CSE596, Fall 2022 **Problem Set 2** **Due Thu. Oct. 6**

Lectures and Reading. This week moved on to Turing machines. The attention in early lectures was on "gritty" character-level detail, but the purpose by next week will be to move away from it. The essence is conveyed in the diagram https://cse.buffalo.edu/ regan/cse396/UTMRAMsimulator.pdf of how a TM can simulate a random-access machine (RAM), really a miniature but serviceable assembly language. Since the assembly language could be a "virtual" compilation target for any (known) high-level lnguage (HLL), this establishes the equivalence of TMs and HLLs, which is major evidence for the *Church-Turing thesis* (CTT).

In fact, t steps of the RAM can be simulated by $O(t^4)$ steps of the TM. Steven Cook—in a less-famous theorem he proved in 1970-71—improved this to $\tilde{O}(t^3)$ by a caching strategy roughly similar to how C++ `vector` amortizes space by doubling the size when needed. Under "fair-cost" RAM time, which counts one time unit per bit of the operands in a step rather than "unit cost" which just counts steps—Cook showed $\tilde{O}(t^2)$. The target TM in all cases has 3 or 4 tapes, and there is a further $O(t^2)$ overhead for getting that down to one tape (as will be sketched in class, Theorem 7.6 on page 22 of the notes). So going from time as reckoned in an algorithms course to a single-tape TM could multiply the exponent by 8, but that's OK in this sense: a polynomial running time $n^{O(1)}$ remains $n^{O(1)}$. Thus TMs and HLLs are equivalent up to polynomial time. This led to a *polynomial-time CTT*, which is challenged by quantum computers.

For our purposes now, CTT says that the notions of *decidability* and *undecidability* defined formally via Turing machines are *general* and *robust* concepts. Next week will define computational decision problems (per section 8 of Debray's notes) and discuss decidable and undecidable problems. Instead of making A_{TM} or its complement the first undecidable problem, lecture will introduce the "Diagonal Language"

$$D_{TM} = \{x : x \text{ is the code of a TM that does not accept } x\}.$$

The idea of this language is handled in the proof of Theorem 8.5 by making "D" a hypothetical machine, but it is much better IMHO to minimize hypothetical quantities and focus instead on the concrete definition of a language. By showing that D_{TM} is not even computably enumerable (that is, not the language accepted—let alone decided—by any TM), we will show that A_{TM} is undecidable. Then we will use *mapping reductions* to show more undecidable problems—so for a week from now, please start section 9 in the new set of notes. *Please read ahead to the course notes that cover this progression of undecidable languages.*

(1) Prove that two of the following three languages are non-regular, via a Myhill-Nerode argument. For the regular one, give a regular expression. Here $\#a(x)$ denotes the number of occurrences of the character a in the string x, and more generally, $\#w(x)$ denotes the number of occurrences of the substring w in x. For example, $\#0010(00100100) = 2$ even though the two occurrences of the substring 0010 overlap each other. Also, for two strings x, y of the same length, $x \oplus y$ denotes the bitwise exclusive-OR, e.g. $1011 \oplus 0010 = 1001$. All three languages are over the alphabet $\Sigma = \{0, 1\}$.

(i) $L_1 = \{x : \#0(x) \leq \#1(x)\}$.

(ii) $L_2 = \{x : \#01(x) = \#10(x)\}$.

(iii) $L_3 = \{xy : |x| = |y| \wedge x \oplus y = 1^{|x|}\}$.

$(3 \times 12 = 36$ pts.)

(2) Now consider $L_4 = \{x : \#010(x) = 0 \wedge \#101(x) = 0\}$. Use the Myhill-Nerode technique to show that any DFA M such that $L(M) = L_4$ requires at least 6 states. Then design such a DFA M—ideally showing how your proof guided you to it (or vice-versa). Finally explain why you can basically "collapse" M into a generalized NFA with only 2 states s, f such that

$$L(M) = L_{s,s} \cup L_{s,f} \cup L_{f,s} \cup L_{f,f},$$

and use that to give a regular expression for L_4. $(12 + 6 + 9 = 27$ pts.)

(3) Design a two-tape deterministic Turing machine M_2 that recognizes the language

$$L_3 = \{x\#0^k\#y : x, y \in \{0,1\}^*, x = y \wedge |x| = k\}.$$

Here $\Sigma = \{0, 1, \#\}$ but the # character is only allowed as a marker to divide the input string w into thirds. Your M_2 should run in $O(n)$ time where $n = |w|$; note that any accepted string gives $n = 3k + 2$ with k as above. A well-commented arc-node drawing is fine; if you use the Turing Kit, please take a screenshot since its own Postscript-based print feature is old and may be wonky.)

Then argue as best you can that every single-tape TM M_1 such that $L(M_1) = L_3$ requires $\Omega(n^2)$ time. Since n is linear in k, it may help to think of this as $\Omega(k^2)$ time. Start by showing that $S = \{0,1\}^k$ is PD fopr L_3. Then argue that this means k bots of information must somehow cross the middle 0^k part in order to decide $y = x$ correctly. Finally reckon how much total time M_1 must spend in that middle region, noting that M has a fixed number $r = |Q|$ of states but k can grow. $(18 + 18 = 36$ pts., for 99 on the set)