Models of Computation, Turing Machines, and the Limits of Turing Computation

Bruce MacLennan

Motivation for Models of Computation

• What questions are models of computation intended to answer?
• What are the simplifying assumptions of models of computation?
• Why were models of computation developed in the early 20th century, before there were any computers?

Effective Calculability

• Mathematicians were interested in effective calculability:
  – What can be calculated by strictly mechanical methods using finite resources?
  – Think of a human “computer”
    – following explicit rules that require no understanding of mathematics
    – supplied with all the paper & pencils required

Related Issues

• Formal mathematics: Can mathematical proof & derivation be reduced to purely mechanical procedures requiring no use of intuition?
• Mechanization of thought: Can thinking be reduced to mechanical calculation?

Formal Logic

• Originally developed by Aristotle (384–322 BCE)
• A syllogism:
  All men are mortal
  Socrates is a man
  ∴ Socrates is mortal
• Formal logic: the correctness of the steps depend only on their form (syntax), not their meaning (semantics):
  All M are P
  S is M
  ∴ S is P
• More reliable, because more mechanical
Calculus

- In Latin, calculus means pebble
- In ancient times calculi were used for calculating (as on an abacus), voting, and may other purposes
- Now, a calculus is:
  - an mechanical method of solving problems
  - by manipulating discrete tokens
  - according to formal rules
- Examples: algebraic manipulation, integral & differential calculi

Assumptions of Calculi

- Information (data) representation is:
  - formal (info. represented by arrangements)
  - finite (finite arrangements of atomic tokens)
  - definite (can determine symbols & syntax)
- Information processing (rule following) is:
  - formal (depends on arrangement, not meaning)
  - finite (finite number of rules & processing time)
  - definite (know which rules are applicable)

Thought as Calculation

- “By ratiocination I mean computation.” — Thomas Hobbes (1588–1679)
- “Then, in case of a difference of opinion, no discussion … will be any longer necessary … It will rather be enough for them to take pen in hand, set themselves to the abacus, and … say to one another, “Let us calculate!”” — Leibniz (1646–1716)
- Boole (1815–64): his goal was “to investigate the fundamental laws of those operations of mind by which reasoning is performed; to give expression to them in the symbolical language of a Calculus”

Early Investigations in Mechanized Thought

- Leibniz (1646–1716): mechanical calculation & formal inference
- Boole (1815–1864): “laws of thought”
- Jevons (1835–1882): logical abacus & logical piano
- von Neumann (1903–1957): computation & the brain

Some Models of Computation

- Markov Algorithms — based on replacement of strings by other strings
- Lambda Calculus — based on LISP-like application of functions to arguments
- SK Calculus — based on two operations:
  - $((K X) Y) \Rightarrow X$
  - $((S X) Y) Z) \Rightarrow ((X Z) (Y Z))$
- Turing Machine — most common

Intuitive Basis of Turing Machine

- What could be done
  - by a person following explicit formal rules
  - with an unlimited supply of paper and pencils?
- Assumption: Any "effective" (mechanical) calculation could be carried out in this way
- Reduce to bare essentials (for simplicity):
  - symbols written on a long tape
  - can read/write only one symbol at a time
  - limited memory for the "state" of the calculation
Colossus: A Real Turing Machine

- Developed in UK in 1943–4 to crack Nazi codes
- Although Turing was not directly involved with Colossus, he was involved with other computerized code-breaking efforts
- Turing described the TM model in 1936

Defining a Specific TM

- We must specify the "alphabet" of symbols used on the tape
  - typically 0, 1, and b (blank)
  - this alphabet is always sufficient (binary coding)
- We must specify the number of states (memory)
- We must specify a finite set of rules of the form:
  - (current state, symbol on tape, symbol to write, next state, direction to move)
  - for example, (3, 1, 0, 2, L)
  - rules may be represented in diagram:

TM Example: Bit Inverter (1)

- 0 1 1
- 1 \rightarrow 0, R
- 0 \rightarrow 1, R

TM Example: Bit Inverter (2)

- 1 1 1
- 0 \rightarrow 1, R
- 1 \rightarrow 0, R

TM Example: Bit Inverter (3)

- 1 0 1
- 1 \rightarrow 0, R
- 0 \rightarrow l, R

TM Example: Bit Inverter (4)

- 1 0 0
- halts!
Unary Addition

- Represent the number $N$ by $N+1$ marks (1 in this case) — *unary* notation
- So the numbers $M$ and $N$ will be represented by $M+1$ and $N+1$ marks (with a blank between)
- The sum should be $M+N+1$ marks

\[
\begin{array}{c}
\text{bl} \cdots \text{bl} \cdots \text{bl} \\
M+1 \\
\text{bl} \cdots \text{bl} \\
N+1 \\
\text{bl} \cdots \text{bl} \\
M+N+1
\end{array}
\]
TM Example: Addition (6)

[Diagram of a Turing Machine]

...

b b b l l l l l l l l b ...

Ordinary Turing Machine

- We can design a Turing machine $M$ for a specific purpose
- For each allowable input $x$ it produces the corresponding output $y$

Universal Turing Machine

- We can design a Turing machine $U$ that can emulate any Turing machine $M$
- Let $m$ be an encoding of $M$ (e.g., its rules)
- For each allowable input $x$ it produces the corresponding output $y$

Equivalence Between TMs and Other Models of Computation

- If we can use some model of computation to program a UTM, then we can emulate any TM
- So this model is at least as powerful as TMs
- If we can design TM to emulate another kind of universal machine, then UTM can emulate it
- So other model is no more powerful than TMs
- The way to prove equivalent "power" of different models of computation
- Equivalent in terms of "computability" not space/time efficiency
General-Purpose Computers

• The Universal Turing Machine is theoretical foundation of general purpose computer
• Instead of designing a special-purpose computer for each application
• Design one general-purpose computer:
  – interprets program (virtual machine description) stored in its memory
  – emulates that virtual machine

Church-Turing Thesis

• CT Thesis: The set of effectively calculable problems is exactly the set of problems solvable by TMs
• Empirical evidence: All the independently designed models of computation turned out to be equivalent to TM in power
• Easy to see how any calculus can be emulated by a TM
• Easy to see how any (digital) computer can be emulated by a TM (and vice versa)
• But, there is research in non-Turing models of computation

The Liar Paradox

• Epimenides the Cretan (7th cent. BCE) said, “The men of Crete were ever liars …”
• “If you say that you are lying, and say it truly, you are lying.” — Cicero (106–43 BCE)

Undecidability of the Halting Problem (Informal)

• Assume we have procedure \text{Halts} that decides halting problem for any program/input pair
• Let \text{P} \text{(X)} represent the execution of program \text{P} on input \text{X}
• \text{Halts} \text{(P, X)} = \text{true} if and only if program \text{P} halts on input \text{X}
• \text{Halts} \text{(P, X)} = \text{false} if and only if program \text{P} doesn’t halts on input \text{X}
• Program \text{P} encoded as string or other legal input to programs

Assumed Turing Machine for Halting Problem

• We can design a Turing machine \text{Halts} that can decide, for any Turing machine \text{P} and input \text{x}, whether \text{P} halts on \text{x}
• Let \text{p} be an encoding of \text{P} (e.g., its rules)
• If \text{P} halts on \text{x}:
  \text{Halts} \text{(p, X)} = \text{true}
Assumed Turing Machine for Halting Problem (2)

- If $P$ doesn’t halt on $x$:

Undecidability of the Halting Problem (2)

- Define the “paradoxical procedure” $Q$:
  1. procedure $Q(P)$:
  2. if $\text{Halts}(P, P)$ then
  3. go into an infinite loop
  4. else // $\text{Halts}(P, P)$ is false, so
  5. halt immediately

- Now $Q$ is a program that can be applied to any program string $P$
Undecidability of the Halting Problem (3)

- What will be the effect of executing $Q(Q)$?
- If $\text{Halts}(Q, Q) = \text{true}$, then go into an infinite loop, that is, don’t halt
  - But $\text{Halts}(Q, Q) = \text{true}$ iff $Q(Q)$ halts
- If $\text{Halts}(Q, Q) = \text{false}$, then halt immediately
  - But $\text{Halts}(Q, Q) = \text{false}$ iff $Q(Q)$ doesn’t halt
- So $Q(Q)$ halts if and only if $Q(Q)$ doesn’t halt
- A contradiction!
- Our assumption (that Halts exists) was false

Rice’s Theorem (Informal)

- Suppose that $B$ is any behavior that a program might exhibit on a given input
  - examples: print a 0, open a window, delete a file, generate a beep
- Assume that we have a procedure $\text{DoesB}(P, X)$ that decides whether $P(X)$ exhibits behavior $B$
- As in Turing’s proof, we show a contradiction

Rice’s Theorem (2)

- Define a paradoxical procedure $Q$:
  1. procedure $Q(P)$:
  2. if $\text{DoesB}(P, P)$ then
  3. don’t do $B$
  4. else
  5. do $B$
- Note that $B$ must be a behavior that we can control

Rice’s Theorem (3)

- Consider the result of executing $Q(Q)$
- $Q(Q)$ does $B$ if and only if $Q(Q)$ doesn’t do $B$
- Contradiction shows our assumption of existence of decision procedure $\text{DoesB}$ was false
- A TM cannot decide any “controllable” behavior for all program/input combinations

Gödel’s Incompleteness Theorem (informally)

- By constructing a “paradoxical proposition” that asserts own unprovability, can prove:
- In any system of formal logic (powerful enough to define arithmetic) there will be a true proposition that be neither proved nor disproved in that system
- Yet by reasoning outside the system, we can prove it’s true
- Does this imply that human reasoning cannot be captured in a formal system (calculus)? Or reduced to calculation?
- Philosophers have been grappling with this problem since the 1930s

Hypercomputation

- CT Thesis says “effectively calculable” = “Turing-computable”
- Some authors equate “computable” with Turing-computable
- If true, then the limits of the TM are the limits of computation
- Is human intelligence “effectively calculable”?
- Hypercomputation = computation beyond the “Turing limit”