CS120: Intro. to Algorithms and their Limitations Prof. Salil Vadhan

Problem Set 10

Harvard SEAS - Fall 2021 *Due: Wed Dec. 1, 2021 (5pm)*

Your name:
Collaborators:
No. of late days used on previous psets:
No. of late days used after including this pset:

The purpose of this problem set is to practice proving that problems are unsolvable via reduction, and gain more intuition for the kinds of problems about programs that are unsolvable (through examples). Throughout this problem set, you may use some pseudocode in describing RAM and Word-RAM programs (like for loops), but be sure that the pseudocode can be implemented using actual RAM/Word-RAM commands that satisfy the constraints of the given problem (e.g. having no arithmetic overflows or being write-free).

1. (Undecidability of arithmetic overflows) An *arithmetic overflow* in the execution of a Word-RAM program is when the result of an arithmetic operation (addition or multiplication) results in a value larger than 2^w, where w is the current word size, so the result has to be taken modulo 2^w. Recall that in Lecture 19, we saw how SMT Solvers were able to find a bug due to arithmetic overflow in Binary Search truncated to two levels of recursion. In this problem, you will see that that finding arithmetic overflow errors in general programs is an unsolvable problem.

   (a) Give an algorithm that converts any RAM program P into an equivalent Word-RAM program P' that never has arithmetic overflow. That is, $P'(x) = P(x)$ for all inputs (arrays of natural numbers) x, and whenever P' carries out an operation $\texttt{var}_i = \texttt{var}_j \texttt{ op } \texttt{var}_k$, the result is always smaller than 2^w, where w is the current word size. (Hint: before each such operation $\texttt{var}_i = \texttt{var}_j \texttt{ op } \texttt{var}_k$ in P, insert loops that MALLOC many times. This is not the most efficient conversion—in practice, bignum arithmetic would be used to simulate RAMs by Word-RAMs—but this should be simpler for you to describe and analyze.)

   (b) Using Part 1a and the undecidability of HaltOnEmpty for RAM programs, prove that the following computational problem is unsolvable:

Input	: A Word-RAM program P
Output	: `accept` if P has an arithmetic overflow when run on input ε, `reject` otherwise

   Computational Problem ArithmeticOverflow

2. (Unsolvable Problems from Diophantine Equations) In this problem, you may (and should!) assume the unsolvability of Diophantine Equations.

   (a) A RAM Program $P = (V, C_0, \ldots, C_{\ell-1})$ is *write-free* if it has no write commands (i.e., no commands of the form $M[\texttt{var}_i] = \texttt{var}_j$). Give an explicit write-free RAM program

P such that for all $x_0, x_1 \in \mathbb{N}$, we have

$$P(x_0, x_1) = 16x_0^3x_1^2 + 9x_0^2x_1 + 5x_1^4.$$

Treat x_0, x_1 as variables in the program rather than given in memory and write the output to a variable y rather than to memory (since this is supposed to be a write-free program).

(b) Generalizing Part 2a, give an algorithm that, given a multivariate polynomial $p(x_0, x_1, \ldots, x_{n-1})$ whose coefficients are natural numbers, constructs a write-free RAM program $P_p(x_0, x_1, \ldots, x_{n-1})$ such that

$$P_p(x_0, x_1, \ldots, x_{n-1}) = p(x_0, x_1, \ldots, x_{n-1})$$

for all $x_0, x_1, \ldots, x_{n-1} \in N$. Here you may assume that p is given as a list L of tuples $(c, e_0, e_1, \ldots, e_{n-1})$ where $c, e_0, e_1, \ldots, e_{n-1} \in \mathbb{N}$ and

$$p(x_0, x_1, \ldots, x_{n-1}) = \sum_{(c,e_0,\ldots,e_{n-1})\in L} c \cdot x_0^{e_0}x_1^{e_1} \cdots x_{n-1}^{e_{n-1}}.$$

(c) Show that the following problem is unsolvable:

Input	: A finite set V of variables and a set of constraints of the form $\texttt{var}_i = c$ for $c \in \mathbb{N}$, $\texttt{var}_i = \texttt{var}_j$, or $\texttt{var}_i = \texttt{var}_j$ `op` $\texttt{var}_k$, where `op` $\in \{+, \times\}$
Output	: `accept` if there is an assignment of natural numbers to the variables in V that satisfies all of the constraints, `reject` otherwise

Computational Problem TheoryOfNaturals

(Hint: use the same ideas as in the construction of your program P_p from Part 2b. However note that the unsolvable DiophantineEquations problem refers to polynomials with *integer* coefficients, while this problem involves only arithmetic on natural numbers (i.e. nonnegative integers).)

This shows in particular that there is no general SMT Solver for the theory of the natural numbers (since an SMT instance for the theory of the natural numbers can include constraints like the above as clauses). However, if we don't allow multiplication of variables ("the *linear* theory of the natural numbers"), then there is an SMT Solver, and that is all we needed for the Binary Search example in Lecture 19.

(d) Using Part 2b, prove that it is undecidable to determine whether a write-free RAM program halts on ε. (Hint: first show how, given n and k, a write-free RAM program can enumerate over all tuples of natural numbers $(\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{n-1})$ such that $|\alpha_i| \leq k$ for all $i = 0, \ldots, n-1$. Again, recall that RAM programs only operate directly on natural numbers, not signed integers.)

(e) (challenge*) Show that, in contrast, the Halting Problem for write-free *Word*-RAMs is solvable. That is, there is an algorithm that given a write-free Word-RAM Program $P = (V, C_0, \ldots, C_{\ell-1})$ an input x, decides whether $P(x)$ ever halts. (Hint: The entire state of a computation of a (Word-)RAM program is determined by the current line

number, the values of all variables, and the contents of memory. Use this to compute a threshold $T(P, x)$ such that if P runs for more than $T(P, x)$ steps, then P must be in an infinite loop.)