- Start as early as possible, and contact the instructor if you get stuck.
- See the course outline for details about the course's grading policy and rules on collaboration.
- Submit your completed solutions to **Crowdmark**.

1. **Descriptions of natural numbers [6 marks]**

   Definition 1. Given an integer $n \geq 0$, the *exponential-Golomb encoding* of n is

   $$\langle n \rangle := \mathtt{0}^{|x|}\mathtt{1}x^R$$

   where $x \in \{\mathtt{0}, \mathtt{1}\}$ is such that $\mathtt{1}x$ is the binary representation of $n+1$.

   There exists a Turing machine M which on input $\langle u \rangle \langle v \rangle$ for all integers $u, v \geq 0$, inserts a separator between the representations,

   $$\langle u \rangle \mathtt{\#} \langle v \rangle,$$

   and then halts. Explain the low-level steps the Turing machine does to accomplish this. You may extend the tape alphabet in your Turing machine if it helps.

   Solution

   (a) Scan right to the first $\mathtt{1}$ and change it to a $\overline{\mathtt{1}}$.
   (b) "Cancel" each symbol before the $\overline{\mathtt{1}}$ with one after it by marking the symbols with bars.
      i. Scan right to the first symbol that does not have a bar and add a bar to it. (Unless it's a $\square$—then something has gone wrong and we reject)
      ii. Scan left to the first symbol that does not have a bar.
         - If it's a $\mathtt{0}$ or $\mathtt{1}$, add a bar and repeat.
         - If it's a $\square$ then break out of the loop.
   (c) Scan right to the first symbol that does not have a bar.
   (d) Replace it with a $\mathtt{\#}$ and shift the rest of the string down.
   (e) As we shift, if we reach a $\square$, turn around and scan to the left, replacing

   $$\overline{\mathtt{0}} \mapsto \mathtt{0}, \qquad \overline{\mathtt{1}} \mapsto \mathtt{1}.$$

   (f) When we reach $\square$, take one step back to the right and halt.

2. **Red-Blue Turing machines [12 marks]**

   For the 90^{th} anniversary of the invention of the Turing machine, they are introducing a special limited-edition *red-blue Turing machine* (RBTM). A red-blue Turing machine has the following additional rules:

   - Every state is coloured red or blue.
   - The initial state, q_0, is red and the unique final state, q_{finish}, is blue.
   - There are no transitions from a blue state to a red state.

   As a result, there are three outcomes for a RBTM: it loops forever on the red states (Red), it loops forever on the blue states (Blue), or it halts (Halt). Coincidentally, a standard Turing machine also has three outcomes: halt and accept (Acc), halt and reject (Rej), or loop forever (Loop).

(a) Prove the following theorem (an RBTM can simulate a TM).

Theorem 2. *There exists a bijection* $f\colon \{\mathsf{Red}, \mathsf{Blue}, \mathsf{Halt}\} \to \{\mathsf{Acc}, \mathsf{Rej}, \mathsf{Loop}\}$ *and a red-blue Turing machine* M' *with the property that on all inputs* $\langle M\rangle x$ *(where* M *is a Turing machine and* $x \in \{0,1\}^*$ *is a bitstring), the outcome of* M' *is* $o \in \{\mathsf{Red}, \mathsf{Blue}, \mathsf{Halt}\}$ *if and only if* M *does* $f(o)$ *on input* x*.*

Solution

Let the bijection be

$$f(\mathsf{Red}) = \mathsf{Loop}, \qquad f(\mathsf{Blue}) = \mathsf{Acc}, \qquad f(\mathsf{Halt}) = \mathsf{Rej}.$$

Let M' be a universal Turing machine made mostly of red states. On input $\langle M\rangle x$, it simulates the Turing machine M on x. If M rejects, then we have M' transition directly to the Halt state ($f(\mathsf{Halt}) = \mathsf{Rej}$). If M accepts, then we have it transition to a blue state that loops forever ($f(\mathsf{Blue}) = \mathsf{Acc}$). Finally, if M does not loop then it remains in the universal simulator (consisting of red states) forever ($f(\mathsf{Red}) = \mathsf{Loop}$).

(b) Prove the following theorem (a TM *cannot* simulate an RBTM).

Theorem 3. *For each bijection* $g\colon \{\mathsf{Acc}, \mathsf{Rej}, \mathsf{Loop}\} \to \{\mathsf{Red}, \mathsf{Blue}, \mathsf{Halt}\}$*, there does not exist a Turing machine* M *such that on input* $\langle M'\rangle x$ *(where* M' *is a RBTM and* $x \in \{0,1\}^*$*), the outcome is* $o \in \{\mathsf{Acc}, \mathsf{Rej}, \mathsf{Loop}\}$ *exactly when* M' *does* $g(o)$ *on input* x*.*

Solution

Suppose to the contrary that for some bijection g, there is a Turing machine M that accepts, rejects, or loops on input $\langle M'\rangle x$ when on input x, red-blue Turing machine M' has outcome $g(\mathsf{Acc})$, $g(\mathsf{Rej})$, or $g(\mathsf{Loop})$ respectively. Recall that $\mathsf{Halt}_{\mathrm{TM}}$ is undecidable, where

$$\mathsf{Halt}_{\mathrm{TM}} = \{\langle T\rangle x \mid T \text{ halts on } x\}.$$

We claim that we can use M to solve the halting problem.[a] Now consider $g(\mathsf{Acc})$ and $g(\mathsf{Rej})$. There's a little case analysis:

- If $g(\mathsf{Rej}) = \mathsf{Red}$ and $g(\mathsf{Acc}) = \mathsf{Halt}$ then given $\langle T\rangle$, construct a RBTM T' by colouring all non-final states red. Then run M on $\langle T'\rangle x$ and note that

$$\begin{aligned} M \text{ accepts } \langle T'\rangle x &\iff T' \text{ halts on } x \\ &\iff T \text{ halts on } x \\ &\iff \langle T\rangle x \in \mathsf{Halt}_{TM}. \end{aligned}$$

- If $g(\mathsf{Rej}) = \mathsf{Red}$ and $g(\mathsf{Acc}) = \mathsf{Blue}$ then given a Turing machine $\langle T\rangle$, construct T' by colouring all non-final states red, but colouring all

final states blue and making them loop. Then run M on $\langle T' \rangle x$ and observe

$$\begin{aligned} M \text{ accepts } \langle T' \rangle x &\iff T' \text{ loops in } \mathsf{Blue} \text{ on } x \\ &\iff T \text{ halts on } x \\ &\iff \langle T \rangle x \in \mathsf{Halt}_{TM}. \end{aligned}$$

- If $g(\mathsf{Rej}) = \mathsf{Blue}$ and $g(\mathsf{Acc}) = \mathsf{Halt}$ then given $\langle T \rangle$, construct a RBTM T' by colouring all non-final states blue. Also, we add a new initial state r_0 and a helper r_1, both red, for the purpose of doing nothing (i.e., take a step right, then take a step left) and transitioning to q_0, the original initial state. We see that when we run M on $\langle T' \rangle x$,

$$\begin{aligned} M \text{ accepts } \langle T' \rangle x &\iff T' \text{ halts on } x \\ &\iff T \text{ halts on } x \\ &\iff \langle T \rangle x \in \mathsf{Halt}_{TM}. \end{aligned}$$

- Finally, if g is any of the above but with the roles of Acc and Rej reversed, then we *reject* when M halts on x.

In all cases we have a computable transformation $T \to T'$ from a Turing machine T to a red-blue Turing machine T'. If we could simulate T' then we would also decide the halting problem, which is a contradiction. Therefore, there is no Turing machine which simulates a RBTM in this bijective way.

[a] A reduction, but unfortunately not the $\leq_m$ kind of reduction.

(c) Discuss (two or three sentences) how this relates to the Church-Turing thesis.

Solution

Part (b) looks like an example of a physically realizable machine that provably computes things Turing machines can't. However, the Church-Turing thesis is about deciding things, and especially getting a definitive answer in finite time. You can't tell whether the outcome will be red or blue in finite time, so the fact that you cannot simulate it does not contradict the Church-Turing thesis.

3. **Logic and Computability [8 marks]**

Fix a logical system, like Zermelo-Fraenkel set theory (ZF). The details of the theory don't matter as long as we have three facts.

- Theorems and proofs can be represented as bitstrings.
- There is a Turing machine $\mathsf{ProofChecker}$ which given a theorem and a proof will check if the proof correctly proves the theorem.
- The system is consistent – one cannot prove a contradiction.

Famously, it is not possible to prove the consistency of a theory like ZF within the theory itself, so we have to take the last point on faith.

Assuming the consistency of ZF, prove the following theorem.

Theorem 4. *There exists a TM which loops **on the empty string**, but there is no* ZF *proof it loops.*

Hint: You can run more than just two computations in parallel.

Solution

Construct a machine M which on any input does the following.

(a) Obtain the description of $\langle M \rangle$ by recursion theorem.

(b) Enumerate all possible bistrings w. For each one:

- Run ProofChecker on w and the statement "M loops on ε".
- If any valid proof is found, halt.

If M does not loop on ε, then by definition it halts because it has found a proof that M loops on ε. If ZF is logically consistent, then it must be that M loops on ε. That is a contradiction, so M actually does loop on ε.

However, M also searches for a proof of the claim "M loops on ε". Since we know M loops, it never finds a proof. It *exhaustively* searches all possible proofs, so if it never finds one then none exists.

Extra

Note that the solution above does not run any computations in parallel. I apologize for the bad hint. Parallel computations are only helpful if ProofChecker can only *recognize* valid proofs, but it's far more natural to assume it *decides* the validity of a proof, and the assignment wasn't clear which one was intended.

Extra

This result is essentially one of Gödel's incompleteness theorems: in a sufficiently powerful (and consistent) logical system, there exist statements that are true but have no proof. The result actually predates Turing's paper, and he cited it noting it had "superficially similar" conclusions.

4. **States vs. Description Length [10 marks]**

Turing machines are unusual because we often measure a TM's size by the *number of states*[1] but this does not coincide with the length of the description, not even up to constant factors (i.e., big-Θ).

For this question, suppose the tape alphabet $\Gamma = \{\square, \mathtt{0}, \mathtt{1}\}$ is fixed, along with $b = \square$ and $\Sigma = \{\mathtt{0}, \mathtt{1}\}$. Write the states $Q = \{0, 1, \ldots, n, n+1\}$ as integers, with $q_0 = 0$ being the first state and $F = \{q_{accept}, q_{reject}\} = \{n, n+1\}$ being the last two.

(a) Given n, the only part of the Turing machine left to specify is $\delta \colon (Q \backslash F) \times \Gamma \to Q \times \Gamma \times \{L, R\}$, the transition function. Argue that there is a description $\langle \delta \rangle \in \{\mathtt{0}, \mathtt{1}\}^*$ for δ (and thus the TM) of length $O(n \log n)$.

[1] Busy beavers, for example, are classified by number of states.

Solution

The simplest way to describe a function δ is a description of each output, written as a list, ranging over all inputs in some predefined order. Say,

$$\langle\delta\rangle = \langle\delta(0,\mathtt{0})\rangle\langle\delta(0,\mathtt{1})\rangle\langle\delta(0,\square)\rangle\cdots\langle\delta(n+1,\mathtt{0})\rangle\langle\delta(n+1,\mathtt{1})\rangle\langle\delta(n+1,\square)\rangle.$$

There are n states in $Q\backslash F$ and three symbols in Γ, so that's $3n$ descriptions of $Q\times\Gamma\times\{L,R\}$. We can express the state as a number from 0 to $n+1$, which by the encoding in Q1 uses $O(\log n)$ bits. The tape symbol can be expressed with two bits, e.g.,

$$\langle\mathtt{0}\rangle = \mathtt{00}, \qquad \langle\mathtt{1}\rangle = \mathtt{01} \qquad \langle\square\rangle = \mathtt{10},$$

and obviously one more bit encodes the direction: $\langle L\rangle = \mathtt{0}$ and $\langle R\rangle = \mathtt{1}$. Altogether, we have $3n$ entries of length $O(\log n)$ bits for a total of $O(n\log n)$ bits in our description.

(b) On the other hand, prove that the description must be at least $\Omega(n\log n)$ bits long because there are $2^{\Omega(n\log n)}$ *semantically* different Turing machines. That is, there is a set of $2^{\Omega(n\log n)}$ Turing machines and for every pair there exists an input where they differ (one accepts and the other rejects, or loops, etc.)
Hint: DFAs are simple Turing machines that never change the tape symbol and always move left.

Solution

I'm going to pick a set of Turing machines that also work for part (c), but there are many other choices of TM one could take. Formally, for any function $h\colon[n]\to[n]$,[a] we define a TM M_h. Let M_h move to the right on any input symbol and output the same symbol. Let's say if it reads a $\square$ then it transitions to $\mathsf{Rej} = n+2$, if it reads a $\mathtt{0}$ then it increments from state q to $q+1$, and if it reads a $\mathtt{1}$ it transitions according to h. That is, for $q = 0,\ldots,n-1$ we have

$$\begin{aligned}\delta(q,\square) &= (n+2,\square,R),\\ \delta(q,\mathtt{0}) &= (q+1,\mathtt{0},R),\\ \delta(q,\mathtt{1}) &= (h(q),\mathtt{1},R).\end{aligned}$$

Given distinct functions $h,h'\colon[n]\to[n]$, we argue M_h and $M_{h'}$ are different by looking at inputs of the form $\mathtt{0}^i\mathtt{1}\mathtt{0}^j$. Specifically, suppose $h(i)\neq h'(i)$ and $j = n-\max\{h(i),h'(i)\}$. When they read $\mathtt{0}^i$, both machines will transition to state i, and then reading $\mathtt{1}$ sends them to $h(i)$ or $h'(i)$ respectively. Since $h(i)\neq h'(i)$, it then takes a different number of increment steps to reach state n. We choose j such that $\mathtt{0}^j$ is *just* enough to increment one machine (either M_h or $M_{h'}$) to state n, while the other machine reads a $\square$ and transitions to $n+1$ instead. Hence, two arbitrary machines disagree on a

specific input and are not interchangeable.
Since every one of the machines in

$$\{M_h \mid h\colon [n] \to [n]\}$$

is functionally different from the others, they all deserve distinct descriptions. There are $n^n = 2^{\Omega(n \log n)}$ total machines in the set. On the other hand, there are $\leq 2^{\ell+1}$ strings of length $\leq \ell$, so the longest description must have length $\max_h |\langle M_h \rangle| = \ell = \Omega(n \log n)$ by pigeonhole principle. Similarly, half of the descriptions are longer than the average (over all M_h), which must therefore also be $\mathbb{E}[|\langle M_h \rangle|] = 2^{\Omega(n \log n)}$.

[a]Define $[n] = \{0, 1, \ldots, n-1\}$.

(c) In the lecture on the recursion theorem, we saw that for any bitstring $s \in \{0,1\}^*$ there is a *printer* P_s which prepends s to the input string. We also saw FindPrinter, which replaced input s with $\langle P_s \rangle$. The printer used $|s| + 3$ states, but we now know that $\langle P_s \rangle$ is asymptotically longer than the string P_s prints.
Prove that there exists a family of printers $\{P'_s\}_{s \in \{0,1\}^*}$, and a FindPrinter′ which generates $\langle P'_s \rangle$ given s, where the length of $\langle P'_s \rangle$ is $O(|s|)$.

Solution

Author's note: This question was intended to be a bit more of a puzzle than the others. It'll be worth ***4 marks*** *and leniently graded.*

We have just seen that there is not a one-to-one conversion between states and bits, contrary to what one might expect. Now we want to leverage this to get an interesting result: there exist printers for length-n bitstrings that have $o(n)$ states.
The idea is to spend $n - O(1)$ states on, essentially, one of the machines from part (b) (the *data*). The remaining $O(1)$ states (the *extractor*) will be designed to systematically extract the data from that machine and print it as a string. More formally, the extractor will be a Turing machine with one special final state (q_{gosub}) and one special non-final state (q_{return}). Given a TM M_h, we splice it into E as follows to obtain a new TM $E[M_h]$.

- Redirect all transitions in E into state q_{gosub} so that they point to state 0 in M_h, and
- redirect transitions into final states of M_h so that they point to q_{return}.

We claim that this can be used to print arbitrary strings.
Claim 5. *For any $m \in \mathbb{N}$, there exists a machine E_m with $O(1)$ states such that for any string s of length $O(m \log m)$, there exists a function $h\colon [m] \to [m]$ such that $E_m[M_h]$ outputs s on the empty input, where M_h is the family of TMs from part (b).*

Let us design the "extractor" part (E_m) first. We extract data from M_h by preparing a configuration like

$$\mathtt{1}q_{\text{gosub}}\mathtt{0}^i\mathtt{10}^m. \tag{1}$$

When we consider $E_m[M_h]$, the q_{gosub} will be replaced with state 0 of M_h. We know M_h will transition $0 \to 1 \to \cdots \to i$ as it reads $\mathtt{0}^i$, then to $h(i)$ reading the $\mathtt{1}$, and then $h(i) \to h(i+1) \to \cdots m$ as it reads more $\mathtt{0}$s. This leaves it in a configuration

$$\mathtt{10}^i\mathtt{10}^{m-j}q_{\text{return}}\mathtt{0}^j. \tag{2}$$

From there, we do the following steps:

- erase the trailing $\mathtt{0}^j$,
- count the number of zeros in $\mathtt{0}^{m-j}$ in binary (e.g., the Golomb encoding) until we reach $\mathtt{1}$, and erase them as we go,
- erase back to the preceding $\mathtt{1}$, leaving

$$\langle m - j\rangle \tag{3}$$

on the tape in place of (2), or (1) before that.

Similarly, we can prepare a configuration like

$$\mathtt{11}q_{\text{gosub}}\mathtt{0}^p \tag{4}$$

and M_h will return to use the configuration $\mathtt{110}^{p-k}q_{\text{return}}\mathtt{0}^k$, since as long as $p \geq m$, reading the zeros will cause M_h to transition through $0 \to 1 \to \cdots \to m$ and halt. Since q_{return} is the same state as above, we are bound to do the same post-processing: erase everything and replace it with $\langle m\rangle$. On the other hand, if $p < m$ then it will halt due to $\square$ and the same steps will lead to $\langle p+1\rangle$ instead.

With that in mind, E proceeds as follows. It constructs configurations ending in (4) for larger and larger p until the answer is not $\langle p+1\rangle$, and therefore must be $\langle m\rangle$. Using knowledge of m, it loops over $i = 0, \ldots, m-1$ and constructs configurations ending in (1) to learn $\langle m - h(i)\rangle$. Each $\langle m - h(i)\rangle$ has $\log_2 m$ bits of information which we then concatenate into an $m \log_2 m$ bit string, and leave on the tape.

Since the string might not be exactly $m \log_2 m$ bits long, the last step is to print a number at the beginning of the string representing its length. We use the number to trim the string to the desired length and then erase the number, leaving only the string. As a result, $m \log_2 m$ only needs to be *larger* than $|s| + \Omega(\log |s|)$, i.e., we need $m \log_2 m = |s| + \Omega(\log |s|)$. That's the size of the data, and then we need the $O(1)$ size extractor on top of that: $n = m + O(1)$. It follows that the description of the machine has length

$$|\langle E_m[M_h]\rangle| = O(n \log n) = O(m \log m) = O(|s|).$$

Finally, we observe that all the steps of this construction are effective—the extractor is fixed and we could work backwards to figure out what h needs to be to print a desired string. Thus $\mathsf{FindPrinter}'$ exists.