Computing Theory

Assignment 2

Name: Duncan Do

Student Number: s3718718

Exercise 1: Turing Machine Design

a. E1-Qa1.jff
b. E1-Qb1.jff
c. E1-Qc1.jff

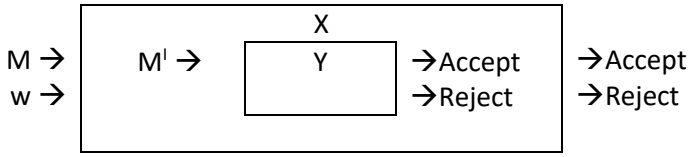
Exercise 2: Undecidability

a. HALT TWO: Determine whether an arbitrary machine (M) will halt on an empty tape/input leaving exactly two words on the tape, Blank delimited. Accepts when M halts on lambda and writes two strings blank delimited. Rejects when the tape is not two words blank delimited or when it does not halt.

   A_{TM}: Determine whether an arbitrary machine (M) will accept on an arbitrary input (w). A_{TM} will return accept/yes if M accepts w, and will return reject/no if M does not accept w OR M never halts on w

   We will reduce A_{TM} to HALT TWO to prove HALT TWO is as hard as A_{TM}, an undecidable problem.

   Assume Y exists that solves HALT TWO. We will construct machine X that solves A_{TM} using Y.



   X takes M and w and constructs M^{I}. M^{I} is a machine that will disregard any input and write w on the tape and runs M on it. M^{I} will accept if M accepts w and will reject if M does not accept. M^{I} will have all of M's accepting states redirected to a process that will leave two words on the tape, blank delimited. When M^{I} does not accept, the tape will be cleared before halting. X will pass M^{I} to Y and accepts if Y accepts.

   We will show that X accepts when run on M and w iff M accepts when run on w.

$$X(M, w) = \text{accepts} \rightarrow M(w) = \text{accepts}$$

   - Assume M accept when run on w, then M^{I} will always write two words on the tape, blank delimited. Because M^{I} will always write that on the tape, when passed into Y, it will always accept when run on M^{I}. If Y accepts, then X accepts, when run on M and w
   - Assume X accepts when run on M and w. Then, Y accepts M^{I}, which means M^{I} will always write two words on the tape, blank delimited; because M^{I}'s functionality disregards any input. Therefore, M will always accept w because M^{I} simulates M on w

   Therefore, we can reduce X, a representation of A_{TM}, to Y, and as such, machine Y is at least as complex as A_{TM}. Since A_{TM} is undecidable, Y cannot exist and HALT TWO is undecidable.

   Hence, Y determines if a machine (M^{I}) halts on an empty input and writes two blank delimited words, meaning Y determines if a machine is an element of L_1. Since we can reduce X, a representation of A_{TM}, to Y, and as such, machine Y is at least as complex as A_{TM}. Since A_{TM} is undecidable, Y cannot exist and HALT TWO is undecidable.

   Therefore, determining if a machine is an element of L_1 is undecidable. ***QED***.

b. Is L_1 recursively enumerable?

Yes, L_1 is said to be a recursively enumerable language because there exists a Turing machine that will enumerate all valid strings of the language. This is because if an arbitrary Turing machine for L_1 would halt, on an input and produce the desired outcome, it would simply do so. To be recursively enumerable, if L_1 was given an input not in the language, it might halt and reject, or run forever. This also holds true for L_1, as if a machine for L_1 did halt on a given input and didn't produce the 2 words required, it would simply reject the input as it does not exist within the language. In addition to this, the machine $M(L_1)$ is designed to only accept if it halts; thus it should produce a definitive no/reject if it never halts. However it is impossible to determine if something will never halt; evident by famous example of the halting problem, as from the question above, can be reduced to HALT TWO, a machine that "represents" L_1. Because the halting problem can be reduced to L_1 and the halting problem is also recursively enumerable, with the above justification; it can be said that L_1 is recursively enumerable

Is L_1 recursively?

No, L_1 cannot be recursive because the criteria to be a recursive language states that the language in question and the compliment of language are recursively enumerable (able to give a yes to an input, but not always being able to give a no). L_1's compliment (L_2) is the language of all machines that will not halt on an empty tape. No other aspect of the language needs be analysed beyond that statement; because you cannot determine if a machine will never halt; it might just take an egregious amount of time to halt. This again, can relate to the halting problem. The compliment of the halting problem is deciding whether a machine will not halt on a given input, a non-recursively enumerable language; a reduction from (Comp)HALT to L_2 can be made to show that L_2 is not recursively enumerable. Thus, since L_2, L_1's compliment is not recursively enumerable, L_1 cannot be recursive

Is L_1 non recursive enumerable?

No. As discussed above, L_1 is a recursively enumerable language. In the simplest of terms, there exists a machine such that you can achieve yes to whether an input exists within the language but a no is not guaranteed, since the machine might say no if the input does not exist within the language or it might run forever. In a more formal terms; there exists $M(L_1)$ that can enumerate over all the valid machines, w, iff $w \in L_1$.

Is L_1 Uncomputable?

No, Uncomputable is just another word for undecidable, and HALT TWO (The problem of L_1) is can be reduced from the Halting Problem. This means that HALT TWO is at least as difficult as the Halting problem, if a problem is at least as hard as an undecidable problem, it is also undecidable and therefore Uncomputable.

Exercise 3: Complexity

a.

$$g(n) \in O\big(h\big(g(n)\big) + c\big)$$

$g(n) \in O\big(h\big(g(n)\big) + c\big)$ iff $\exists\ n_0, d$, where $d \in \mathbb{R}^+$, $\forall n \in \mathbb{N}, n \geq n_0$

$$d \times \big(h\big(g(n)\big) + c\big) \geq g(n)$$

Let $n = g(n)$
Since $h(n) \geq n$
Then $h(g(n)) \geq g(n)$

$$h\big(g(n)\big) + c \geq h\big(g(n)\big), c \geq 0$$

Since $d \in \mathbb{R}^+$,
Then $d\ \times h\big(g(n)\big) + c \geq h\big(g(n)\big) \geq g(n)$
$\therefore d \times h\big(g(n)\big) + c \geq g(n)$

Let $d = 1$, Since $1\ \times h\big(g(n)\big) \geq g(n)$ for all n, there exists an n_0 and d such that $\forall n \in \mathbb{N}, n \geq n_0$

$\therefore$ If a Turing Machine runs in the time complexity of $O(g(n))$, then it can run in the time complexity of $O(h\big(g(n)\big) + c)$.

QED.

b.

The diagram is a visual representation if the different time complexities decision problems can fit into, structured in a hierarchal type format. Those higher in the hierarchy take more effort (time/space) to solve.

Log Time: The first and smallest section at the bottom of the diagram (LOG Time) represents all the decision problems can be solved with a Turing machine using a logarithmic amount of time.

Log Space: The section right above it (LOG Space), represents all the decision problems can be solved with a Turing machine using a logarithmic amount of space.

P Time: P Time represents all the decision problems can be solved with a deterministic Turing machine using a polynomial amount of computing time or polynomial time.

NP Time: P Time represents all the decision problems can be solved with a non-deterministic Turing machine using a polynomial amount of computing time or polynomial time.

NPC: NPC means NP Complete, a special class of decision problems where the problems exist in the NP class (Can be solved in non-deterministic polynomial time and checked in deterministic polynomial time) and are as hard as any other problem in the NP class, NP Hard.

Co-NP Time: A decision problem, C, exist within the Co-NP class if there exists an algorithm of polynomial time that can transform any other problem in the complexity class to C. The resulting consequence of existing in this class means that with the polynomial algorithm for problem C could be used to solve all other Co-NP problems in polynomial time.

P Space: P Space represents all the decision problems can be solved with a deterministic Turing machine using a polynomial amount of space.

EXP Time: EXP Time represents all the decision problems that have an exponential run time, meaning they are an element of big O class $O(2^{p(n)})$, where $p(n)$ is a polynomial function of n.

EXP Space: EXP space represents all the decision problems that can be solved by a deterministic turning machine exponential space, $O(2^{p(n)})$, where $p(n)$ is a polynomial function of n.

2 EXP Time: EXP Time represents all the decision problems that have an exponential run time, meaning they are an element of big O class $O(2\text{^}(2^{p(n)})$, where $p(n)$ is a polynomial function of n.

Elementary: The Elementary time complexity class is a special class which is defined by the union $n \times$ EXP classes where $n \geq 1$. Meaning ELEMENTARY = EXP ∪ 2EXP ∪ 23EXP… The class specifically relates NEXP, the class represents decision problems that can be solved in non-deterministic polynomial time using a Turing machine. This is because NEXP's time complexity upper bound is 2^ n^k. Meaning decision problems in the Elementary class can be solved in 2^poly(n) time, 2^2^poly(n) time, 2^2^2^poly(n) time, etc.

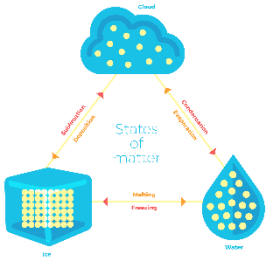
R: R is the upper-class which is the limit of our human knowledge, It's the class of decision problems that can be solved with a Turing machine (Which is equivalent to the set of all recursive languages), the most powerful machine we know of and which we use to define our basis of what is decidable. Hence why it is at the top of the diagram, as if it cannot be decided by a Turing machine, we deem it undecidable (The thick black line represents the end of our ability to solve decidability).

c. This problem is not NP-Complete, rather its NP-Hard. Meaning it is as hard as any other problem in NP. However, this problem is not an NP problem, as it cannot be checked in polynomial time with a deterministic turning machine. If you were given a set of subsets which was said to be the most optimised set of subsets to cover set 'U'.

   The only way to check that is to, first check that subset does cover set 'U'; and second, run that set of subsets against every other set of subsets that use less subsets; this cannot be done deterministically in polynomial time.

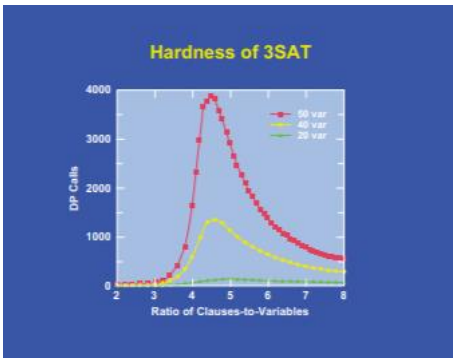
   It either must use a machine with non-determinism or at a time complexity of $O(n!)$ where n is the number of subsets to choose from, a time complexity that exceeds NPTIME.
   The time complexity of solving SC and checking SC is the same, $O(n!)$.

d. The phase transition phenomenon itself is where there a sudden sharp transition from one state to another. We can depict this in a general circumstance. Take the states of matter when it comes to H_2O.



Although all these states have the same fundamental make up, their forms, or states, are drastically different.

This in terms of Random 3-SAT, or the satisfiability problem in general, is the sudden change/transition in difficulty of the problem based on how many clauses exist versus how many variables exist.



This diagram illustrates how the ratio of clauses/variables can have a drastic impact on the difficulty of 3SAT. This diagram does convey the idea that the state transition phenomenon leaves many 3SAT problems "very difficult, however that is not always a detriment to 3SAT. Using the theory of the state transition phenomenon, we can conclude that r exists where $r \in \mathbb{R}^+$, such that a 3SAT problem with a clause/variable ratio higher than r is so hard that it is unsatisfiable. Subsequently, 3SAT problems with a clause/variable ratio lower than r are satisfiable.

Researchers throughout the ages have attempted to find an upper and lower bound for r

Notable concluded results are:

Upper bound of R

- r = 5.1909 (1983) Franco, Paull (and others)
- r = 5.19 – 10^{-7} (1992) Frieze and Suen
- r = 4.667 (1996) Kirousis, Kranakis, Krizanc
- r = 4.642 (1996) Dubois, Boufkhad
- r = 4.596 (1999) Janson, Stamatiou, Vamvakari
- r = 4.506 (1999) Dubois, Boukhand, Mandler
- r = 4.49 (2008) Diaz, Kirousis, Mitsche, Perez
- r = 4.453 (2008) Maneva, Sinclair

Lower Bound

- r = 2.66 (1986) Chao, Franco
- r = 2.99 (1986) Chao, Franco
- r = 3.145 (2000) Achlioptas
- r = 3.42 (2002) Kaporis, Kirousis, Lalas
- r = 3.52 Kaporis, Kirousis, Lalas (2003)
- r = 3.52 Hajiaghayi, Sorkin (2003)

Exercise 4: Pumping Lemma & Regularity

a. 9/10
   You say the word must hold for **every** value of n where $n \geq 1$, this is not true for the law of Pumping Lemma. If this was true, then this limits the language (e.g. if it must hold for $n = 1$, it limits the language to a sequence of a single character, such as “aaaaaaa”, “bbbbbbb”). n represents the number of states in the DSM, by saying every, there must exist a DSM with 1 state for every language which is not the case.
b. MEACFHJLN
c. [3] $a^{(k+1)^2+k-5}$
   [5] $|xy| \leq n$
   [6] y must follow the form a^m where $1 \leq m \leq n < k$
   [7] xyyx, $|w^{I}|$ = xyyz = |xyz| + |y|= $a^{(k+1)^2+k-5} + m$
   [9] let k = 3, then w = $a^{(3+1)^2+3-5}$, w = 14, then let k = 4, the w = $a^{(4+1)^2+4-5}$, w = 24. However let m = 2 when k = 3. when constructing |xyyz|, w^{I} = $a^{(3+1)^2+3-5} + 2$, w^{I} = 16. which does not follow the language rules set by L.
   [11] Chomsky’s Hierarchy
d. L2 is not regular because it is the compliment of L1 (stated in E4-Qb1), a non-regular language. By flipping all states of a DFA, you can easily find the compliment DFA and subsequent regular language. Thus, if a language is regular, so is its compliment. Since L2's compliment language, L1, is not regular, L2 cannot be regular.