Unsolvable Problems

Reading

- Malik Magdon-Ismail. Discrete Mathematics and Computing.
  - Chapter 27

Overview

- Programmable Turing Machines.
- Examples of unsolvable problems.
  - Post's Correspondence Problem (PCP)?
  - HalfSum?
  - Auto-Grade?
  - Ultimate-Debugger?
- $\mathcal{L}_{TM}$: The language recognized by a Universal Turing Machine.
  - $\mathcal{L}_{TM}$ is undecidable – cannot be solved!
- Auto-Grade and Ultimate-Debugger do not exist.
- What about HalfSum?

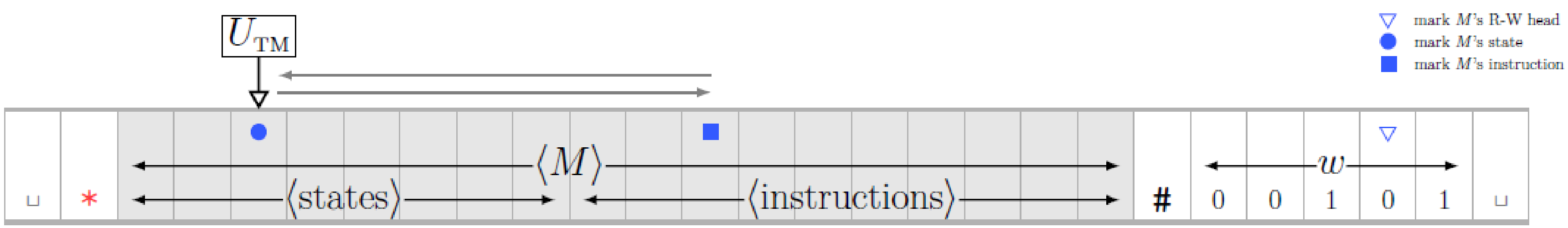
Programmable Turing Machine: Universal Turing Machine

- A Turing Machine M has a binary encoding $< M >$. Its input w is a binary string.
- The encoding $< M > \#w$ can then be the input to another Turing Machine U_{TM}

$$U_{TM}(< M > \#w) = \begin{cases} \textit{halt with ACCEPT} \;\; \textit{if} \;\; M(w) = \textit{halt with ACCEPT}; \\ \textit{halt with REJECT} \;\; \textit{if} \;\; M(w) = \textit{halt with REJECT}; \\ \textit{loop forever} \;\; \textit{if} \; M(w) = \textit{loop forever} \end{cases}$$

computer program program input

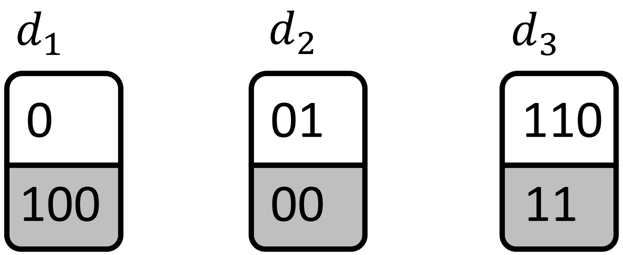
- ***U_{TM}* outputs on $< M > \#w$ whatever *M* outputs on *w*. *U_{TM}* simulates *M***
- U_{TM} is fixed but can simulate any M, even one with a million states



- **Entire simulation is done on the tape!**

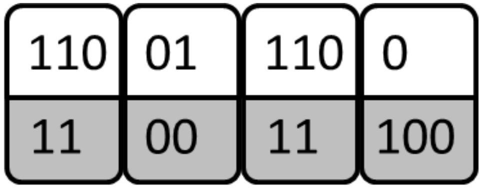
Post's Correspondence Problem (PCP)

- **PCP**: Consider 3 dominos



- Can I arrange dominos (using multiple copies of each) so that top and bottom strings match

$d_3 d_2 d_3 d_1 =$



- INPUT: Dominos $\{d_1, d_2, \ldots, d_n\}$.

- TASK: Can one line up finitely many dominos so that the top and bottom strings match?

HalfSum

- Consider the multiset $S = \{1,1,1,3,4,4,5,6,9\}$ and subset $A = \{1,3,4,9\}$

$$sum(A) = 17 = \frac{1}{2} \times sum(S)$$

- INPUT: Multiset $S = \{x_1, x_2, \dots, x_n\}$. For example, $S = \{1,1,1,3,4,4,5,6,9\}$
- TASK: Is there a subset whose sum is $\frac{1}{2} \times sum(S) = \frac{1}{2} \times (x_1 + x_2 + \cdots + x_n)$?

Auto-Grade

- **Your first CS assignment:** Write a program to print "Hello World!" and halt.

- **CS1**: 700+ submissions!
- Naturally, we do not grade these by hand.
- Auto-Grade: runs each submission and determines if it is correct.
  - **program verification**
  - (more like testing really – verification in general means *proving* your program is correct over all possible inputs)
- What does Auto-Grade say for this program:

```
n = 4;
while(n > 0){
  if(n is not a sum of two primes){
    print("Hello World!") and exit;
  }
  n ← n + 2;
}
```

Ultimate-Debugger

- Wouldn't it be nice to have the Ultimate-Debugger
  - Would solve the *Halting Problem*

$$\textsc{Halts}\left(\boxed{\text{program}}\right) = \begin{cases} \boxed{\text{YES}} & \text{if program halts} \\ \boxed{\text{NO}} & \text{if program infinitely loops} \end{cases}$$

```
n = 4;
while(n > 0){
 if(n is not a sum of two primes){
  print("Hello World!") and exit;
 }
n ← n + 2;
}
```

- We can grade the students program correctly.
- We can solve Goldbach's conjecture.
- Just think what you could do with Ultimate-Debugger.
  - No more infinite looping programs.

Does a Program Successfully Terminate?

- Let's define the language of all Turing Machines that accept a string
$$\mathcal{L}_{TM} = \{< M > \#w | M\ is\ a\ Turing\ Machine\ and\ M\ accepts\ w\}$$
- Our simulator U_{TM} is a *recognizer* for $\mathcal{L}_{TM}$
- Is there a Turing Machine A_{TM} which **decides** $\mathcal{L}_{TM}$?
  - A decider must *always* halt with an answer
  - U_{TM} may loop forever if M loops forever on w
  - Question: What do these mean: $M(< M >)$ and $A_{TM}(< M > \# < M >)$?

- A diabolical Turing Machine D built from A_{TM}:

  D: "Diagonal" Turing Machine derived from A_{TM} (the decider for $\mathcal{L}_{TM}$)

  input: $< M >$ where M is a Turing Machine

  1. Run A_{TM} with input $< M > \# < M >$
  2. If A_{TM} accepts then REJECT; otherwise (A_{TM} rejects) ACCEPT

- D does the *opposite* of A_{TM}. Is D a decider?

Theorem. A_{TM} does not exist ($\mathcal{L}_{TM}$ Cannot be Solved)

- First note that if A_{TM} exists, then D must exist
- If D exists, then it will appear on the list of all Turing Machines:

$$< M_1 >, < M_2 >, < M_3 >, < M_4 >, < D >, \ldots$$

- Consider what happens when M_i runs on $< M_j >$, that is $A_{TM}(< M_i > \# < M_j >)$

$A_{TM}(< M_i > \# < M_j >)$	$< M_1 >$	$< M_2 >$	$< M_3 >$	$< M_4 >$	$< D >$
$< M_1 >$	ACCEPT	ACCEPT	REJECT	ACCEPT	ACCEPT
$< M_2 >$	ACCEPT	REJECT	REJECT	ACCEPT	REJECT
$< M_3 >$	REJECT	ACCEPT	REJECT	REJECT	ACCEPT
$< M_4 >$	REJECT	ACCEPT	REJECT	ACCEPT	REJECT
$< D >$					

Theorem. A_{TM} does not exist ($\mathcal{L}_{TM}$ Cannot be Solved)

Rensselaer

- First note that if A_{TM} exists, then D must exist
- If D exists, then it will appear on the list of all Turing Machines:

$$< M_1 >, < M_2 >, < M_3 >, < M_4 >, < D >, \ldots$$

- Consider what happens when M_i runs on $< M_j >$, that is $A_{TM}(< M_i > \# < M_j >)$

$A_{TM}(< M_i > \# < M_j >)$	$< M_1 >$	$< M_2 >$	$< M_3 >$	$< M_4 >$	$< D >$
$< M_1 >$	ACCEPT	ACCEPT	REJECT	ACCEPT	ACCEPT
$< M_2 >$	ACCEPT	REJECT	REJECT	ACCEPT	REJECT
$< M_3 >$	REJECT	ACCEPT	REJECT	REJECT	ACCEPT
$< M_4 >$	REJECT	ACCEPT	REJECT	ACCEPT	REJECT
$< D >$	REJECT	ACCEPT	ACCEPT	REJECT	???

- $D(< M_i >)$ does the opposite of $A_{TM}(< M_i > \# < M_j >)$

Theorem. A_{TM} does not exist ($\mathcal{L}_{TM}$ Cannot be Solved)

- $D(< M_i >)$ does the opposite of $A_{TM}(< M_i > \# < M_j >)$
- Suppose $A_{TM}(< D > \# < D >)$ accepts
  - That means that D accepted the input $< D >$
  - But then D should reject $< D >$ because $A_{TM}(< D > \# < D >)$ accepted
  - **FISHY**!
- Suppose $A_{TM}(< D > \# < D >)$ rejects
  - That means that D rejected the input $< D >$
  - But then D should accept $< D >$ because $A_{TM}(< D > \# < D >)$ rejected
  - **FISHY**!
- This proof should remind you of Cantor's diagonalization argument
- It's essentially the same idea: we have countably many Turing Machines but uncountably many functions
  - Some functions cannot be implemented by a Turing machine
  - One of the most important results in the theory of computation!

Ultimate-Debugger and Auto-Grade Don't Exist

- No *general* program/algorithm to analyze *any* other program M and tell if M will accept or not a particular input.

  ☹ ☹ ☹
  - No Ultimate-Debugger
  - No Auto-Grade for CS-1 Programs
  - No solver for PCP
- Suppose Ultimate-Debugger H_{TM} exists and *decides* if any other program halts
- We can use H_{TM} to construct a solver A_{TM} for $\mathcal{L}_{TM}$

  A_{TM}: Turing Machine derived from H_{TM} (the decider for $\mathcal{L}_{HALT}$)

  input: $< M > \#w$ where M is a Turing Machine and w an input to M
  1. Run H_{TM} with input $< M > \#w$. If H_{TM} rejects, then REJECT
  2. Run U_{TM} with input $< M > \#w$ and output the decision U_{TM} gives
- This problem is known as the *halting problem*!
- **Exercise.** Show that Auto-Grade does not exist.
- **Exercise.** Show that HalfSum is solvable by giving a decider.

Non-Recognizable Languages

- How many recognizable languages are there?
  - At most countable, since every recognizable language must be recognized by a corresponding TM
- Is A_{TM} a recognizer?
  - Yes, it halts when M halts on $< w >$ and accepts
- So some undecidable languages are recognizable!
  - But countably many
- What does that mean about the remaining computing problems?
  - They must be non-recognizable
  - There are uncountable many non-recognizable languages!
  - Most languages are not recognizable!
- Is CS useless?
  - Many decidable problems are in fact useful (sorting, shortest path, etc.)
    - Plus, we can force halting by setting a limit on computation time
  - The next challenge is how fast can we solve the problems that we know are solvable (algorithms course)!

The Landscape