Homework 6: Types and Type-checking

Due November 1st at 10pm

There are two steps to submitting this assignment:

(1) Place your write-up in a Google Drive folder that is shared with me.

(2) Fill out the Homework 6 Submission Form.

Part 1: Extending our type-checker (40 points)

In this part of the homework, you will extend the type-checker we wrote in class to handle more language features.

1. Built-in operators (25 points)

Extend the type-checker to handle the following built-in operators:

1. **and** and **or**
2. **+** and **=**
3. **equal?**

2. Conditionals (15 points)

Now, extend the type-checker to handle conditional expressions: **if** and **cond**.

This question has 3 parts:

1. Implement the type judgment rule for **if** that we developed in class.
2. Write a type judgment rule for **cond**.
3. Implement your type judgment rule for **cond** in the type-checker.

Remember: for **cond**, you must be able to handle an arbitrary number of clauses!

Part 2: Type-checking lists (38 points)

Programming languages make different design decisions about lists. In some languages, lists must be homogenous: all members must have the same type. In other languages, lists are heterogenous: their members do not need to have the same type.

For example, **(list 1 2 3)** is homogenous, because all items are numbers, but **(list 1 #f 3)** is heterogenous, because it contains both booleans and numbers.

Heterogenous lists are harder to type-check than homogenous lists. For this problem, you will focus on homogenous lists.

1. Type judgments for lists (15 points)

Write a type judgement for homogenous lists and implement it in the type-checker. Represent them as:

```
'(homlist 1 2 3) ;; homogenous list
```

You may find the **(list-rest)** pattern for **match** helpful for this! It allows you to name list elements up to a point, and also name the tail of the list. For example, the following **match** names the first item of the list "first" and the tail "rest", and returns a new list with "ice cream" inserted after the first item.

```
(match '(1 2 3)
  ((list-rest first rest) (cons first
                                (cons "icecream" rest))))

> '(1 "icecream" 2 3)
```

Hint: you may also want to use higher-order functions!

2. Empty list (10 points)

What does your type-checker return for **(list)**?

The empty homogenous list presents a challenge for type-checking because we do not know what type the list will end up storing. You may be tempted to assign the type List. But consider what this will mean for the following function, which adds 1 to each number in a list:

```
(define (add-one l : '(list int)
     (if (empty? l)
          null
          (cons (+ 1 (first l))
                (add-one (rest l)))))
```

The base case cannot be reached without throwing a type error, because the empty list is not of type List[int].

Instead, you should use the type annotation approach: because the type-checker cannot statically determine what type will be added to an empty list, the programmer must provide a type annotation on any empty list.

Modify the type-checker so that it handles homogenous lists correctly.

3. List operators (8 points)

Our type-checker still faces one issue with the program above (aside from recursion, which we have not implemented): it uses the operator **empty?**, which can be applied to any list.

Write the type judgment for **empty?**.

4. List operator problems (7 points)

Now think about how you would implement type-checking for the following function, which also checks whether a list is empty.

```
(lambda (l) (if (equal? l null)  #t  #f))
```

Since **empty?** is a built-in operator, we can declare it syntactically invalid without an argument. But the function above is a value, and must be given a type.

What type annotation would you give the function above? You may propose new notation and formalism in your answer if you think it is necessary.

Sketch a suggestion for how the type-checker could be adapted to handle this case. What modifications would be necessary?

You do not need to implement anything for this question. Please describe your ideas in words.

Part 3: Explore list types in other languages (22 points)

Languages make different decisions about the list type problem that we ran into in Part 2. In this problem, you will explore how lists are typed by three different type checkers.

For each language, you must provide a **table of observations**. You should experiment with a variety of lists, list operators, and functions that involve lists, and observe how they are typed. You should also vary the contents of the lists: consider homogenous versus heterogenous, nested, and empty lists.

You must also write a **short summary of what you have found**. You should answer the following questions:

- Are heterogenous lists supported?
- Are lists mutable or immutable?
- What does the type checker do with empty lists?
- What approach does it take to the problem posed in Part 2, Question 4?

For each of these questions, if you have a hypothesis, explain the evidence that supports it. If not, describe what evidence you are lacking or what behavior remains unexplained.

Here are some suggested type checkers to explore:

- mypy: a type checker for Python (`https://mypy.readthedocs.io/en/stable/`)
- pyre: another type checker for Python (`https://pyre-check.org/docs/getting-started`)
- Typed Racket (`https://docs.racket-lang.org/ts-guide/index.html`)
- TypeScript: typed JavaScript (`https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html`)

- Scala (`https://docs.scala-lang.org/tour/tour-of-scala.html`)

You may also choose a different programming language if you like.

Extra Credit

1. list design decisions

Outline the pros and cons of heterogenous versus homogenous lists. If you are more used to using one kind, you may find it hard to see the advantages of the other. Try to list at least one benefit for each approach.

2. list operators

Implement support for **empty?** in the type-checker. You may use any strategy as long as it can type the program presented in Part 2, Question 5.