Computer Science & Engineering 505 – Final Exam

December 14, 2001
Open book & notes – 110 minutes – 10 points per question
100 points total

Name: _____

Please use the back of the page if necessary for long answers.

1. Consider the CLP($\mathcal{R}$) queue problem from Assignment 3. The first question asked you to write predicates `put(Q0,I,Q1)` and `get(Q0,I,Q1)`. The `put` predicate puts an item `I` on a queue `Q0` to give `Q1`. The `get` predicate gets an item `I` from a queue `Q0` to give `Q1`. Here is the solution:

```
put(Q0, I, Q1) :- append(Q0, [I], Q1).
get([I|Q1], I, Q1).

/* convenience rule to make an empty queue, or test if a queue is empty */
empty([]).

/* reference definition of append */
append([],Ys,Ys).
append([X|Xs],Ys,[X|Zs]) :- append(Xs,Ys,Zs)
```

   If we try the goal

```
?- empty(Q0), put(Q0,100,Q1).
```

   this will succeed with the answer `Q0=[]`, `Q1=[100]`.

   What will happen when we try the following goal? (Say whether it succeeds or fails, and if it succeeds, give the answer.)

```
empty(Q0), put(Q0,100,Q1), put(Q1,200,Q2), get(Q2,I,Q3).
```

2. Now consider the difference list version of the queue rules.

```
/* queue(N,L,T) represents a queue of length N, where the list of elements
   is stored in L with T lopped off the end */
put_dl(queue(N,L,[I|T]), I, queue(N+1,L,T)).
get_dl(queue(N,[I|L],T), I, queue(N-1,L,T)) :- N>0.

/* convenience rule to build an empty queue, or test if a queue is empty */
empty(queue(0,T,T)).
```

What will happen when we try a similar goal for the difference list version?

```
empty(Q0), put_dl(Q0,100,Q1).
```

And what happens for this goal?

```
empty(Q0), put_dl(Q0,100,Q1), put_dl(Q1,200,Q2), get_dl(Q2,I,Q3).
```

3. The Pizza paper describes two techniques for translating Pizza into Java. What are they, and what are their advantages and disadvantages? (Please be brief.)

4. Consider the following expressions in the lambda calculus with let-polymorphism. Give the principal type of each expression, or say why it cannot be given a type.

   (a) λ f. λ x. (f (f x))

   (b) let id = λ x.x
   in ((id id) true)

   (c) λ id.((id id) true)

5. Consider the following Smalltalk class definition.

```
Object subclass: #BlockHolder
   instanceVariableNames: 'myblock'
   classVariableNames: ''
   poolDictionaries: ''

setBlock: b
   myblock := b

getValue
  ^ myblock value
```

What is the result of evaluating the following code? (The code evaluates without error. The value in each case will be the value of the last `s getValue` expression.)

(a)
```
| s |
s := BlockHolder new.
s setBlock: [3+4].
s getValue
```

(b)
```
| s a |
a := 3.
s := BlockHolder new.
s setBlock: [a+10].
a := a+1.
s getValue
```

6. Joe Mocha is defining an interface `Appendable` in Pizza that includes an `append` method. He then defines two classes, `MyString` and `MyList`, which both implement `Appendable`. He wants Pizza's type system to allow a `MyString` to be appended to a `MyString`, and a `MyList` to be appended to a `MyList`, but not a `MyString` to a `MyList`, or a `MyList` to a `MyString`.

   Here is his definition of `Appendable`:

   ```
   interface Appendable {
     Appendable append(Appendable a);
   }
   ```

   What is wrong with this definition? What is a correct one?

   Also write a definition for a class `MyString` that uses the revised definition of `Appendable`. (Just put ... in the body of the method — we only care about the header.)

7. Suppose that we have a Java class `Vehicle`, and another class `Bus` that extends `Vehicle`. Consider the following pieces of Java code. In each case, say whether the code runs correctly, gives a compile time error, or gives a run time error. If there is an error, identify the line on which it occurs.

(a)
```
Vehicle[] v = new Vehicle[10];
v[0] = new Bus();
v[1] = new Vehicle();
```

(b)
```
Bus[] b = new Bus[10];
b[0] = new Vehicle();
b[1] = new Bus();
```

(c)
```
Vehicle[] v = new Bus[10];
v[0] = new Vehicle();
v[1] = new Bus();
```

8. Suppose we are writing a simulation of a local network. We have two flavors of packet: an ordinary type `Packet`, and a subtype `EncryptedPacket`. We also have a type `Ethercard` that has a `receive` method with an argument of type `Packet`, and another type `DecryptingEthercard` that defines a `receive` method with an argument of type `EncryptedPacket`.

   We might sketch these classes as follows (using Java-like syntax):

```
class Packet
{ ....
   String getHeader () { ... }
}

class EncryptedPacket
{ ...
   String getHeader () { ... }
   int keylength () { ... }
}

class Ethercard
{
    void receive(Packet p)
    { ...
       h = p.getHeader();
      ...
    }
}

class DecryptingEthercard
{
    void receive(EncryptedPacket p)
    { ...
       h = p.getHeader();
       k = p.keylength();
      ...
    }
}
```

   Using structural subtyping rules (i.e. the ones given in class to define covariance and contravariance), clearly `EncryptedPacket` is a subtype of `Packet`.

   (a) What is the subtyping relation, if any, between `Ethercard` and `DecryptingEthercard` under the contravariant typing rule?

   (b) What is the subtyping relation if any between `Ethercard` and `DecryptingEthercard` under the covariant typing rule?

   If one of the rules gave an unsound result, describe an example that is statically correct but that has a runtime type error. (Continue your answer on the other side if necessary.)

9. Suppose we're given that Int is a subtype of Num. Formally, we add the following rule to the structural subtyping rules given in class for the simply typed lambda calculus with records:

$$\frac{}{\text{Int} \leq \text{Num}} \text{ (S-IntNum)}$$

Show a derivation of the fact that $\{x{:}\text{Num}\to\text{Int}, y{:}\text{Int}\}$ is a subtype of $\{x{:}\text{Int}\to\text{Num}\}$.

10. I’m deciding whether to make my new programming language be statically typed or dynamically typed. What benefits would I get by choosing static typechecking over dynamic typechecking? What benefits would I get by choosing dynamic typechecking over static typechecking?