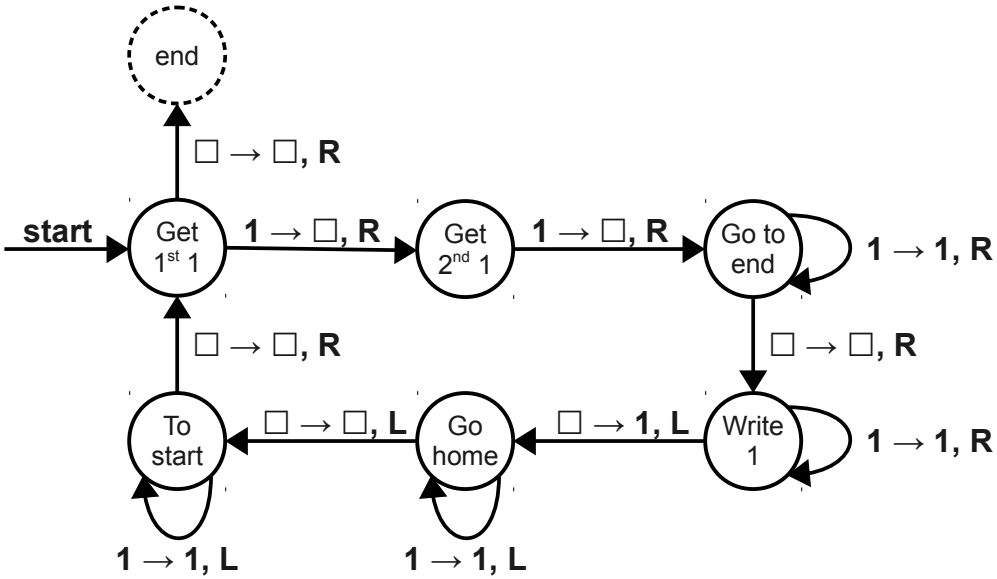
Problem Set 7 Solutions

Problem One: The Collatz Conjecture (24 Points)

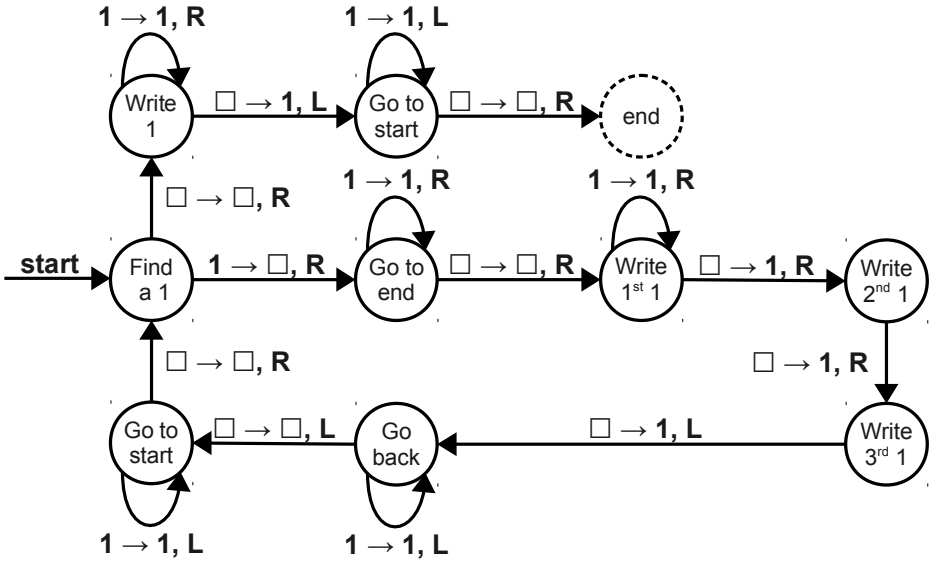
i. Draw the state transition diagram for a Turing machine that, when given a tape holding $\texttt{1}^{2n}$ surrounded by infinitely many blanks, ends with $\texttt{1}^{n}$ written on its tape, surrounded by infinitely many blanks.

Here is one possible TM:



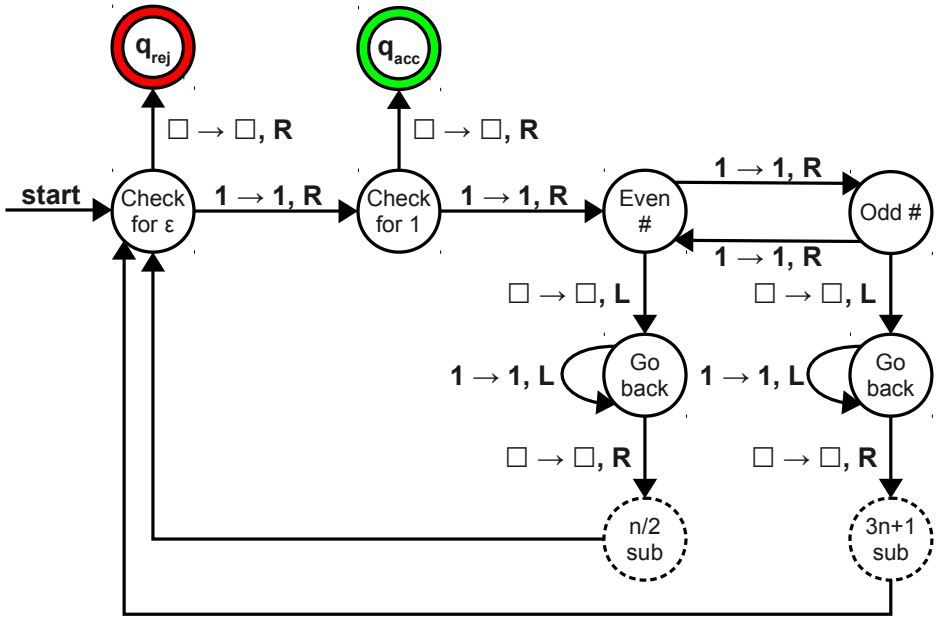
This TM works by checking if no more `1`s are left, and, if so, stopping. Otherwise, it crosses of two `1`s, skips to the end of the input, and then appends a single `1` to a new group of `1`s it builds up at the end of the input.

ii. Draw the state transition diagram for a Turing machine that, when given a tape holding $\mathtt{1}^n$ surrounded by infinitely many blanks, ends with $\mathtt{1}^{3n+1}$ written on its tape, surrounded by infinitely many blanks.



This TM works by repeatedly crossing off a `1`, then writing three more `1`s at the end. Once all `1`s have been exhausted, the TM then writes a final `1` to the end.

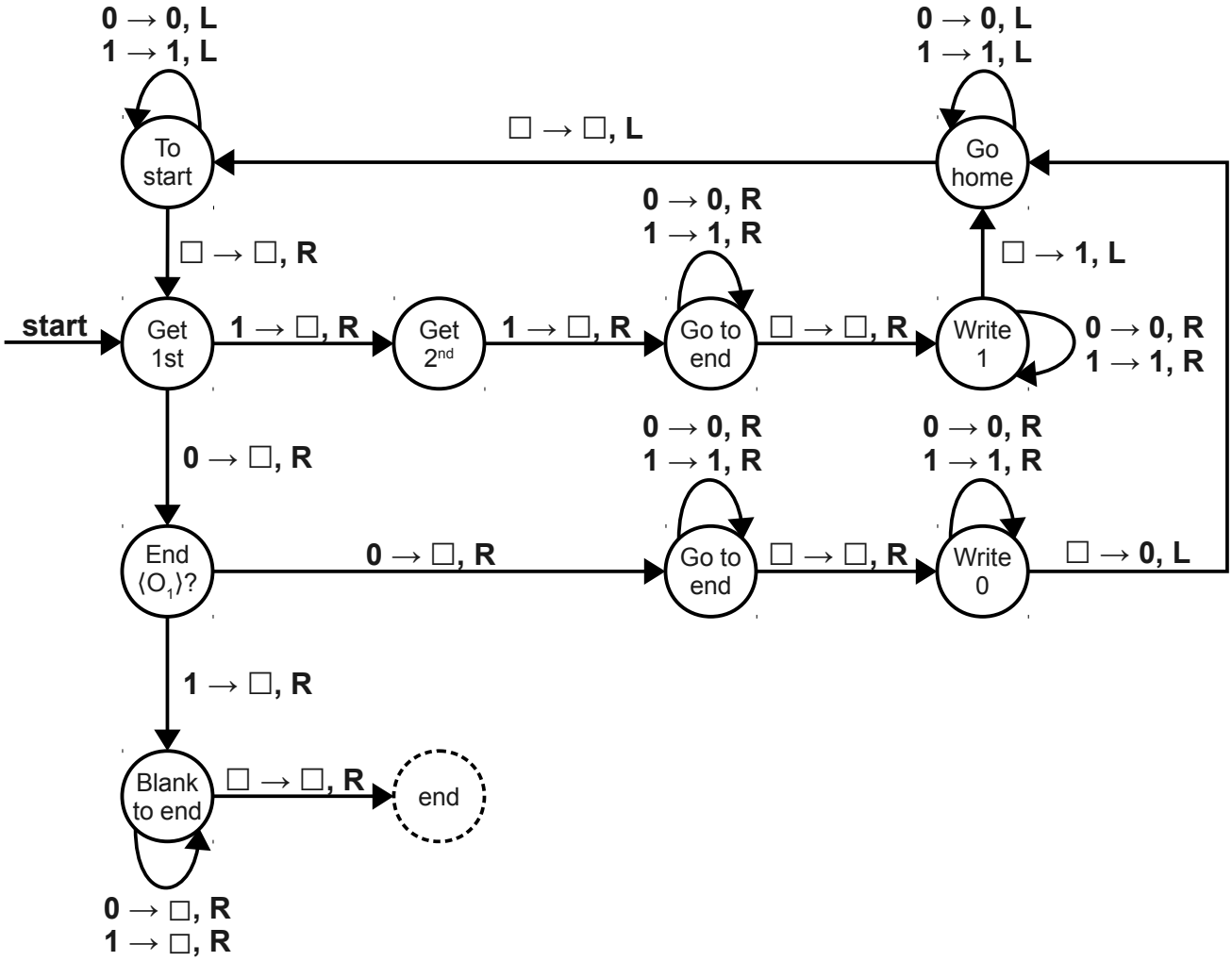
iii. Using your TMs from parts (i) and (ii) as subroutines, draw the state transition diagram for a Turing machine M that recognizes L.



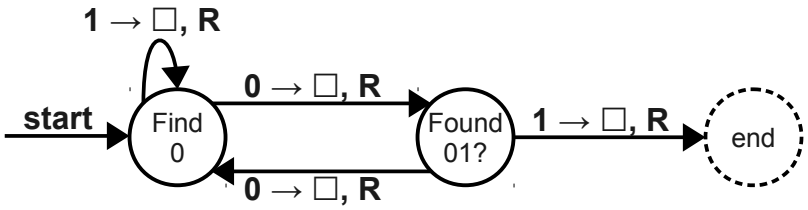
Why we asked this question: Parts (i) and (ii) of this problem require you to build a TM that can multiply or divide by a constant, and we hoped that it would give you a sense for how to build TMs that can do arithmetic operations. Part (iii) of this problem includes a construction for testing the parity of a string, as well as two spots where a subroutine TM should be snapped in. We hoped that this would help you see how to design complex TMs. Plus, there's something nifty about designing a TM whose behavior on all inputs isn't fully understood!

Problem Two: Manipulating Encodings (16 Points)

i. Draw the state transition diagram for a Turing machine that, given an encoding $\langle O_1, O_2 \rangle$ of two objects, ends with the string $\langle O_1 \rangle$ written on its tape, surrounded by infinitely many blanks.



ii. Draw the state transition diagram for a Turing machine that, given an encoding $\langle O_1, O_2 \rangle$ of two objects, ends with the string $\langle O_2 \rangle$ written on its tape, surrounded by infinitely many blanks.



Why we asked this question: Unlike the TMs from Problem One, the TM for part (i) of this problem requires you to build constant storage into the finite-state control of the Turing machine. We figured it would be good for you to get your feet wet putting this together. Additionally, we wanted you to see in part (ii) that complex TM behaviors can often be expressed quite simply.

Problem Three: Finding Flaws in Proofs (12 Points)

One counterexample would be the following languages and machines:

$L_1 = \varnothing$ $\qquad$ $L_2 = \Sigma^*$

M_1 = "On input w, loop infinitely." $\qquad$ M_2 = "On input w, accept."

In this case, the generated machine will always loop infinitely, since upon running M_1 the machine goes into an infinite loop. Thus $L(M) = \varnothing \neq \varnothing \cup \Sigma^*$.

The flaw in the proof is that if $w \notin M_1$, then M_1 doesn't necessarily have to reject. It could loop infinitely as well. Consequently, although it's true that M accepts w iff M_1 accepts w or M_1 rejects w and M_2 accepts w, this doesn't mean that M accepts w iff M_1 accepts w or if M_1 does not accept w and M_2 accepts w.

Why we asked this question: This question was all about the possibility that a TM can loop forever as a way of not accepting. The construction works fine as long as the first TM is a decider, but might fail if it loops on some input. We wanted you to both figure out why the machine was incorrect and to see where the logic in the proof broke down so that you could see how the possibility that a TM can loop factors into rigorous proofs about Turing machines.

Problem Four: Nondeterministic Algorithms (20 points)

i. Prove that **RE** is closed under union. That is, if $L_1 \in$ **RE** and $L_2 \in$ **RE**, then $L_1 \cup L_2 \in$ **RE** as well. Your proof can proceed along the lines of the ones we covered in lecture, so feel free to use high-level descriptions of Turing machines rather than formally writing out transition tables.

Proof: Let L_1 and L_2 be **RE** languages and let R_1 and R_2 be recognizers for them. Then construct the following NTM N:

> N = "On input w:
> Nondeterministically choose to run either R_1 or R_2 on w.
> If the chosen machine accepts w, then N accepts w.
> If the chosen machine rejects w, then N rejects w.

We claim that $\mathscr{L}(N) = L_1 \cup L_2$. To see this, we prove that there is a nondeterministic choice such that N accepts w iff $w \in L_1 \cup L_2$. We prove both directions of implication. First, if there is a nondeterministic choice such that N accepts w, note that either R_1 accepts w, in which case $w \in L_1$, or R_2 accepts w, in which case $w \in L_2$. In either case, $w \in L_1 \cup L_2$. Second, suppose that $w \in L_1 \cup L_2$. Without loss of generality, assume that $w \in L_1$. Then N can choose to run R_1 on w, which will accept w because R_1 is a recognizer for L_1. Consequently, N accepts. ■

ii. Prove that the **RE** languages are closed under concatenation. That is, if $L_1 \in$ **RE** and $L_2 \in$ **RE**, then $L_1L_2 \in$ **RE** as well.

Proof: Let L_1 and L_2 be **RE** languages and let R_1 and R_2 be recognizers for them. Then construct the following NTM N:

> N = "On input w:
> Nondeterministically guess two strings x and y.
> If $w \neq xy$, reject.
> Run R_1 on x.
> If R_1 rejects, reject.
> If R_1 accepts, run R_2 on y.
> If R_2 accepts, accept; if R_2 rejects, reject."

We claim that $\mathscr{L}(N) = L_1L_2$. To see this, we prove that there is a nondeterministic choice of x and y such that N accepts w iff $w \in L_1L_2$. First, if there is a nondeterministic choice of x and y such that N accepts w, then this means that R_1 accepts x, so $x \in L_1$, and R_2 accepts y, so $y \in L_2$, and $w = xy$. Consequently, $w \in L_1L_2$. Second, if $w \in L_1L_2$, then there exists $x \in L_1$, $y \in L_2$ such that $w = xy$. Thus the nondeterministic choice of x and y corresponding to these strings will cause N to accept, since R_1 will accept x and R_2 will accept y. ■

Why we asked this question: Nondeterministic Turing machines are tricky to design; programming them requires a very different perspective than normal problem solving. We wanted this question to introduce you to the techniques underlying the design of NTMs. We also hoped it would help you get a better intuition for how to design nondeterministic machines.

Problem Five: R and RE Languages (24 Points)

i. Give a high-level description of a TM M such that $\mathscr{L}(M) \in$ **R**, but M is not a decider. This shows that just because a TM's language is decidable, it's not necessarily the case that the TM itself must be a decider.

One option is M = "On input w, loop infinitely." This TM has language Ø, since it never accepts, and Ø is decidable because it is regular. However, M is not a decider, because M never halts.

ii. Only *languages* can be decidable or recognizable; there's no such thing as an "undecidable string" or "unrecognizable string." Explain why there is no string w such that any language that contains w is undecidable and no string w such that any language that contains w is unrecognizable (a short paragraph should be sufficient). This result is important – the reason that languages become undecidable or unrecognizable is that there is no TM that can always give back the correct answer for *every* string in the language, not because there is some "bad string" that makes the language undecidable or unrecognizable.

All strings belong to Σ^*, which is decidable because it's a regular language. Since $\Sigma^* \in$ **R** and **R** $\subseteq$ **RE**, all strings also belong to at least one recognizable language. Therefore, all strings belong to at least one decidable language and at least one recognizable language.

iii. Prove that for every language L, there is a decider M^+ that accepts every string in L and a decider M^- that rejects every string not in L. Explain why this result doesn't prove that every language is in **R**.

Proof: Let M^+ be the TM M^+ = "On input w, accept" and let M^- be the TM M^- = "On input w, reject." Then M^+ and M^- are deciders, since they always halt. Moreover, M^+ accepts all strings in L (since it accepts all strings) and M^- rejects all strings not in L (since it rejects all strings), as required. ■

This doesn't prove that all languages are decidable. For a language to be decidable, there must be a single TM M where M accepts all strings $w \in L$ and rejects all strings $w \in L$. Both claims must hold at the same time, whereas here M^+ and M^- only satisfy one of the claims.

iv. Show that there is a TM M with the following properties: $\mathscr{L}(M)$ is an infinite subset of A_{TM}, but M is a decider. That is: M accepts infinitely many strings of the form $\langle N, w \rangle$, where N is a TM that accepts string w, yet the machine M is a decider. Prove that your machine has the required properties. This shows that even though A_{TM} is undecidable, it is still possible to build a TM that will decide A_{TM} for infinitely many inputs.

There are several solutions to this problem. Here is one based on the idea that you can hardcode a machine into the decider:

Proof: Let A = "On input w, accept." Then consider the following TM:

> D = "On input $\langle M, w \rangle$, where M is a TM and w is a string:
> If $M \neq A$, reject.
> If $M = A$, accept."

We claim that $\mathscr{L}(D) = \{ \langle A, w \rangle \mid w \in \Sigma^* \}$, since D accepts $\langle M, w \rangle$ iff $M = A$. This set is a subset of A_{TM}, since every string in $\mathscr{L}(D)$ is a TM/string pair where the TM accepts the string. Moreover, this set is infinite, since for each $w \in \Sigma^*$ the string $\langle A, w \rangle \in \mathscr{L}(D)$. Finally, D is a decider because it always halts. ■

Why we asked this question: We want to make sure that everyone has a strong, nuanced understanding of the **R** and **RE** languages. All of the problems given here represent common misconceptions that students in past CS103 offerings have had. By forcing you to work through these problems, we hoped that you would refine your understanding of **R** and **RE** in a way that would help you in the tail end of this course.

Problem Six: Why Decidability and Recognizability? (24 Points)

i. Consider the language $REJECT_{HM} = \{ \langle H \rangle \mid H \text{ is an HM that rejects } \langle H \rangle \}$. Prove that $REJECT_{HM}$ is decidable.

Proof: Consider the following TM M:

> M = "On input $\langle H \rangle$, where H is an HM:
> Run U_{HM} on $\langle H, \langle H \rangle \rangle$
> If U_{HM} accepts $\langle H, \langle H \rangle \rangle$, reject.
> If U_{HM} rejects $\langle H, \langle H \rangle \rangle$, accept."

We claim that $\mathscr{L}(M) = REJECT_{HM}$ and that M is a decider, thereby proving that $REJECT_{HM}$ is decidable.

To see that M is a decider, consider what happens when we run M on any string $\langle H \rangle$. Since U_{HM} is a decider, U_{HM} halts on all inputs. Accordingly, it halts on $\langle H, \langle H \rangle \rangle$. Since M halts as soon as U_{HM} halts, this means that M halts on all inputs.

To see that $\mathscr{L}(M) = REJECT_{HM}$, note that M accepts $\langle H \rangle$ iff U_{HM} rejects $\langle H, \langle H \rangle \rangle$. U_{HM} rejects $\langle H, \langle H \rangle \rangle$ iff H rejects $\langle H \rangle$. Finally, H rejects $\langle H \rangle$ iff $\langle H \rangle \in REJECT_{HM}$. Thus M accepts $\langle H \rangle$ iff $\langle H \rangle \in REJECT_{HM}$, so $\mathscr{L}(M) = REJECT_{HM}$. ■

ii. Prove that there is no HM that decides $REJECT_{HM}$.

Proof: By contradiction; assume that there is a HM that decides $REJECT_{HM}$; call it B. Thus $\mathscr{L}(B) = REJECT_{HM}$. Notice that $REJECT_{BM} = \{ \langle H \rangle \mid H \text{ is an HM that rejects } \langle H \rangle \} = \{ \langle H \rangle \mid H \text{ is an HM and } \langle H \rangle \notin \mathscr{L}(H) \}$, since $\langle H \rangle \in \mathscr{L}(H)$ iff H accepts $\langle H \rangle$. We know that either $\langle B \rangle \in \mathscr{L}(B)$ or $\langle B \rangle \notin \mathscr{L}(B)$. We consider these cases separately:

Case 1: $\langle B \rangle \notin \mathscr{L}(B)$. Then $\langle B \rangle \in REJECT_{HM}$. But since $\langle B \rangle \notin \mathscr{L}(B)$ and $\mathscr{L}(B) = REJECT_{HM}$, this means that $\langle B \rangle \notin REJECT_{HM}$, contradicting our earlier assertion.

Case 2: $\langle B \rangle \in \mathscr{L}(B)$. Then $\langle B \rangle \notin REJECT_{HM}$. But since $\langle B \rangle \in \mathscr{L}(B)$ and $\mathscr{L}(B) = REJECT_{HM}$, this means that $\langle B \rangle \in REJECT_{HM}$, contradicting our earlier assertion.

In either case we reach a contradiction. Thus our earlier assumption must have been wrong, so there must not be a HM for $REJECT_{HM}$. ■

Why we asked this question: Part (i) of this question boils down to designing a decider and proving it has the appropriate language. We wanted you to think through how to do this and how to structure the proof of correctness. Part (ii) of this question is essentially a diagonalization argument that's very similar to the one we used to show that L_D is not **RE**. (In fact, $REJECT_{HM}$ is the diagonal language for HMs!), and we wanted to see if you were comfortable structuring a proof to this effect.