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

Bruce MacLennan

Models

- A *model* is a tool intended to address a class of questions about some domain of phenomena
- They accomplish this by making simplifications (*idealizing assumptions*) relative to the class of questions
- As tools, models are:
  - *ampliative* (better able to answer these questions)
  - *reductive* (make simplifying assumptions)

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
  <u>Socrates is a man</u>
  ∴ Socrates is mortal
- Formal logic: the correctness of the steps depend only on their *form* (syntax), not their *meaning* (semantics):

  All M are P
  <u>S is M</u>
  ∴ 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, logical 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
- Turing (1912–1954): neural nets, artificial intelligence, "Turing test"

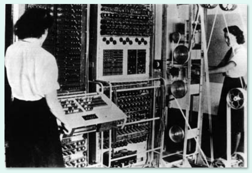
Some Models of Computation

- 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:
  $((\mathsf{K}\ X)\ Y) \Rightarrow X$
  $(((\mathsf{S}\ X)\ Y)\ Z) \Rightarrow ((X\ Z)\ (Y\ Z))$
- 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

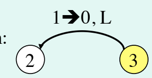


- Developed in UK in 1943–4 to crack Nazi codes
- Although Turing was *not* directly involved with Colossus, he was involved with other computerized code-breaking efforts
- Turing described the TM model in 1936

COSC 312 — Turing Machines 13

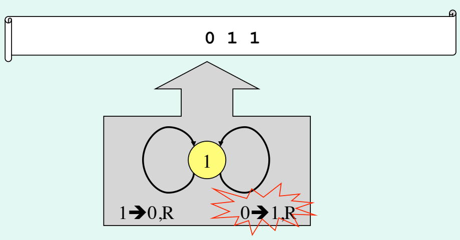
Defining a Specific TM

- 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:



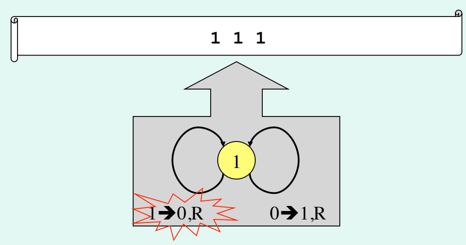
COSC 312 — Turing Machines 14

TM Example: Bit Inverter (1)



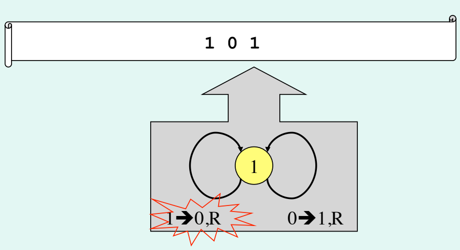
COSC 312 — Turing Machines 15

TM Example: Bit Inverter (2)



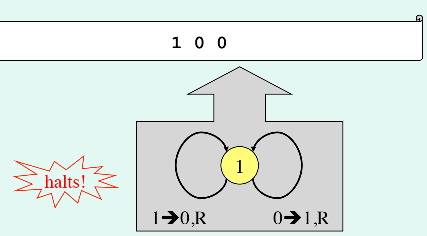
COSC 312 — Turing Machines 16

TM Example: Bit Inverter (3)



COSC 312 — Turing Machines 17

TM Example: Bit Inverter (4)



COSC 312 — Turing Machines 18

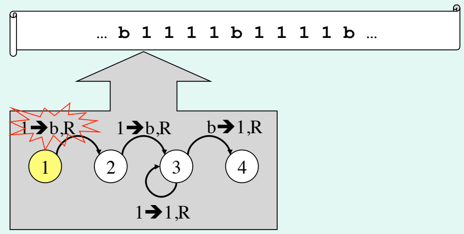
Unary Addition

- 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

$$b\underbrace{1\ \cdots\ 1}_{M+1}b\underbrace{1\ \cdots\ 1}_{N+1}b \Rightarrow b\underbrace{1\ \cdots\ 1}_{M+N+1}b$$

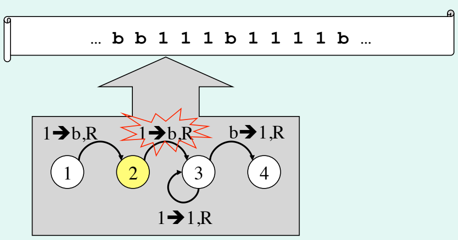
COSC 312 — Turing Machines 19

TM Example: Addition (1)



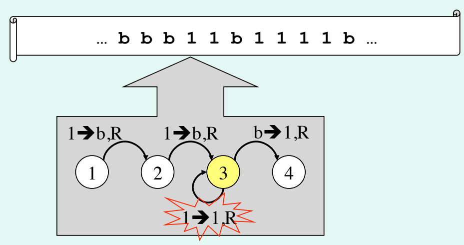
COSC 312 — Turing Machines 20

TM Example: Addition (2)



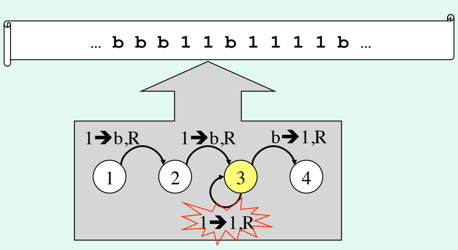
COSC 312 — Turing Machines 21

TM Example: Addition (3)



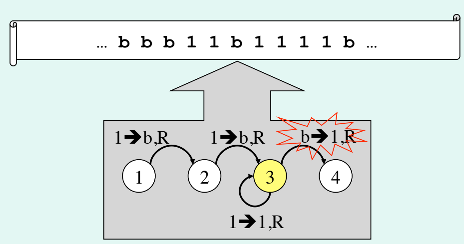
COSC 312 — Turing Machines 22

TM Example: Addition (4)



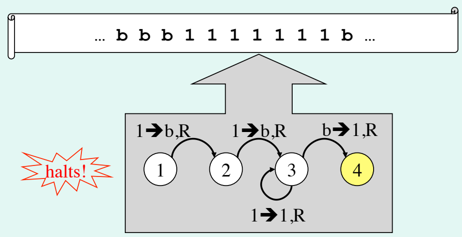
COSC 312 — Turing Machines 23

TM Example: Addition (5)



COSC 312 — Turing Machines 24

TM Example: Addition (6)



COSC 312 — Turing Machines 25

A Physical Turing Machine

See http://aturingmachine.com

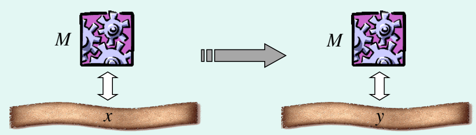
COSC 312 — Turing Machines 26

The Universal Turing Machine

COSC 312 — Turing Machines 27

Ordinary Turing Machine

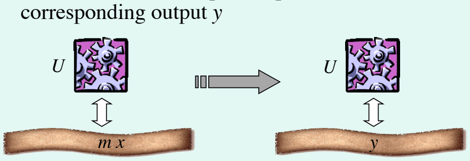
- We can design a Turing machine *M* for a specific purpose
- For each allowable input *x* it produces the corresponding output *y*



COSC 312 — Turing Machines 28

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*



COSC 312 — Turing Machines 29

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 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

COSC 312 — Turing Machines 30

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 Limits 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)

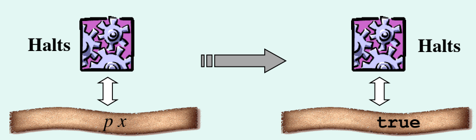
"I am lying."

Undecidabilty of the Halting Problem (Informal)

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

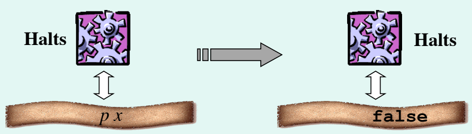
Assumed Turing Machine for Halting Problem

- We can design a Turing machine **Halts** that can decide, for any Turing machine *P* and input *x*, whether *P* halts on *x*
- Let *p* be an encoding of *P* (e.g., its rules)
- If *P* halts on *x*:



Assumed Turing Machine for Halting Problem (2)

- If P doesn't halt on x:



COSC 312 — Turing Machines 37

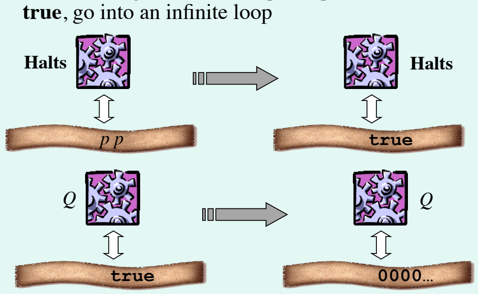
Undecidabilty of the Halting Problem (2)

- Define the "paradoxical procedure" Q:
  1. procedure Q (P):
  2. if **Halts** (P, P) then
  3. *go into an infinite loop*
  4. else // **Halts** (P, P) is false, so
  5. *halt immediately*
- Now Q is a program that can be applied to any program string P

COSC 312 — Turing Machines 38

Turing Machine Q

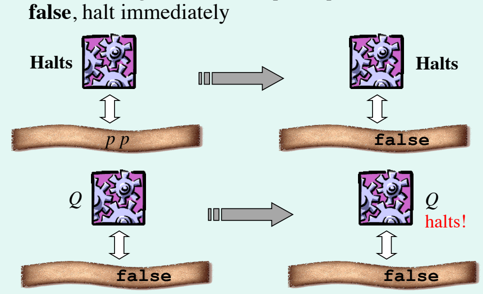
- After running TM **Halts** on p and p, if result was **true**, go into an infinite loop



COSC 312 — Turing Machines 39

Turing Machine Q (2)

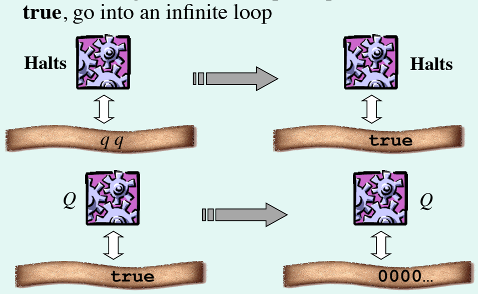
- After running TM **Halts** on p and p, if result was **false**, halt immediately



COSC 312 — Turing Machines 40

TM Q Applied to q

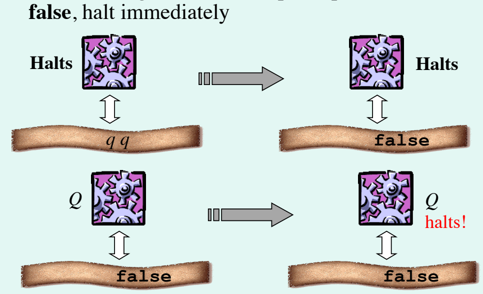
- After running TM **Halts** on q and q, if result was **true**, go into an infinite loop



COSC 312 — Turing Machines 41

TM Q Applied to q (2)

- After running TM **Halts** on q and q, if result was **false**, halt immediately



COSC 312 — Turing Machines 42

Undecidabilty of the Halting Problem (3)

- What will be the effect of executing Q (Q)?
- If **Halts** (Q, Q) = **true**, then go into an infinite loop, that is, don't halt
  - But **Halts** (Q, Q) = **true** iff Q (Q) halts
- If **Halts** (Q, Q) = **false**, then halt immediately
  - But **Halts** (Q, Q) = **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

COSC 312 — Turing Machines 43

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 **DoesB** (P, X) that decides whether P (X) exhibits behavior B
- As in Turing's proof, we show a contradiction

COSC 312 — Turing Machines 44

Rice's Theorem (2)

- Define a paradoxical procedure Q:
  1. procedure Q (P):
  2. if **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

COSC 312 — Turing Machines 45

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 **DoesB** was false
- A TM cannot decide any "controllable" behavior for all program/input combinations

COSC 312 — Turing Machines 46

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

COSC 312 — Turing Machines 47

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"

COSC 312 — Turing Machines 48