CSCI-GA-2110 – Problem Set 5 – Written Part

This document describes the written exercises for problem set 5. Each exercise is designated as a "Task" in this document. Please write or type your solutions neatly to these tasks, produce a legible PDF clearly indicating where each question is answered, and upload the results to gradescope.

1 Typechecking Let without Annotations

In lecture 12's implementation of a type checker, we required a type annotation in `lambda` that gave the type of the argument to the function. Similarly, we also required a type annotation on the variable be defining in a `let` binding. We used the annotation for the `let` binding so that when desugaring `let` into `lambda`, we could use the annotation from `let` to fill in the type annotation for `lambda`.

In the programming part of this assignment, you have to implement type checking for `let` **without** a type annotaiton giving the type of the variable.

Task 1.1 (5 pts). Explain why it is easy to type check `let` without the annotation but more challenging to type check `lambda` without the annotation.

2 Polymorphism and Boxes

In the programming part of the assignment, you have to implement type checking for lists. We require the `empty` constructor to come with a type annotation so that we know what type the elements of the lists will be. However, in `plai-typed`, the `empty` constructor is polymorphic and does not take a type annotation. For example, if you just input `empty` at the REPL and hit enter you will get:

```
> empty
- (listof 'a)
'()
```

indicating that the `empty` value is polymorphic.

Consider the following two functions written in `plai-typed`

```
(define f1
  (let ([b (box empty)])
    (lambda (x)
      (set-box! b (cons x (unbox b))))))
```

```
(define f2
  (lambda (x)
    (let ([b (box empty)])
      (set-box! b (cons x (unbox b))))))
```

Task 2.1 (5 pts). What are the types of `f1` and `f2`? Why does `(begin (f1 #t) (f1 1))` cause a type error, but `(begin (f2 #t) (f2 1))` does not?