COT 4210 Final Exam Part C: TMs, Decidable/Undecidable Languages 5/4/2021 Solution

1) (8 pts) Let L_1 and L_2 be Turing Recognizable languages. Prove that $L_1 \cup L_2$ is also Turing Recognizable. In order to earn full credit, your proof must be detail oriented.

Solution

Since Let L_1 and L_2 be Turing Recognizable languages, there must exist Turing Machines that recognize these languages. Let these machines be TM_1 and TM_2, respectively. We will use these two machines to design a new Turing Machine, TM_3 that recognizes $L_1 \cup L_2$, to prove that $L_1 \cup L_2$ is Turing Recognizable.

Create a multitape Turing Machine that copies the initial input onto two separate work tapes. Then, the machine should use the instructions of TM_1 to simulate one step on the first work tape, followed by simulating one step from TM_2 on the second work tape. Continue alternating in this fashion. If either simulation accepts, TM_3 should accept and halt. If both simulations reject, then reject and halt. If neither simulation accepts and at least one doesn't reject, then this machine may loop.

This new machine is a recognizer because if the input string belongs to $L_1 \cup L_2$, then one of the two simulations is guaranteed to accept and halt. Since we alternate steps, we are guaranteed to reach any fixed number of steps running on the input for either TM_1 or TM_2. If a string is not in the language, it is guaranteed that TM_3 will never accept it. (It may either reject when both simulations reject or loop.)

Grading: 2 pts for stating that we're going to design a TM to recognize this language.
2 pts for stating that there exist TMs that recognize both L_1 and L_2 and invoking these TMs in some way.
3 pts for alternating steps (or doing what a multitape really does, running both in parallel…)
1 pt for explaining why this machine is a recognizer

Note: This proof also works by creating an enumerator for the language. Assign points accordingly.

2) (8 pts) Prove that the following language C, is a decidable language:

C = { (<G>, k, m) | <G> is a Context Free Grammar such that at least k distinct strings are generated by applying m or fewer rules starting from the start variable. Both k and m are given positive integers.}

Applying a single rule means picking a single variable in the current derivation and substituting it with the right hand side of a rule for that variable. Please give your algorithm to decide membership in this language at a high level without Turing Machine level details and prove that your algorithm must terminate and return the correct result.

Solution

Start with the start variable. Recursively apply each possible rule to each variable in the current intermediary string, keeping track of how many rules have already been applied. If the intermediary string is a terminal string, see if this string has been previously derived by comparing it to all past derived strings. If not, add 1 to a counter. Stop recursing on any intermediary string which has already gone through m rule applications. After all possibilities have been tried, see if the running counter is k or greater. If so, accept, otherwise reject.

This process is guaranteed to halt because for any intermediary string, there is an upper limit to the number of derivations that can be applied. Therefore, eventually though the number might be large, there is a finite limit to the number of unique intermediary strings this algorithm will produce. For each unique intermediary string, some finite amount of work is done before proceeding.

Since the algorithm tries all possibilities, it will correctly solve the problem at hand and terminate.

Here is some very rough pseudocode that illustrates the solution idea:

```
void genStrings(String intermediate, int rulesApplied, Set set) {
    if (isTerminal(intermediate)) set.add(intermediate)
    if (rulesApplied == m) return
    for (String next: applyRule(intermediate))
        genStrings(next, rulesApplied+1)
}

boolean isPartOfC(Grammar G, int k, int m) {
    set = new Set()
    genStrings("S", 0, set)
    return set.size() >= k
}
```

Grading: 4 pts for stating that there are a finite # of derivations to try and roughly explaining how to do this (just noting that since m is fixed each branch of derivation will eventually stop), 3 pts for checking each terminal derivation and adding 1 to a counter for each new one, 1 pt for counting the number of distinct strings and accepting if this is k or greater.

3) (9 pts) Let L = { <M_1, M_2, k> | M_1 and M_2 are Turing Machines for which there exists a string s such that |s| = k and both M_1 and M_2 accept s. } Prove that L is undecidable from first principles (without using Rice's Theorem or any other short cut).

Solution

Assume the opposite, that L is decidable. Then, there exists some Turing Machine, call it M_L, that decides membership in L. We will now use M_L to design a Turing Machine M' which decides membership in A_{TM} as follows:

```
boolean MPrime(TM M, String w) {

    1. Generate a TM MX, which accepts w and no other string.
    2. Run ML( M, MX, |w|) and accept if it accepts, reject
       if rejects.

}
```

The key here is that the machine MX only accepts one string of length |w|, and that string is w. Thus, if M_L accepts, we know for a fact that M must accept w. Alternatively, if M_L rejects, then we know that M does not accept w and can reject accordingly.

Since A_{TM} is known to be undecidable, there must be an error with the proof. But the only step that wasn't justified was the initial assumption that L was decidable. Thus, this assumption must be faulty itself, proving that L is undecidable, as desired.

Grading: 2 pts for setting up proof by contradiction and assuming a TM to decide L.
2 pts for picking a known undecidable language to decide membership in.
4 pts for the actual construction details of the TM to decide the known Undecidable problem
1 pt to conclude the proof by contradiction