Homework 12: Conclusions

Due Thursday, May 14 at 9:55 am

Work on this assignment yourself. You may discuss the questions (but not the precise solutions) with other students, as on regular assignments per the syllabus instructions. The first half of the assignment is required and the second half is optional. I will grade optional questions that you answer to help you prepare for the final exam.

1. **Reflection (10 points)**
   Java's Reflection API (http://java.sun.com/docs/books/tutorial/reflect/index.html) makes classes and methods into first-class values. For example, you can take an instance of an object and get its class, and then create another instance from that class. You can also ask a class what members and methods its instances have, and call a method by name. For example, the following uses Reflection:

   ```
   String s = "Hi";
   Class c = s.getClass();
   Class d = Class.forName("java.lang.Boolean");
   Class e = d.getSuperclass();
   Method m = …; // (Some code to walk through the methods of String until compareTo is found)
   m.invoke(s, "Hello");
   ```

   a. What Scheme feature does java.lang.reflect.Method.invoke correspond to?

   b. Reflection allows you to write a method takes a Class as an argument, and then returns an instance of that class. Give a rigorous proof that this feature makes static type checking undecidable.

2. **Subtypes (25 points)**
   Assume the following classes have been implemented in C++:

   ```
   class Food {…};
   class Treat : public Food {…}; // This is the C++ inheritance syntax
   class Chocolate : public Treat {…};
   ```

   Note that C++ supports functions, which are just like methods except that they don't require an instance (i.e., like Scheme, ML, and Python functions—Java is one of the few languages we've seen that doesn't support this).

   a. C++ defines pointer subtypes to be covariant. That is, if S is a subtype of T, then S* is a subtype of T*. Write a formal type judgment for this.

   b. Give the normal subtype judgment for an expression (T-sub)

   c. Give the procedure subtype judgment (T-proc-sub)

   d. Write the types of the following procedures:

   ```
   Treat* cook(Treat* t) {…}
   Chocolate* bake(Food* f) {…}
   ```

   e. Prove that the type of *bake* is or is not a subtype of the type of *cook*.

3. **Life of a Program (10 points)**
   Describe the role and structure of a tokenizer, parser, and evaluator.

4. **Concurrency (5 points)**
   Briefly describe each of the following synchronization primitives and how it relates to the others:
   a. Atomic datatypes
   b. Spinlocks
   c. Mutexes
   d. Barriers
   e. Java's Synchronized statement

5. **Epiphany (5 points)**
   How will you program or approach algorithms differently as a result of taking CS334?

6. **Reductive Questions (10 points)**
   When I tell non-computer scientists about CS334, the first two questions they ask are invariably:

   1. What is your favorite programming language?
   2. How many programming languages do you know?

   a. Explain why both of these questions are missing the point of the course and computer science. (5 points)

   b. Give a course description in 256 or fewer words that would help non-computer scientists understand the point of CS334 and the relationship of programming languages to computer science. (5 points)

7. **The Big Picture (10 points)**
   Draw a picture and give a short explanatory caption and/or labels that illustrates something you've learned from the course. For example, in a Greek myth, of Daedulus makes wings of feathers and wax for he and his son Icarus to escape a prison. Icarus flies too close to the sun and his wings melt. A theme of this course is pushing expressivity as far as possible without crossing the line to undecidability, so I would draw Icarus flying near the sun, where the sun is labeled "Statically Undecidable" and drops of falling wax are things that "cross the line": "Y-combinator", "C++ Templates", "Java Array subtypes", "Downcast", etc. I will post the drawings in a public area.

Optional from here on…

8. BNF

Write a BNF grammar for e-mail addresses that can express the following examples:

morgan@cs.williams.edu	377..5@hotmail.com
steele@java.com	president@whitehouse.gov
Morgan.McGuire@williams.edu	_underscorer_@slashdot.org
-dingle@_.com	scott_mccann@2mail.f4st.111.org

and rejects the following:

bad#email.com	hello@world
funny/symbol@none.gov	jon@edu
illegal@domain.name	whole@lota@at.com
empty@..com	

Assume that the only legal top-level domains (TLDs) in this grammar are gov, edu, com, and org, and that there must be at least two period-separated names to the right of the @, and that those names must contain at least one character each. Assume that the only legal symbols in an e-mail address to the left and right of the @ are period, underscore, and dash (minus). (*This is all a simplification of real e-mail grammars to make the problem easier. If you happen to know the real rules...forget them while you're working on this problem! You can find the real grammar in RFCs 1034 and 822)* Remember to put quotes around terminals and angle-brackets around non-terminals. You may use the regular expression operators [ ] + * for convenience in addition to pure BNF grouping parentheses and the exclusive or operator, |. The following productions are provided:

```
<digit>  ::= `0' | `1' | `2' | `3' | `4' | `5' | `6' | `7' | `8' | `9'
<alpha> ::= 'a' | `b' | `c' | `d' | `e' | `f | `g' | `h' | `i' | `j' | `k' | `l' | `m' |
            `n' | `o' | `p' | `q' | `r' | `s' | `t' | `u' | `v' | `w' | `x' | `y' | `z' |
             'A' | `B' | `C' | `D' | `E' | `F' | `G' | `H' | `I' | `J' | `K' | `L' | `M' |
            `N' | `O' | `P' | `Q' | `R' | `S' | `T' | `U' | `V' | `W' | `X' | `Y' | `Z'
```

9. Next

a. Write code in Scheme that binds the identifier "next" to a procedure of no arguments that returns successively greater integers every time it is invoked. Your procedure may not use any global variables.

b. It is possible, but a little tricky, to perform the same operation in Python. How?

10. Sorting

Implement a sort in any language using only functional operations. Tip: of the languages we've studied, it is easiest in Python and Scheme.

11. Lazy

What are the advantages and disadvantages of lazy evaluation, from a language designer's perspective?

12. Lambda Calculus

Implement TRUE, FALSE, AND, OR, and NOT in Python using only LAMBDA and IF.

13. Macros

Write a feature specification for a macro language for Java. How should macros be defined? What features should they have? What features can be removed from Java if your system is added. Briefly explain each of your choices.

14. Passing Judgment

List, with brief supporting notes, positive and negative features of two languages.