CS1200: Intro. to Algorithms and their Limitations Anshu & Vadhan

Problem Set 9

Harvard SEAS - Fall 2024 *Due: Wed. Dec. 4, 2024 (11:59pm)*

Your name:

Collaborators and External Resources (please cite all collaborators, and any resources or tools that you used outside of the core course resources, which are lectures, sections, SREs, office hours, earlier problem sets and their solutions):

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. (recognizing solvability and unsolvability) Which of the following computational problems about programs are solvable? Justify your answers, for example by giving an algorithm to solve the problem or using a reduction to prove unsolvability.

(a)

Input	**:** A RAM program P
Output	**:** `yes` if there is a Turing machine M that solves the same computational problem as P, `no` otherwise

Computational Problem RAMvsTM

(b)

Input	**:** A Turing Machine M
Output	**:** `yes` if M halts and outputs `yes` on every input, `no` otherwise

Computational Problem AgreeableTM

(c)

Input	**:** A Word-RAM program P that solves the 3-coloring problem, a word-length w, and two graphs G and G'
Output	**:** `yes` if the running time of $P[w]$ on G is smaller than the running time of $P[w]$ on G', `no` otherwise

Computational Problem WhichGraphIsEasier

2. (undecidability of arithmetic overflows) An *arithmetic overflow* in the execution of a Word-RAM program $P[w]$ is when the result of an arithmetic operation (addition or multiplication) results in a value at least 2^w, where w is the word size, so the result has to be capped at $2^w - 1$. In Lecture 24, you will see how SMT Solvers are 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 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, for all inputs x,

- There exists a word length w such that $P'[w]$ halts on x without crashing if and only if P halts on x,
- For all word lengths w such that $P'[w]$ halts without crashing on x, its output $P'[w](x)$ equals the output $P(x)$, and
- For all word lengths w, whenever $P'[w](x)$ carries out an operation $\texttt{var}_i = \texttt{var}_j \texttt{ op } \texttt{var}_k$, the result is always smaller than 2^w.

Do not worry about the efficiency of your simulation (in contrast to Theorem 7.1.1 of Lecture 7, which does a fairly involved simulation in order to obtain the runtime as described; something much simpler suffices here). (Hint: Try to make $P'[w]$ crash or go into an infinite loop if an overflow would happen.)

(b) Using Part 2a and the undecidability of HaltOnEmpty for RAM programs, prove that the following computational problem is unsolvable:

Input	: A Word-RAM program P
Output	: `yes` if there is a word length w such that $P[w]$ has an arithmetic overflow when run on input ε, `no` otherwise

Computational Problem ArithmeticOverflow

(c) Show that the ArithmeticOverflow problem is solvable if we fix the word length w:

Input	: A Word-RAM program P and a word length w
Output	: `yes` if $P[w]$ has an arithmetic overflow when run on input ε, `no` otherwise

Computational Problem ArithmeticOverflowFixedWordLength

(Hint: consider the *state* of the computation of $P[w]$ when run on input ε, namely the current size and contents of memory, the values of all the variables, and the current line number. Note that if the state is ever repeated prior to halting, then P will never halt.)

(d) Discuss why your algorithm from Item 2c is not practical for real-world programs with word length $w = 64$. This (and the following item) motivate the use of SMT Solvers to find bugs like arithmetic overflows.

(e) (optional) Show that ArithmeticOverflowFixedWordLength is NP-hard. (In fact, it is much harder than NP-hard and is provably not in P, but this fact is beyond the scope of this course.)

3. (reflection) How have the ideas of this course enlarged your sense of what it means to do computer science? Be specific, pointing to the particular course content that made this change for you.

Note: As with the previous psets, you may include your answer in your PDF submission, but the answer should ultimately go into a separate Gradescope submission form.

Quick note on grading: Good responses are usually about a paragraph, with something like 7 or 8 sentences. Most importantly, please make sure your answer is specific to this class and your experiences in it. If your answer could have been edited lightly to apply to another class at Harvard, points will be taken off.

4. Once you're done with this problem set, please fill out this survey so that we can gather students' thoughts on the problem set, and the class in general. It's not required, but we really appreciate all responses!