Lectures and Reading. After today’s lecture on regular expressions and their equivalent NFAs (with \(\lambda\)-transitions), the next three lectures will complete the cycle of showing those two formalisms recognize the same class of languages as DFAs. The class is standardly denoted by \(\text{REG}\) for the class of \textit{regular} languages. Then lectures will move into a technique for showing that certain other languages are \textit{non}-regular. Most often this is done via something called the “Pumping Lemma,” but for philosophical as well as personal reasons I prefer the \textit{Myhill-Nerode Theorem}, which is named for John Myhill of UB (who passed away in 1987 before I joined here) and Anil Nerode (whom I had a postdoc under at Cornell and who is still alive and well). The reading for this is the Rochester slides linked from the course webpage, plus the Myhill-Nerode handout given out in hardcopy.

In the following week, lectures will move back to computability and undecidability, and we will rejoin Chapter 1 of the text by Arora and Barak. Whereas several older texts treat computability long before even mentioning time and space complexity, Arora-Barak introduces all the most important concepts from the get-go, and my lectures will do likewise. On undecidability they will have a little more detail, as exemplified by the handout on diagonalization also given out in hardcopy.

(1) Suppose we change the rules of the “spears and dragons game” to read as follows:

- Each character in the input string—figuratively, each char is a “room” in a linear “dungeon”—is either \(s\) for “spear,” \(d\) for “dragon,” or \(\ell\) for “lamp.”
- The Player \(P\) may hold a maximum of 2 spears at any one time.
- Upon entering a room with a spear, \(P\) may pick it up unless already carrying two spears.
- Upon entering a room with a dragon, if \(P\) has no spear, \(P\) is dead. Else, \(P\) uses one spear to kill the dragon, and is then carrying one fewer spear.
- Upon entering a room with a lamp, if \(P\) has killed two dragons since the last time \(P\) picked up a spear (it follows that \(P\) currently has no spear), then \(P\) may rub the lamp and the genie in the lamp will give \(P\) one spear. Otherwise, the lamp has no effect.

Design a deterministic finite automaton \(M\) that simulates this game. In particular, \(L(M)\) should equal the language of strings representing “dungeons” that \(P\) survives. The final state of \(M\) should also tell the number of spears that \(P\) has when exiting the dungeon. (If you sense an ambiguity in the rules, you may ask on \textit{Piazza}, and/or you may describe the perceived ambiguity and detail the interpretation you took to resolve it. 18 pts.)

(2) Let \(\Sigma = \{0, 1, \#\}\) where the \(\#\) is a “loud comma” used to form pairs. Unlike the remark on p8 of the Arora-Barak notes (section 0.1) where things are then re-converted to binary notation, you may keep \# as a literal character. Define \(A\) to be the language of pairs \(u\#v\) where \(u\) and \(v\) are binary integers written with their \textit{least} significant bits first, such that \(u + v\) is 1 less than a power of 2. For instance, 010011\#1011 belongs to \(A\) because the numbers are 50 and 13 which sum to 63, but 111111\#1111 is not in \(A\) because the sum is even, for one thing. Leading zeroes are OK, so 01001100\#10110 also belongs to \(A\).
(a) Design a two-tape deterministic Turing $M$ such that $L(M) = A$. Your $M$ should naturally run in $O(n)$ time where $n$ is the number of input chars (including the #—so $n = 11$ in the first two example strings above, $n = 14$ in the last). Does it run in real time—meaning that its input-tape head reads a char and moves right (‘R’) at each step and the machine halts immediately when it reaches the blank after the last char?

(b) Now suppose the first number $u$ is written with its most significant bit first, but the $v$ part is still written least-first. Make a revised version $M'$ of your $M$ to handle this case, and also answer: Is $M'$ a deterministic pushdown automaton? (That is: does its input tape head never move left, and does its second-tape head always write a blank if and when it moves left?) Does $M'$ run in real time? What if we always use leading 0s to pad the $u$ and $v$ parts to the same length when we give $u\#v$ as input?

(c) Finally, suppose we have no # and define the language $A'$ this way: $A' = \{x \in \{0, 1\}^*: x$ can be broken as $x =: uv$ such that $u + v$ is 1 less than a power of 2, where $u$ is written with its most significant bit first and $v$ is written with its least bit first$\}$. For instance, $x' = 0100111011$ does not belong to $A$, because the breakdown $u = 010011$, $v = 1011$ would work only if we were writing $u$ lsb-first, and because no other breakdown works either. But $x'' = 1100101011$ works because we are now writing only the $v$ part reversed in lsb-first order, and $110010 + 1101 = 111111 = 63$ as before.

Revise your $M'$ from part (b) to be a nondeterministic TM $N$ such that $L(N) = A'$. Is it (still) a pushdown automaton? Does it (still) run in real time (maybe under assumptions)? Without actually carrying this out in Turing machine code, think how you would design a deterministic TM $M''$ to recognize $A'$...would it still be a PDA and/or run in real time?

It is neither required nor expected to use the Turing Kit software to design these machines. Well-commented hand-drawn diagrams are sufficient. It is also OK for your answers to (b) and (c) to be partial sketches just indicating changes to parts(s) of the answer to (a), provided they are clear enough. Each of the three questions in (b) and the three in (c) needs a brief explanation as well as yes/no answers, for 3 pts., each. The total points are $18 + (6 + 3x3) + (6 + 3x3)$ giving 48 on the problem and 66 points on the set.