Introduction to Algorithms **Solution Set 9**
CS 4820, Spring 2014

1. *(10 points)* The class **NP** is defined in terms of polynomial-time verifiers and polynomial-length certificates. In this exercise, we will see that semidecidable problems can be characterized in terms of verifiers and certificates of unbounded length. (A problem is called *semidecidable* if the set of YES instance is recursively enumerable.)

Show that a problem A is semidecidable if and only if there exists a Turing machine V that halts on every input and satisfies the following property: a string x is a YES instance of A if and only there exists some string y such that V on input $x\#y$ accepts.

Solution

($\Rightarrow$) Suppose A is semidecidable and let M be a Turing machine that accepts every YES instance of A and does not accept any NO instance of A. Design a Turing machine V as follows. On input $x\#y$ it first checks that y is a string of the form 1^t for some non-negative integer t. If not, it rejects immediately. Otherwise, it runs a simulation of M on input x, terminating the simulation after t steps if it is still running at that time. If M accepts x before or during step t of the simulation, then V accepts $x\#y$. Otherwise V rejects $x\#y$. Note that V halts on every input. (In fact, V runs in polynomial time because it halts after a number of steps that is linear in t and the length of x.) If x is a YES instance of A and t is the number of steps that it takes for M to accept x, then V will accept the string $x\#1^t$. If x is a NO instance of A, then V will not accept any string of the form $x\#y$, because the only case in which V accepts $x\#y$ is if M accepts x in its simulation, and our hypothesis that x is a NO instance implies that M will never accept x in the simulation.

($\Leftarrow$) Suppose V is a Turing machine with the stated property. The following construction designs a Turing machine M that accepts every YES instance of A and does not accept any NO instance of A. The machine M simply iterates through every binary string y in order of increasing length. Within an iteration, it simulates V running on input $x\#y$. If V accepts $x\#y$ then M accepts x. Otherwise M moves to the next iteration, i.e. the next binary string y. If x is a YES instance of A then there exists a string y such that V accepts $x\#y$, and M will accept x when it encounters the first such y in its main loop. If x is a NO instance of A then V never accepts any string of the form $x\#y$, so M never accepts x and instead it loops forever. We have shown that M accepts a string x if and only if x is a YES instance of A, hence A is semidecidable as claimed.

2. *(10 points)* Recall the halting problem: An instance of the halting problem is a string of the form $x\#y$ and the goal is to decide if the Turing machine M_x encoded by the string x halts on input y. (If M_x halts on y, then $x\#y$ is a YES instance. Otherwise, $x\#y$ is a NO instance.)

Show that every Turing machine fails to solve the halting problem on an infinite number of instances. (A Turing machine M *fails to solve* the halting problem for instance $x\#y$ if it loops on input $x\#y$ or if it produces the wrong answer, i.e., it rejects in case that $x\#y$ is a YES instance or it accepts in case that $x\#y$ is a NO instance.)

Solution

Suppose, by way of contradiction, that M is a Turing machine and that the set F, consisting of all halting problem instances which M fails to solve, is finite. Define a new Turing machine M' which operates as follows. On input $x\#y$, M' first checks whether $x\#y$ belongs to F. If so, M' outputs the correct halting problem solution for $x\#y$; in other words, it accepts or rejects $x\#y$ according to whether

or not M_x halts on input y. To achieve this, both the contents of the set F and the list of correct halting problem solutions for every input $x\#y \in F$ are hard-coded into M' (i.e. into its state set and transition rules), which is possible because F is a finite set. In case $x\#y$ does not belong to F, then M' simulates M on input $x\#y$ and accepts or rejects $x\#y$ if and only if M does so.

We claim that M' accepts all strings $x\#y$ that are YES instances of the halting problem, and rejects all others; in other words, we claim that M' decides the halting problem. There are two cases: for strings $x\#y$ that belong to F, M' correctly decides the halting problem on $x\#y$ because the correct answer is hard-coded into M'; for strings $x\#y$ that do not belong to F, M' correctly decides the halting problem on $x\#y$ because M does so; this follows from the definition of the set F. Thus M' decides the halting problem, as claimed, yielding the desired contradiction because the halting problem is undecidable.

The preceding two paragraphs constitute a complete solution to the problem, but we would like to conclude the solution by emphasizing one counterintuitive aspect. When proving a problem is undecidable by reducing the halting problem to it, one assumes the reduction is given a description of some Turing machine, M, as the input to the first problem, and the reduction usually proceeds to transform this into the description of some related Turing machine M'. In such cases it is necessary to justify that there is an algorithm (i.e., a Turing machine) that can produce the description of M', given the description of M; otherwise the "reduction" cannot be implemented by an algorithm and hence cannot legitimately be called a reduction. In the solution above, we have not specified an algorithm to produce the description of M' given the description of M. Indeed, to do so would require finding the set F of halting problem instances that M fails to solve, and finding the correct halting problem solution on those instances — both of these tasks are in general undecidable. However, this issue doesn't invalidate our solution, because we are not solving this problem via reduction. We are solving it via a proof by contradiction. To yield the contradiction, we need not specify an algorithm for constructing a description of M' given the description of M, we only need to prove that M' exists, given the assumption that M exists.

Karma exercise. In this exercise, we will see that no single algorithm can be "best-possible" for the halting problem, in the sense that for every algorithm we can find an algorithm that solves the halting problem on infinitely more instances.

More formally, show that for every Turing machine M, there exists a Turing machine M' that solves the halting problem on all instances that M solves plus an infinite number of other instances. (A Turing machine *solves* the halting problem for instance $x\#y$ if it accepts in case $x\#y$ is a YES instance or if it rejects in case $x\#y$ is a NO instance.)

Solution

The solution to this part will be written at a later date.

3. *(10 points in total)* Show that each of following problems is semidecidable but not decidable. (A problem is called *decidable* if the set of YES instance is recursive.)

3.a. *(5 points)* Given a string of the form $x\#y\#q$, determine if the Turing machine M_x on input y enters state q at some point during its computation.

Solution

Let us call this problem the "state reachability problem". State reachability is semidecidable because the following Turing machine M accepts every YES instance and does not accept any NO instance. On input $x\#y\#q$, M begins a universal Turing machine simulation of M_x running on input y. In each step of the simulation it checks whether M_x enters state q, and if so, it accepts $x\#y\#q$. If M_x halts in the

simulation, then M rejects $x\#y\#q$. Otherwise, M proceeds to the next step of the universal Turing machine simulation. Clearly, this machine M accepts $x\#y\#q$ if and only if M_x on input y enters state q at some point during its execution. Thus, state reachability is semidecidable.

To prove that the state reachability problem is undecidable, we reduce from the membership problem. Given an instance $x\#y$ of the membership problem, the reduction translates it to $x\#y\#t$, where t is the accept state of M_x, and it treats $x\#y\#t$ as an instance of the state reachability problem. To prove that this is a valid reduction, we need to show that $x\#y$ is a YES instance of the membership problem if and only if $x\#y\#t$ is a YES instance of the state reachability problem. By definition, $x\#y$ is a YES instance of the membership problem if and only if M_x accepts input y, or equivalently, M_x enters state t at some time during its execution on input y. But this is exactly the same definition of what it means for $x\#y\#t$ to be a YES instance of the state reachability problem. Thus, the membership problem reduces to the state reachability problem. Since the membership problem is undecidable, state reachability is undecidable as well.

3.b. *(5 points)* Given a string x, decide if the Turing machine M_x accepts at least one string y that contains a 0 in some position.

Solution

Let us call this problem the "accept string containing 0 problem" (ASC0P). To prove that ASC0P is undecidable, we use Rice's Theorem. The ASC0P problem, on input x, asks whether $L(M_x)$ has the property that it contains at least one string containing 0. To apply Rice's Theorem, we merely need to show that there is at least one Turing machine M such that $L(M)$ has this property, and at least one Turing machine M' such that $L(M')$ does not have the property. This is easy; for example, if M is a Turing machine that accepts every string then $L(M)$ has the property, and if M' is a Turing machine that rejects every string than $L(M')$ does not have the property. By Rice's Theorem, it follows that ASC0P is undecidable.

To prove that ASC0P is semidecidable, we now describe a Turing machine that accepts the YES instances of ASC0P and only those instances. The Turing machine, on input x, checks whether x is a valid Turing machine description. If not, it rejects x. Otherwise it runs a pair of nested loops. In its outer loop, it increments a counter t through the sequence $1, 2, \ldots$. In its inner loop, for a given value of t, it iterates through every binary string y that has length less than or equal to t and contains a 0 in some position. For a given t and y, our machine simulates M_x running for t steps on input y. If M_x accepts y in t or fewer steps then our machine accepts x. Otherwise, it continues to the next string y in its inner loop. To prove that this works, we must show two things. First, if there exists a y containing 0 such that M_x accepts y, then our machine accepts x; this is true, because if t_0 is the number of steps it takes for M_x to accept y, and $t_1 = |y|$, and $t = \max\{t_0, t_1\}$, then our machine will accept x during the loop iteration in which the outer loop counter is at t, and the inner loop input string is at y. Second, if our machine accepts x, then there exists a string y containing 0 such that M_x accepts y; this is straightforwardly true, because the only circumstance in which our machine accepts x is if it has simulated M_x on some input y containing 0, and M_x has accepted y in the simulation. Thus, our machine accepts all YES instances of ASC0P, and only those instances, which implies that ASC0P is semidecidable.