

Problem Set 4 Comments

Problem 1: Describing a Turing Machine. (Average: 16.8/20) Design a deterministic Turing Machine that decides the language $\{0^n 1^n 2^n \mid n \geq 0\}$. You should provide descriptions of your Turing Machine at the three levels of detail described in Section 3.3:

- a. Provide a *high-level description* of your machine.

Answer: The machine will go back and forth over the tape. Each iteration will start at the leftmost 0, replace it with a mark, then move right to find the leftmost 1 (if there is no such 1, reject), replace it with a mark, then move right to find the leftmost 2 (if there is no such 2, reject), replace it with a mark, and then return to the left edge of the tape. At the end, if there is no leftmost 0, then the tape is scanned and if there is any other symbol (not a mark or a blank) reject.

- b. Provide an *implementation description* of your machine.

Answer: Our machine has 5 main states: (1) (the start state) a state for finding the leftmost 0, (2) a state for finding the leftmost 1, (3) a state for finding the leftmost 2, (4) a state for finding the left edge of the tape. In state 1, if the current square contains a 1, write an X, move right, and enter state 2; if the current square contains an X, move right and stay in state 1; if the current square contains a blank, accept. In state 2, if the current square contains a 0 or X, move right and stay in state 2; if the current square contains a 1, replace it with an X, move right and enter state 3; otherwise, reject. In state 3, if the current square contains a 1 or and X, move right and stay in state 3; if the current square contains a 2, replace it with an X, move left, and enter state 4; otherwise, reject. In state 4, keep moving left until the left edge of the tape is found. This requires a few more states, but can be done by moving left, writing a temporary symbol, and knowing the left edge is reached when the temporary symbol is read; if not, replace the temporary symbol with the previous value in that square.

- c. Provide a full *formal description* of your machine (this is tedious, but everyone should do it once!).

Answer: The full formal description needs to show the states and all the transition rules. It can be built from the implementation description.

Problem 2: Deciding $0^n 1^{n^2}$. (Average: 15.8/20) Provide an *implementation description* of a Turing Machine that decides the language $0^n 1^{n^2}$ for $n \geq 0$.

Answer: First, scan the tape left to right checking that it is $0^* 1^*$. This can be done with a state 1 that on tape square 0 moves right and returns to state 1, and on input 1 moves right and enters state 2. In state 2, on input 1, move right and stay in state 2. On a blank, the input tape has the correct form. Enter a state that returns to the far left edge of the tape (as

described in the answer to the previous problem). After reaching the end, enter state B1. All other transitions from states 1 and 2 go to the reject state.

Now, we need to count the 0s and 1s. For each of the n 0s on the tape, there should be n 1s. We can do this by marking the 0s as we go through them, but we will need two marks - one to keep track of how many we have matched with n 1s, and one to keep track of the n 1s that match this 0. We will use these four tape symbols: X (a 0 that has been, or is being, matched with n 1s), $+$ (a 0 that has been matched in the current group of 1s), $*$ (for when we need both the X and $+$ marks on the same square, and $=$ (for marking the 1s that have been counted).

In state B1, move right until a 0 is found; if a 1 is found, reject; if a blank is found, accept; if an $=$ is found, go to state B5. If a 0 is found, replace it with an X , enter state B2. In state B2, find the left edge of the tape and enter state B3. In state B3, move right until a 0 or an X is found; if a 1 is found, reject; if an $=$ is found go to state B4. Replace it with either a $+$ (if it was a 0), or a $*$, if it was an X . Enter state B4, which moves right until a 1 is found (if no one is found before the ending blank, reject). Replace that 1 with an $=$. Move back to the left edge of the tape and enter state B3.

In state B4, we need to reset the 0 counters, and advance to the next 0. Moving left-to-right on the tape, replace $+$ with 0, and $*$ with X . Then, move back to the left edge of the tape and enter state B1.

In state B5, we need to check all the 1s have been matched. Move left-to-right on the tape (passing over the $=$ marks). If a 1 is found, reject. If a blank is found, accept.

Problem 3: Equivalence of 2-DPDA $+\epsilon$ and Turing Machine. (Average: 14.4/20) In Class 12, we informally argued that a 2-stack deterministic pushdown automaton with forced ϵ -transitions is equivalent to a Turing Machine. For this problem, prove one direction of the equivalence: that a 2-DPDA $+\epsilon$ can simulate any Turing Machine. (Hint: your proof should show how to construct a 2-DPDA $+\epsilon$ from a given Turing Machine. You will need to introduce a formal notation for describing a 2-DPDA $+\epsilon$.)

Answer: First, we need a notation for describing a 2-DPDA $+\epsilon$, which is similar to a regular DPDA but with two stacks. We assume both stacks use the same alphabet. So, we can describe it similarly to a regular DPDA, except the transition function has extra inputs and outputs. A 2-DPDA $+\epsilon$ is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where $Q, \Sigma, \Gamma, q_0,$ and F are defined as they are for a DPDA, and

$$\delta: Q \times \Gamma_\epsilon \times \Gamma_\epsilon \rightarrow Q \times \Gamma_\epsilon \times \Gamma_\epsilon$$

The two Γ_ϵ inputs correspond to the two stacks, and we have added ϵ to Σ to allow the forced ϵ transitions. Note that we removed Σ_ϵ from the δ inputs, since we will not use the input string at all in our 2-DPDA $+\epsilon$. Instead, we will start with the input on Stack 1.

Now, we need to show that we can simulate any Turing Machine with a 2-DPDA $+\epsilon$. Since a TM with a doubly infinite tape is equivalent to a TM (as proven in the next problem), we show that we can simulate a doubly infinite tape TM instead (this avoids the complications

caused by the left end of the tape). We can model the TM tape using the 2 stacks. Stack 1 will contain everything to the left of the TM head, so the top of the stack corresponds to the square one position to the left of the head. State 2 contains everything from the TM head to the right, so the top of the stack corresponds to the current square. Initially, Stack 1 is empty and Stack 2 contains the input w (pushed in reverse order, so the top of the stack will be the leftmost letter in w).

To simulate the TM, we need to define the corresponding δ rule for each possible TM rule.

If the TM δ_{TM} function includes a rule, $\delta_{TM}(q, b) \rightarrow (q_r, c, R)$ then, for the 2-DPDA+ ϵ ,

$$\delta(q, \epsilon, b) = (q_r, c, \epsilon)$$

That is, we pop a b from the right stack and push a c on the left stack.

If the TM δ_{TM} function includes a rule, $\delta_{TM}(q, b) \rightarrow (q_r, c, L)$ then, for the 2-DPDA+ ϵ ,

$$\begin{aligned}\delta(q, a, b) &= (q_{pushar}, \epsilon, c) \\ \delta(q_{pushar}, \epsilon, \epsilon) &= (q_r, \epsilon, a)\end{aligned}$$

Note that it is necessary to add an extra state to simulate this rule, since we need to push *two* symbols on the right stack when we move left. If we used a formal notation for describing a 2-DPDA+ ϵ that supported pushing multiple items, this would be simpler.

The only remaining issue to deal with is simulating the left edge of the tape correctly. The left stack represents the part of the tape to the left of the tape head. At the beginning of the simulation, we push a special symbol (not part of the original tape alphabet) on the left stack (we'll use ∇ as our special symbol). Then, we need a special rule to cover the case where the TM would try to move past the left end of the tape (recall that when a regular TM does this, it just stays in the leftmost square). To simulate this with the 2-DPDA+ ϵ , we need to add a rule corresponding to the $\delta_{TM}(q, b) \rightarrow (q_r, c, L)$ rule when the TM is at the left edge of the tape:

$$\delta(q_{pushar}, \nabla, \gamma) = (q_r, \nabla, a)$$

We need a corresponding rule for each q_{pushxy} state.

The set of states, Q of the 2-DPDA+ ϵ is Q_{TM} plus all the new states that are needed to simulate the right rules. At most, we need a new state for every $\Gamma \times Q_{TM}$ pair since they need to keep track of both the resulting state (q_r) and the symbol to push (a).

Problem 4: Robustness of Turing's Model. (Average: 14.6/20) (Sipser problem 3.11) A *Turing machine with doubly infinite tape* is similar to an ordinary Turing machine, but its tape is infinite to the left as well as to the right. The tape is initially filled with blanks except for the portion that contains the input. Computation is defined as usual except that the head never encounters an end to the tape as it moves leftward. Show that the class of languages recognized by a Turing machine with doubly infinite tape is equivalent to the class of languages recognized by a regular Turing machine.

Answer: To show equivalence, we need to show proofs in both directions.

A doubly-infinite TM can simulate a regular TM. This seems obvious (since the doubly-infinite tape adds functionality to the TM), but is a bit subtle because of the way a regular TM behaves when it would go off the left edge of the tape. We need to show we can simulate that same behavior with a doubly-infinite tape. One way to do this is to add a new symbol to the tape alphabet, #, to mark the edge of the tape. Add a step at the beginning of the TM that moves left one square, writes a #, and moves right back to the original starting square. Then, from each state, add a transition rule that says if you see a #, move right. This simulates the same behavior as the regular TM since if the machine ever tries to go to the # square, it moves back to the square representing the leftmost square on the regular TM.

A regular TM can simulate a doubly-infinite TM. One way to show this is to take advantage of the property established in the previous question: a 2-DPDA+ ϵ is equivalent to a TM. Thus, if we can simulate a TM with a doubly-infinite tape using a 2-DPDA+ ϵ , then we can simulate it with a regular TM, which we know is equivalent to a 2-DPDA+ ϵ . To prove this, we just need to modify our proof of equivalence to the 2-DPDA+ ϵ to change how we deal with the bottom of the stack when moving left. This is actually simpler than the original proof: we just eliminate pushing the initial ∇ on the stack and the rules involving ∇ .

Problem 5: Deciders and Recognizers. (Average: 21.1/20+) In Class 12, we argued that DFAs, DPDAs, and NFAs always terminate, hence the sets of languages that can be recognized by these machines are equivalent to the sets of languages that can be decided by them. By contrast, a Turing Machine may run forever on some inputs, so there may be languages which can be recognized but not decided by a Turing Machine (we will see examples of such languages next week). What about nondeterministic pushdown automata?

- a. Draw a Venn diagram including these sets:

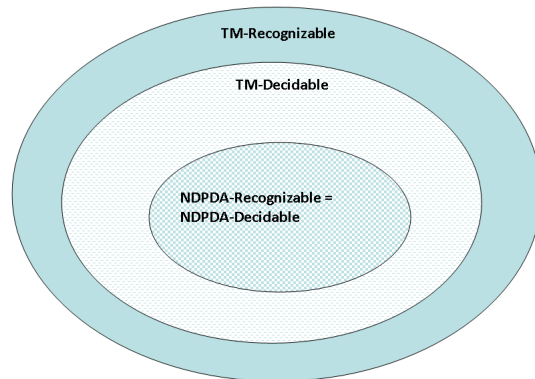
NDPDA-Recognizable: languages that can be recognized by a NDPDA.

NDPDA-Decidable: languages that can be decided by a NDPDA.

TM-Decidable: languages that can be decided by a Turing Machine.

TM-Recognizable: languages that can be recognized by a Turing Machine.

Answer:



The set of languages that can be decided by an NDPDA is the same as the set of languages that can be recognized by a NDPDA, namely the context-free languages.

- b. (bonus) Prove that your diagram correctly depicts the relationship between *NDPDA-Recognizable* and *NDPDA-Decidable*. (For example, if your diagram showed them as the same circle, prove that the sets are equivalent. If your diagram shows one set inside the other, prove that all languages in one set are in the other set but that there is some language in the outer set that is outside the inner set.)

Answer: To show they are the same we need to argue for every language described by a CFG, there is some way of describing the same language for which the language acceptance is guaranteed to terminate. One way to do this is to use the fact that every CFL can be generated by a context-free grammar in Chomsky normal form (Sipser's Theorem 2.9). In this form, it is clear to see that for a given-length string, there are a limited number of steps that could be used to generate the string. Hence, the process of generating the string will always terminate. This shows that any language that can be recognized by an NDPDA can also be decided by an NDPDA, since there is an NDPDA equivalent to any CFG.

- c. (bonus) Prove that your diagram correctly depicts the relationship between *NDPDA-Recognizable* and *TM-Decidable*.

Answer: This is an example of a question that should convince everyone to try the bonus questions — it is actually the easiest question on this problem set since it is just asking if a TM can decide a language that is not a context-free language. Your answer to Problem 1 demonstrates that there are some languages in *TM-Decidable* that are not in *NDPDA-Recognizable*.