: solves one specific problem.
TOY program: solves one specific problem.
TM: solves one specific problem.

Java simulator in Java: Java program to simulate any Java program.
TOY simulator in TOY: TOY program to simulate any TOY program.
UTM: Turing machine that can simulate any Turing machine.

General purpose machine.

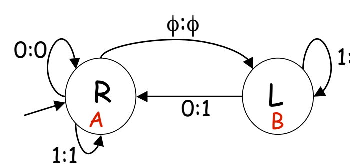
- UTM can implement any algorithm.
- Your laptop can do any computational task: word-processing, pictures, music, movies, games, finance, science, email, Web, …

TOY Simulator in TOY ?!

```
// TOY simulator in TOY code
// simulated code starts at 90 (i.e., simulated 0 is 80)
// addresses (jump, load, store) in simulated code
// must be relocated (add 80)
// main trick is keeping simulated register i in Mem[i]
// Doug Clark, 3/11/01

08:   0090            //simulated PC (initially: simulated 10)

10:   B000            r0 = 0   //ALWAYS

11:   9508   top:     r5 = M[PC]
12:   B101            r1 = 1
13:   9E50            r6 = M[r5+r0]  //inst fetch
14:   1551            r5 = r5 + r1   //pc++
15:   A508            M[PC] = r5

16:   B7FF            r7 = 255   //addr mask
17:   D767            r7 = r6 & r7

18:   B507            r5 = 7     //reg num mask
19:   D465            r4 = r6 & r5  //rb
1A:   E604            r6 = r6 >> 4
1B:   D365            r3 = r6 & r5  //ra
1C:   E604            r6 = r6 >> 4
1D:   D265            r2 = r6 & r5   //rd
1E:   E603            r6 = r6 >> 3   //opcode with index bit
1F:   B51F            r5 = 1F
20:

21:   9B30            r3 = M[r3 + r0]  //get contents of
22:   9C40            r4 = M[r4 + r0]
23:   9920            r1 = M[r2 + r0]  //but save rd num

24:   B501            r5 = 1        //mask for index check
25:   D565            r5 = r6 & r5
26:   6529            jpos r5, index
27:   B530   dispch:  r5 = BASE     //start of dispatch t
28:   5856            jmp [r5 + r6]  //opcode << 1 for 2-i

29:   B51E   index:   r5 = 1E
2A:   D665            r6 = r6 & r5   //lose index bit
2B:   1734            r7 = r3 + r4   //indexed addr
2C:   5027            jmp dispch

30:   0000   halt:    halt  //opcode 0 == BASE

32:   1134   add:     r1 = r3 + r4  //opcode 1 = BASE + 2
33:   5054            jmp opEnd

34:   2134   sub:     r1 = r3 - r4  //etc., etc.
35:   5054            jmp opEnd

36:   3134   mul:     r1 = r3 * r4
37:   5054            jmp opEnd

38:   4100   print:   print r1
39:   5011            jmp top

3A:   5056   jmp:     jmp newPC
```


Representations of a Turing Machine

Graphical:



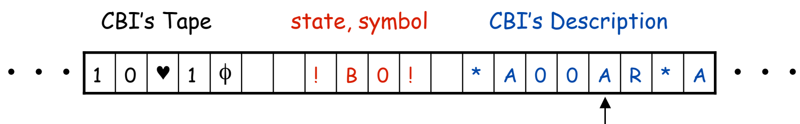
Continuous
Binary
Incrementer

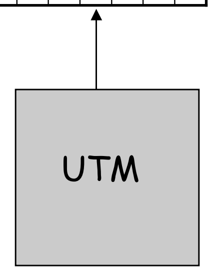
Tabular:

Current state	Symbol read	Symbol to write	Next State	Direction
A	0	0	A	R
A	1	1	A	R
A	ϕ	ϕ	B	L
B	0	1	A	R
B	1	0	B	L

Linear: * A 0 0 A R * A 1 1 A R * A ϕ ϕ B L * B 0 1 A R * B 1 0 B L *

Universal Turing Machine





UTM Operation:
- Find state, symbol in Description
- Copy new symbol to CBI's tape
- Move ♥ L or R
- Update state, symbol
- Repeat

Church-Turing Thesis

Church Turing thesis (1936). Turing machines can do any computation that can be done by any real computer.

Implications:
- No need to seek more powerful machines.
- If a computational problem can't be solved with a Turing machine, then it can't be solved on any physical computing device.

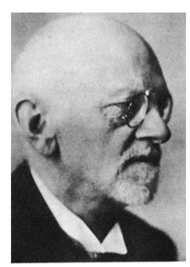
Remarks.
- "Thesis" and not a mathematical theorem because it's a statement about the physical world and not subject to proof.

Turing machine: a simple and universal model of computation.

Other Universal Models of Computation

Model of Computation	Description
Enhanced Turing Machines	Multiple heads, multiple tapes, 2D tape, nondeterminism.
Untyped Lambda Calculus	A method to define and manipulate functions. Basis of functional programming language like Lisp and ML.
Recursive Functions	Functions dealing with computation on natural numbers.
Unrestricted Grammars	Iterative string replacement rules used by linguists to describe natural languages.
Extended L-Systems	Parallel string replacement rules that model the growth of plants.
Cellular Automata	Boolean array of cells whose values change according only to the state of the adjacent cells, e.g., Game of Life.
Random Access Machines	Finitely many registers plus memory that can be accessed with an integer address. TOY, G5, Pentium IV.
Programming Languages	Java, C, C++, Perl, Python, PHP, Lisp, PostScript, Excel

7.7: Computability



Take any definite unsolved problem, such as the question as to the irrationality of the Euler-Mascheroni constant γ, or the existence of an infinite number of prime numbers of the form 2^n+1. However unapproachable these problems may seem to us and however helpless we stand before them, we have, nevertheless, the firm conviction that their solution must follow by a finite number of purely logical processes.
-David Hilbert, in his 1900 address to the International Congress of Mathematics

Halting Problem

Halting problem. Write a Java function that reads in a Java function f and its input x, and decides whether f(x) results in an infinite loop.

Ex: is there a *perfect* number of the form: 1, 1+x, 1+2x, 1+3x, ... (perfect: integer that equals the sum of its proper divisors)

- x = 1: halts when n = 6 = 1 + 2 + 3.
- x = 2: finding odd perfect number is famous open math problem.
- x = 3: halts when n = 28 = 1 + 2 + 4 + 7 + 14.

```
public void f(int x) {
   for (long n = 1; true; n = n + x) {
      long sum = 0;
      for (long i = 1; i < n; i++)
         if (n % i == 0) sum = sum + i;
      if (sum == n) return;
   }
}
```

halt if n is perfect

Undecidable Problem

A yes-no problem is undecidable if no Turing machine exists to solve it.

Theorem (Turing, 1937). The halting problem is undecidable.

- No Turing machine can solve the halting problem.
- By universality, not possible to write a Java function either.

Proof intuition: lying paradox.

- Divide all statements into two categories: truths and lies.
- How do we classify the statement: *I am lying.*

Key element of paradox: self-reference.

Halting Problem Proof

Assume the existence of `halt(f,x)`:

- Input: a function `f` and its input `x`.
- Output: `true` if `f(x)` halts, and `false` otherwise.
- Note: `halt(f,x)` does not go into infinite loop.

We prove by contradiction that `halt(f,x)` does not exist.

- *Reductio ad absurdum* : if any logical argument based on an assumption leads to an absurd statement, then assumption is false.

encode f and x as strings

```
public boolean halt(String f, String x) {
   if (???) return true;
   else     return false;
}
```


Halting Problem Proof

Assume the existence of `halt(f,x)`:

- Input: a function `f` and its input `x`.
- Output: `true` if `f(x)` halts, and `false` otherwise.

Construct function `strange(f)` as follows:

- If `halt(f,f)` returns `true`, then `strange(f)` goes into an infinite loop.
- If `halt(f,f)` returns `false`, then `strange(f)` halts.

f is a string so legal (if perverse) to use for second input

```
public void strange(String f) {
   if (halt(f, f)) {
      while (true)
         ;
   }
}
```

Halting Problem Proof

Assume the existence of `halt(f,x)`:

- Input: a function `f` and its input `x`.
- Output: `true` if `f(x)` halts, and `false` otherwise.

Construct function `strange(f)` as follows:

- If `halt(f,f)` returns `true`, then `strange(f)` goes into an infinite loop
- If `halt(f,f)` returns `false`, then `strange(f)` halts.

In other words:

- If `f(f)` halts, then `strange(f)` goes into an infinite loop.
- If `f(f)` does not halt, then `strange(f)` halts.

Halting Problem Proof

Assume the existence of `halt(f,x)`:

- Input: a function `f` and its input `x`.
- Output: `true` if `f(x)` halts, and `false` otherwise.

Construct function `strange(f)` as follows:

- If `halt(f,f)` returns `true`, then `strange(f)` goes into an infinite loop
- If `halt(f,f)` returns `false`, then `strange(f)` halts.

In other words:

- If `f(f)` halts, then `strange(f)` goes into an infinite loop.
- If `f(f)` does not halt, then `strange(f)` halts.

Call `strange()` with ITSELF as input.

- If `strange(strange)` halts then `strange(strange)` does not halt.
- If `strange(strange)` does not halt then `strange(strange)` halts.

Halting Problem Proof

Assume the existence of `halt(f,x)`:

- Input: a function `f` and its input `x`.
- Output: `true` if `f(x)` halts, and `false` otherwise.

Construct function `strange(f)` as follows:

- If `halt(f,f)` returns `true`, then `strange(f)` goes into an infinite loop
- If `halt(f,f)` returns `false`, then `strange(f)` halts.

In other words:

- If `f(f)` halts, then `strange(f)` goes into an infinite loop.
- If `f(f)` does not halt, then `strange(f)` halts.

Call `strange()` with ITSELF as input.

- If `strange(strange)` halts then `strange(strange)` does not halt.
- If `strange(strange)` does not halt then `strange(strange)` halts.

Either way, a contradiction. Hence `halt(f,x)` cannot exist.

Consequences

Halting problem is not "artificial."

- Undecidable problem reduced to simplest form to simplify proof.
- Self-reference not essential.
- Closely related to practical problems.

No input halting problem. Give a function with no input, does it halt?

Program equivalence. Do two programs always produce the same output?

Uninitialized variables. Is variable x initialized?

Dead code elimination. Does control flow ever reach this point in a program?

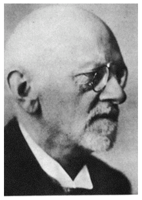
More Undecidable Problems

Hilbert's 10th problem.

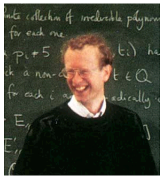
- "*Devise a process according to which it can be determined by a finite number of operations whether a given multivariate polynomial has an integral root.*"

Examples.

- $f(x, y, z) = 6x^3 y z^2 + 3xy^2 - x^3 - 10$. ⇐ yes: $f(5, 3, 0) = 0$
- $f(x, y) = x^2 + y^2 - 3$. ⇐ no
- $f(x, y, z) = x^n + y^n - z^n$ ⇐ yes if $n = 2, x = 3, y = 4, z = 5$ ⇐ no if $n \geq 3$ and $x, y, z > 0$. (Fermat's Last Theorem)



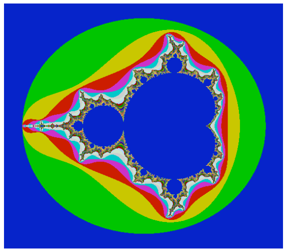
Hilbert



Andrew Wiles, 1995

More Undecidable Problems

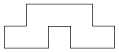
Optimal data compression. Find the shortest program to produce a given string or picture.

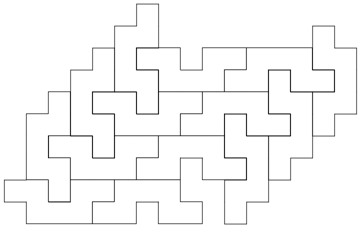


Mandelbrot Set (40 lines of code)

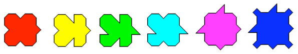
More Undecidable Problems

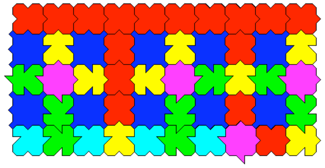
Polygonal tiling. Given a polygon, is it possible to tile the whole plane with copies of that shape?





Difficulty. Tilings may exist, but be aperiodic!





Reference: http://www.uwgb.edu/dutchs/symmetry/aperiod.htm

More Undecidable Problems

Virus identification. Is this program a virus?

```
Private Sub AutoOpen()
On Error Resume Next
If System.PrivateProfileString("", CURRENT_USER\Software\Microsoft\Office\9.0\Word\Security",
                               "Level") <> "" Then

CommandBars("Macro").Controls("Security...").Enabled = False
. . .
For oo = 1 To AddyBook.AddressEntries.Count
   Peep = AddyBook.AddressEntries(x)
   BreakUmOffASlice.Recipients.Add Peep
   x = x + 1
   If x > 50 Then oo = AddyBook.AddressEntries.Count
Next oo
. . .
BreakUmOffASlice.Subject = "Important Message From " & Application.UserName
BreakUmOffASlice.Body = "Here is that document you asked for ... don't show anyone else ;-)"
. . .
```

Can write programs in MS Word.
This statement disables security.

Melissa Virus, March 28, 1999

Implications of Computability

Step-by-step reasoning.

- We *assume* that it will solve any technical or scientific problem.
- *Not quite* says the halting problem.

Practical implications.

- Work with limitations.
- Recognize and avoid undecidable problems.
- Anything that is (or could be) like a computer has the same flaw.

Speculative Models of Computation

Rule of thumb. Any pile of junk that has state and a deterministic set of rules is universal, and hence has intrinsic limitations!

Model of Computation	Description
Quantum Computer	Compute using the superposition of quantum states.
Billiard Ball Computer	Colliding billiard balls with barriers and elastic collisions.
DNA Computer	Compute using biological operations on DNA strands.
Soliton Collision System	Time-gated Manakov spatial solitions in a homogeneous medium.
Dynamical System	Dynamics based computing based on chaos.
Logic	Formal mathematics.
Human Brain	? ? ?

Turing's Key Ideas

Turing's 4 key ideas.

- Computing is the same as manipulating symbols.
  Encode numbers as strings.
- Computable at all = computing with a Turing machine.
  Church-Turing thesis.
- Existence of Universal Turing machine.
  general-purpose, programming computers
- Undecidability of the Halting problem.
  computers have inherent limitations

Turing's Key Ideas

Turing's 4 key ideas.

Computing is the same as manipulating symbols.
Encode numbers as strings.
Computable at all = computing with a Turing machine.
Church-Turing thesis.
Existence of Universal Turing machine.
general-purpose, programming computers
Undecidability of the Halting problem.
computers have inherent limitations

Hailed as one of top 10 science papers of 20th century.

Reference: *On Computable Numbers, With an Application to the Entscheidungsproblem* by A. M. Turing. In Proceedings of the London Mathematical Society, ser. 2. vol. 42 (1936-7), pp.230-265.