Problem 1.

Are the following languages Turing decidable, Turing acceptable but not Turing-decidable, or not even Turing acceptable?

Solution

Part (a)

$$L = \{p(M)p(w) : M \text{ uses a finite number of tape cells when running on input } w\}.$$

acceptable Assume we are given some machine M and a string w for which our condition holds, that is M uses a finite number of tape cells when running on input w. We want L to halt and accept this input configuration. Note that it is not necessary for M to halt, it is possible that M loops over a finite subset of the tape, thus never halting, but never using infinite tape cells. If this is the case (finite memory), note that Q, Σ are both finite, the tape head can only be on a finite number of positions, and the tape stores a finite amount of data. Given this, we can produce some sort of finite 2-tuple (machine state and the finite state of the tape) which encodes every possible state this machine can be in (I reiterate we are using the strong assumption that M uses finite resources). Since we know that there are finite such possible 2-tuples (there is a finite number of unique configurations our machine can be in), and that the machine never halts, we can conclude that the machine will enter the exact same configuration at least twice (infinitely in particular) for at least one configuration of the machine, this follows from the Pigeon-Hole Principle (infinitely iterations cannot fit in a finite number of states). Further if a machine enters a given state, call it S and later enters that same exact state S, we can guarantee that the machine will infinitely enter state S as it runs. This is because our current configuration is exactly the same and one run following S led to another S, all subsequent occurrences of S will lead to another.

From all this we conclude if the machine does not halt but uses finite resources, it MUST repeat some configuration of the machine (including state and tape layout). Since S is repeated infinitely often, we know that it is impossible that we are writing to more cells in the tape then are used in S, else the next time we reach S we would have a different tape configuration contradicting $S = S$ (that is repeated configurations $\Rightarrow$ finite memory).

Given input M and w our machine will halt and accept if M halts. If M halts it must have done so in finite steps $\Rightarrow$ it used finite resources $\Rightarrow$ we should accept it. For each step of M we will encode it's current configuration (as an 2-tuple) at some empty point on our tape.

Then we will check all configurations this machine has been in (which will always be finite, but arbitrarily large) and if any configuration is repeated we will also halt and accept. This is because as argued above M has entered an infinite loop via repeated configurations $\Rightarrow$ M is using at most the number of cells used in that repeated configuration $\Rightarrow$ M is using a finite number of cells. In both cases (M halts and M does not halt) L will halt and accept.

If M uses an infinite number of tape cells, L will not halt (thus will not accept), this is argued below for decidability but mentioned to verify that L does not accept these configurations. Therefore L is Turing-acceptable.

decidable Assume we are given some machine M and a string w for which M uses an infinite number of cells. As M can only modify one tape cell in one step, use of infinite tape cells requires infinite steps $\Rightarrow$ M does not halt. Further, we can make now assumptions about the behavior of M with regards to not halting (above we could make assumptions about space usage). Thus we would need to be able to solve the halting problem in order to conclude to halt and reject M. But as the halting problem is undecidable, L must also be **not** decidable.

Part (b)

$$L = \{p(M)p(w)01^n0 : M \text{ uses at most } n \text{ tape cells when running on input} w\}.$$

acceptable Here we have a similar setup to part (a) with the following modifications: after each iteration of M, check the simulated tape for M, if the number of used tape cells exceeds n (encoded as a string of 1's), then halt and reject. If M halts, then it never exceeded n tape cells (else we would have already halted and rejected), thus we can halt and accept. If M repeats a configuration (and enters a loop as noted above), then we know that the number of currently used tape cells is the maximum it will ever use (argued in part (a)), therefore we can halt and accept (again if it was using more than n we would have already rejected it).

Assume we are given M, w and n encode into L and that M computes w using at most n tape cells. Then, if M halts our machine must also halt, and since it never used more than n tape cells, we never rejected it, therefore we have accepted it. If M does not halt, n is finite $\Rightarrow$ M uses a finite amount of memory $\Rightarrow$ M will enter a repeat configuration. That configuration will not use more than n tape cells, therefore we will have never rejected the input and will instead halt and accept. In both cases, L halts and accepts, therefore L is Turing-acceptable.

decidable

case M halts:

Since L simply simulates M with a couple of extra steps in between each iteration, if M halts then so must L this is fairly trivial. Further if M has used more than n tape cells, we must have rejected it whenever it past that barrier. L halts and rejects in this case.

case M does not halt, uses finite tape cells:

M must have entered a repeat configuration and we must have halted at latest at that point. If M uses more than n tape cells then we rejected as soon as it did so. Thus L both halts and rejects in this case.

case M does not halt, uses infinite tape cells:

M uses infinite resources therefore it must use exactly $n+1$ resources at some point. Further it reaches $n+1$ resources in finite time, if it reached it in infinite time that would contradict the idea that M uses infinite tape cells. L is designed to halt and reject as soon as $n+1$ tape cells are used, and as these are used in finite time. L halts and rejects in finite time in this case.

In all cases, L halts and rejects. Therefore L is Turing-decidable.

Problem 2.

Are the following languages Turing-decidable? Turing-acceptable but not Turing-decidable? Not even Turing-acceptable? For each answer, just give an intuitive explanation of your reasoning, no formal proof is required (just as in class, M is a generic deterministic Turing machine, w is a generic input string to it, and p is an encoding function).

Solution

In all the problems below, I discuss using a counter to keep track of iterations run by M, this counter is stored at some arbitrary point on the tape. Not that important, but maybe worth mentioning.

Part (a)

$$L = \{p(M) : |L(M)| \leq 10\}$$

acceptable Assume we are given a machine for a language such that $|L(M)| \leq 10$. We want to verify this condition given only the machine M. To do so would requiring enumerating all possible strings and checking whether or not M accepts them, if so we would increment a counter. After enumerating all strings we would check that counter ≤ 10 and if so we would accept. Note that this requires checking **all** possible strings and this is the problem. Strings can be arbitrarily large to infinite (though not infinite) and thus we cannot possible check all strings. There is also a potential problem of M not halting for a given input, though this we can overcome by simulating multiple Turing machines, and adding a new set of Turing machines to simulate adding the next character to the current string (as a string which is substring to another string will behave equivalently until the difference is reached). Therefore L is not Turing-acceptable and cannot possibly then be Turing-decidable (note it cannot be decidable without first being acceptable).

Part (b)

$$L = \{p(M) : |L(M)| \geq 10\}$$

acceptable Assume we are given a machine for a language such that $|L(M)| \geq 10$. We have a finite alphabet we are working with that is encoded into $p(M)$. We can enumerate every possible string for sizes counting from $0, 1, 2, ...$ and so on keeping a counter on when we find a string accepted by M. Once this counter reaches 11, we halt and accept. We must do this search in parallel, in a sort of breadth first search branching, this is because M may not halt for certain inputs. If we do this search one iteration at a time branching out and

expanding to simulate adding characters to the string, then we can guarantee that we will terminate when we find 11 strings, even if some early input is still running.

That is, because we know M has at least 11 such strings, and these strings have finite length, we must therefore be guaranteed to halt in finite time. Given a machine we would want to reject, our machine will never halt (as argued below under decidability), thus this machine accepts the definition. Therefore L is Turing-acceptable.

decidable We employ a similar approach here, this time given a machine such that $|L(M)|< 10$. Our machine must verify this condition given only M. However doing this would require permuting all possible input strings and making sure less than 10 are produced by M, if so our machine would halt and reject. However input strings can be arbitrarily large to infinite, thus our machine will loop forever checking every possible string. Therefore L is **not** Turing-decidable.

Part (c)

$$L = \{p(M)p(w) : M \searrow w \text{ in 10 steps or less } \}$$

acceptable Assume we are given a machine M and string w and assume that the condition "$M \searrow w$ in 10 steps or less" is true. This is easy to verify, simply run M on w keeping a counter on each step, once our counter reaches 11 we halt and reject. If M halts before that our machine will halt and accept. Because it is given that M will halt in 10 steps or less on w, we know that our machine L will halt as well (and then accept). If the machine halts in more than 10 steps, our machine will clearly reject it after it reaches 11. Therefore L is Turing-acceptable.

decidable Assume we are given a machine M and string w for which M will not halt in 10 steps or less, this could mean that it halts in finite time or runs indefinitely. L should reject this input. Again, following the outline given above, we will halt and reject as soon as our counter reaches 11, thus even if M does not halt on w we don't care as L won't need to run (simulate) past step 11 (of M). Therefore L is Turing-decidable.

Part (d)

$$L = \{p(M)p(w) : M \searrow w \text{ in 10 steps or more } \}$$

acceptable Assume we are given a machine M and a string w for which M will halt in 10 steps or more. This will be some finite positive integer, call it $n \geq 10$. Our machine L will run M on w and keep a counter on the number of iterations if M halts and the counter is

less than 10 we halt and reject otherwise we halt and accept. We know that M will halt and specifically will halt with our counter set to n. Further we know that $n \geq 10 \Rightarrow$ counter ≥ 10, thus our machine is guaranteed to halt and accept the input of M and w. In the counter case, if M halts in less than 10 steps, the counter will be less than 10 and we will thus appropriately reject. If M does not halt, as argued in decidability, L will also not halt. Therefore L is Turing-acceptable.

decidable If our machine M halts on w and does so in less than 10 steps we will obviously not have troubles. However the case of M not halting on w is trouble. In this case in fact we just have the halting problem. We cannot decide whether or not M will halt in finite time using L. Therefore L is **not** Turing-decidable.

Problem 3.

Use reduction to prove that the language

$$L = \{p(M_1)p(M_2) : L(M_1) \subseteq L(M_2)\}$$

is not decidable (M_1 and M_2 are Turing machines, of course).

Solution

Proof. □