Homework 5

Due Sunday, March 15 by 11:59pm

Handout 12
CSCI 334: Spring 2020

Turn-In Instructions

For this assignment, upload a PDF called `hw5.pdf`. For this assignment, you must use LaTeX. Please use the LaTeX starter template for the assignment which is in your repository.

Turn in your work using the Github repository assigned to you. The name of the Github repository will have the form `cs334hw5-<your user name>`. For example, my repository would be `cs334hw5-dbarowy`. You should have received an invite to commit in the repository in your email. If you did not receive this email, please contact me right away!

Honor code: You may collaborate with one or more people on this assignment, but you may not write code or solutions together. All submitted work must be your own original work. If you work with a partner, please submit a `collaborators.txt` file that includes their names.

This assignment is due on Sunday, March 15 by 11:59pm.

Reading

1. **(Required)** "Appendix A: Introduction to LaTeX"

Problems

Q1. (*30 points*) ..... Partial and Total Functions

For each of the following function definitions, (i) give the graph of the function. Say whether this is a (ii) partial function or a total function on the integers. If the function is partial, say where the function is (iii) defined and where it is (iv) undefined.

For example, take the function $\texttt{f(x)} = \texttt{if x} > 0 \texttt{ then x} + 2 \texttt{ else x}/0$

The graph of this function is the set of ordered pairs $\{\langle x, x+2\rangle \mid x > 0\}$. The function is partial. It is defined on all integers greater than 0 and undefined on integers less than or equal to 0.

Functions:

(a) $\texttt{f(x)} = \texttt{if x} < 10 \texttt{ then } 0 \texttt{ else f(x} - 2\texttt{)}$

(b) $\texttt{f(x)} = \texttt{if x} + 3 > 3 \texttt{ then x} + 4 \texttt{ else x}/0$

(c) $\texttt{f(x)} = \texttt{if sin(x)} > 0 \texttt{ then } 1 \texttt{ else f(x} + \pi\texttt{)}$

Q2. (*20 points*) ..... Detecting Errors

Evaluation of a Lisp expression can either terminate normally (and return a value), terminate abnormally with an error, or run forever. Some examples of expressions that terminate with an error are `(/ 3 0)`, division by 0; `(car 'a)`, taking the `car` of an atom; and `(+ 3 "a")`, adding a string to a number. The Lisp system detects these errors, terminates evaluation, and prints a message to the screen. Suppose that you work at a software company that builds software using Impure Lisp. Your boss wants to handle errors in Lisp programs without terminating the entire computation, but doesn't know how.

(a) You boss asks you to implement a Lisp construct `(error E)` that detects whether an expression `E` *will* cause an error. More precisely, your boss wants evaluation of `(error E)` to

i. halt with value `t` if evaluation of `E` *would* terminate in error, and

ii. halt with value `nil` otherwise.

Explain why it is not possible to implement the `(error E)` construct. If you find it helpful to write code to answer this question, please do.

(b) After you finish explaining why `(error E)` is impossible, your boss proposes an alternative. Instead, your boss wants you to implement a Lisp construct `(guarded E)` that either executes `E` and returns its value, or if `E` halts with an error, returns 0 without performing any side effects. This could be used to *try* to evaluate `E`, and if an error occurs, to use 0 instead. For example,

```
(+ (guarded E) E2)
```

will have the value of `E2` if evaluation of `E` halts in error, and the value of `E + E2` otherwise. Observe that unlike `(error E)`, evaluation of `(guarded E)` does not need to halt if evaluation of `E` does not halt.

i. How might you implement the `guarded` construct?

ii. What difficulties might you encounter? Keep in mind that your company uses Impure Lisp.

Q3. (*10 points*) ..... Garbage Collection

A *garbage collection algorithm* performs automatic cleanup of unused memory in a program. Modern programming language runtimes routinely perform garbage collection in order to dramatically simplify memory management. *Garbage* has the following definition.

> At a given point i in the execution of a program P, a memory location m is *garbage* if continued execution of P from i **will not** access location m again.

Nonetheless, garbage collection using the above definition of garbage is not computable. Instead, languages solve a simpler problem by using a slightly different definition of garbage:

> At a given point i in the execution of a program P, a memory location m is *definitely garbage* if continued execution of P from i **cannot** access location m again.

McCarthy's "mark sweep" algorithm uses this latter definition, because it only reclaims memory that is *impossible* to re-read.

Prove that garbage collection using the first definition is not computable. You should prove this fact using the "reductio ad absurdum" proof technique. Specifically, your proof should employ a reduction of another non-computable function to garbage collection. For example, you may rely on the fact that we *know* that the halting problem is not computable.

Assume that you have the following `isGarbage` function available in the standard library of the programming language of *your choice*.

```
boolean isGarbage(String p, String m, int i)
```

Calling `isGarbage` with the source code for program text `p`, variable name `m`, and line number `i` has the following behavior.

`isGarbage(p, m, i)`	returns `true` if `m` is garbage at line `i` of program `p`.
`isGarbage(p, m, i)`	returns `false` otherwise.

You may assume that `isGarbage` always halts. You may also assume that `p` is "simple" code that does not contain class or function definitions.

Q4. (*10 points*) ..... Computability of Total Functions

Recall the definition of a *total function*:

> A function `f(i)` is total if and only if `f` is defined for every input `i`.

Suppose you have the following LISP function, `tot`, at your disposal:

```
(defun tot (f) ...)
```

where `f` is an arbitary function that takes one numeric argument. `tot` has the following behavior:

`(tot f)` returns `t` if `f` is a total function.
`(tot f)` returns `nil` if `f` is a partial function.

For example, calling `(tot #'atom)` returns `t` because `atom` is a total function. You may assume that `tot` *always halts*.

(a) Is `tot` computable?

(b) Why or why not?

Your proof should be in the form of a LISP program, and you should explain its behavior in plain English. Recall, as in the previous problem, that we *know* that the halting problem is not computable.

Hint: You may find it convenient to construct a lambda that takes one numeric argument in your proof, for example, `(lambda (i) <something>)`.

Q5. (*30 points*) ..... Project Brainstorming

For this part of the assignment, you should think of three different possible final projects you might explore. For your final project you will design and implement a programming language.

Note that such a programming language need not be a so-called *general purpose* programming language. A general purpose programming language is equivalent in expressive power to the lambda calculus (or Turing machines, if you've encountered these before). In fact, I strongly encourage you to explore *domain specific programming languages*, or DSLs. You have probably used a DSL without even realizing it. Some example DSLs are:

(a) `make`
(b) GraphViz
(c) Hypertext Markup Language (HTML)
(d) Extensible Markup Language (XML)
(e) Structured Query Language (SQL)
(f) Markdown
(g) LaTeX
(h) Scalable Vector Graphics (SVG)
(i) Csound

If you have a personal itch, scratch it. For example, I often have to grade student work. Therefore, the grading rubrics I supply to my teaching assistants are actually written in a domain specific language I designed called `tabulator`.

Be creative! I love music and art, especially abstract art. Can you make a language that generates music? Could you make a language that creates art? Some of your former classmates have designed languages that have done precisely these things and more.

For each potential language, describe

(a) What problem it solves. What does it do?

(b) What a sample program in the language might look like. Feel free not to be constrained by reality at this point.