CS 126 Lecture T4: Computability

Outline

- **Introduction**
- Nature of Turing machines
- Uncomputability
- Conclusions

Where We Are

- T1
  - Simplest language generators: regular expressions
  - Simplest language recognizer: FSAs
- T2: more powerful machines
  - FSA, NFSA
  - PDA, NPDA
  - TM
- T3: more powerful languages associated with the more powerful machines
- **T4:**
  - **Nature of TM: the most powerful machines**
  - **Languages that no machine can ever deal with**

Limits

- As we make machines more powerful, we can recognize more languages

Are there languages that no machine can recognize?
Are there limits on the power of machines that we can imagine?

- Intuively, machines are finite representations of languages
- There are “more” languages than machines (“uncountable” vs. “countable”)
- Therefore, there have to be some “weird” languages! Let’s look for those!

A Puzzle ("Post's Correspondence Problem")

Given a set of cards

- N types of cards, as many as needed
- each has a top string and a bottom string

Example 1:

BAB	A	AB	BA
A	ABA	B	B

Puzzle: find a way to arrange the cards (using as many copies of each as you want) so that top and bottom strings are the same (or report that it's impossible).

Post's Correspondence Examples

Solution to Example 1:

A	BA	BAB	AB	A
ABA	B	A	B	ABA

ABABABABA

Example 2: (no solution)

A	ABA	B	A
BAB	B	A	B

Surprising fact: this puzzle is UNSOLVABLE!

- can't write a program to determine whether a given set of cards can be so arranged

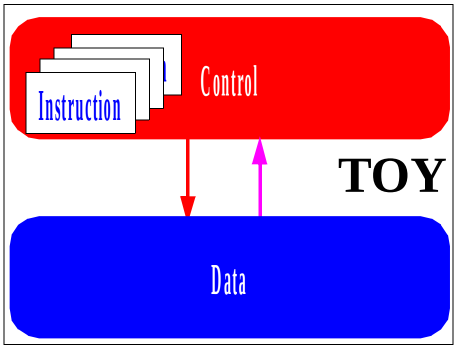
Outline

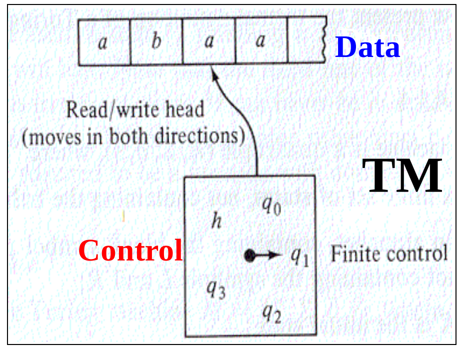
- ~~Introduction~~
- **Nature of Turing machines**
  - Can match power of any sophisticated **automata**
  - Can match power of any **real special purpose computer**
  - Can be made general to **simulate any special purpose TM**
  - Therefore can match power of any **real general purpose computer**
  - In fact, it can match the power of **any computation methods**
- Uncomputability
- Conclusions

TM: the Ultimate Machine!

- "Power" = ability to recognize languages
- "Impossible" to make a Turing Machine more powerful!
- All the following attempts have been proven to be equivalent to a vanilla TM:
  - Composition of multiple TMs
  - Multiple tapes
  - Multiple read/write heads
  - Multi-dimensional tapes
  - Non-determinism
- In other words, we can construct a regular TM that is equivalent to any of these

TMs: as Powerful as any Real Programs





proof sketch:

- encode state of memory, PC, etc. on TM tap
- develop TM states for each instruction
- can do because all instructions
  - examine current state
  - make well-defined changes
    - depending on current state
- could simulate at gate level, machine level,...

TMs: as Powerful as any Real Programs

CLAIM: Turing machines are equivalent
to C programs

proof sketch:

C program -> TOY program -> TM

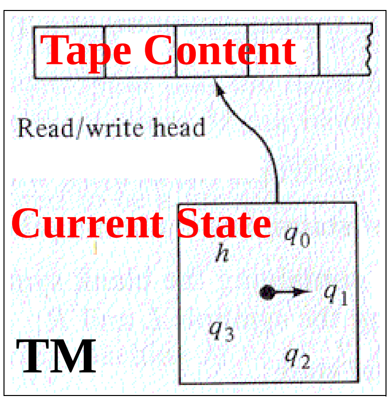
TM -> C program

Works for all programs and machines ?

Universal Turing Machine

- So far, we have built **special purpose TMs** for each different problem, example: one that recognizes palindrome
- This is like **special purpose computers** piror to von Neumann store-program computers
- Question: can we make a general purpose TM just like the general purpose computers?
- Universal Turing Machine (UTM): **a general purpose Turing machine** that can simulate the operation of **any** special purpose TM
- How? Just like a von Neumann architecture, the idea is to **store the representation** of a TM inside a UTM

What Are the “Ingredients” of a TM?

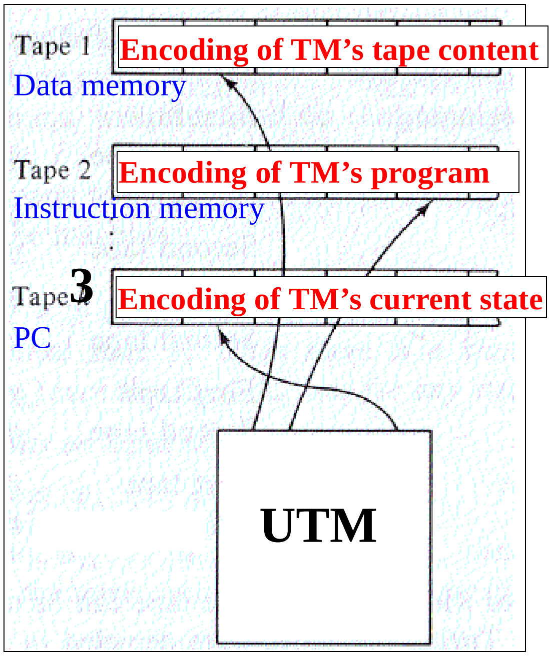


The “Program”

		0			1			2		
mark	0	3	0	L	1	0	R	2	0	R
move1	1	5	0	L	1	1	R	1	2	R
move2	2	6	0	L	2	1	R	2	2	R
YES	3									
NO	4	4	0	L	4	1	L	4	2	L
test1	5	3	0	L	7	0	L	4	2	L
test2	6	3	0	L	4	1	L	7	0	L
back	7	0	0	R	7	1	L	7	2	L

- Three “ingredients” of a special purpose TM:
  - The TM “program”
  - The tape content
  - The current TM state

How to Make a UTM? How Does It Work?



- Encode the three ingredients of TM using three tapes of a UTM
- UTM (simulates TM)
  - read tape 1
  - read tape 3
  - consult tape 2 for what to do
  - write tape 1 if necessary
  - move head 1
  - write tape 3
- Very much like the fetch-incr-execute cycle of a von Neumann machine!!
- Can reduce 3-tape UTM to a single-tape one

Church/Turing Thesis

Q. Which problems can a Turing machine solve?
A. *Any* problem *any* computer can solve!

A "thesis", not a "theorem"
can't be proved because we can't precisely define "solving" a "problem" (computability)

- Turing machines are so powerful that they are basis of the very definition of **algorithm**: an algorithm is what a TM can do!

Church/Turing Thesis (cont.)

More evidence in favor

- different ways to define "computable"
  - universal TM
  - lambda calculus
  - Post production system
  - "recursive" functions
- all have been proven equivalent

Church/Turing Thesis (cont.)

If a problem can't be solved by a TM, we ASSUME that it can't be solved by any other computer

If a problem can't be solved by *any* specific particular machine, we ASSUME that it can't be solved by any other computer T4.5

Outline

- ~~Introduction~~
- ~~Nature of Turing machines~~
- **Uncomputability**
- Conclusions

Halting Problem

Write a C program that reads in another program and its input and decides whether or not it goes into an infinite loop

Program 1:

```
while (x != 1)
   if (x > 2) x = x - 2; else x = x + 2;
```

Ex:

8 6 4 2 4 2 4 2 4 2 4 2

9 7 5 3 1

Halts iff x odd

Program 2:

```
while (x != 1)
   if (x % 2) x = 3*x+1; else x = x/2;
```

Ex:

8 4 2 1

7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

???

A Warmup Paradox

- Classify statements into two categories: truths and lies
- How do we classify the statement "I'm lying"?
  - If I'm telling the truth, then I'm lying.
  - If I'm lying, then I'm telling the truth.
- Well known problem with self-referential statements:
  - The barber that must cut hair only for all those who don't cut their own hair; should the barber cut his own hair?
  - A set of things that are not members of themselves; is this set a member of itself? (Russell's Paradox)

Halting Problem (cont.)

THEOREM: The halting problem is unsolvable.

Proof:

- Assume the existence of HALT(P, x), which
  - takes any program P and input x as inp
  - outputs YES if P(x) halts, NO otherwise

    note: always returns either YES or NO

    [does not go into an infinite loop]
- Construct the strange program XX(P)
  - calls HALT(P, P)
  - halts if HALT(P, P) outputs NO
  - infinite loops if HALT(P, P) outputs YES
- In other words
  - if P(P) does not halt, XX(P) halts
  - if P(P) halts, XX(P) does not halt
- Call XX with *itself* as input

  if XX(XX) does not halt, XX(XX) halts

  if XX(XX) halts, XX(XX) does not halt

Both cases lead to a contradiction

HALT(P, x) cannot exist

Unsolvable Problems

- Halting Problem not "artificial"
  - reduced to simplest terms to simplify proof
  - closely related to some practical problems

Very profound implications

- Unsolvability of Halting Problem can be used to show other problems to be unsolvabl

Technique: "code" Turing machine into problem

- given a TM
  create an instance of the problem
  with the property that
  if there is a solution to the problem
  the corresponding TM halts

Unsolvable Problems (cont.)

Examples

* Post correspondence problem
* Do two programs produce the same output?
* Hilbert's Tenth Problem (see next slide)
* Equivalence of context-free grammars
* Optimal data compression
  (shortest program to output a given string)

Hilbert's Tenth Problem

Write a program to test whether a given multivariate polynomial has an integral root

Example 1:

$$6x^3yz^2 + 3xy^2 - x^3 - 10$$

YES: x = 5, y = 3, z = 0.

Example 2:

$$x^2 + y^2 - 3$$

NO.

Hilbert's Tenth Problem (cont.)

- Such a program would be useful in numerous applications in physics, biology, statistics, and other fields
- Dates back to Diophantine (over 2000 years old)
- Listed as one of 23 fundamental problems for the next century by Hilbert in 1900

Matijasevic in the 1970s proved the problem to be UNSOLVABLE (!!)

Outline

- ~~Introduction~~
- ~~Nature of Turing machines~~
- ~~Uncomputability~~
- **Conclusions**
  - There are far "more" languages than there are machines
  - Therefore, there are far "more" provably unsolvable problems than solvable ones!
  - What's the implication?

Implications

Practical

- work with limitations
  - recognize and avoid unsolvable problems
- learn from structure
  - same theory tells us about efficiency

Philosophical

(caveat: ask a philosopher)

- we "assume" that step-by-step reasoning will solve *any* scientific or technical problem
- "not quite" says the halting problem
- anything that "is like" (could be) a computer has the same flaw
  - physical machine (rods/gears, etc.)
  - human brain?
  - matter itself?
  - universe?