CMSC330 Spring 2010 Practice Problems 8 Solutions

1. Operational semantics

Use operational semantics to determine the values of the following OCaml codes:

a. \[ 1 \]
   \[ 1 \rightarrow 1 \]

b. \[ + 3 7 \]
   \[ 3 \rightarrow 3 \]
   \[ 7 \rightarrow 7 \]
   \[ + 3 7 \rightarrow 10 \]

c. \[ + 1 ( + 2 3 ) \]
   \[ 1 \rightarrow 1 \]
   \[ ( + 2 3 ) \rightarrow 3 \]
   \[ + 1 ( + 2 3 ) \rightarrow 5 \]

d. \[ (\text{fun } x = 4) \]
   \[ \cdot ; (\text{fun } x = 4) \rightarrow (\cdot , \lambda x.4) \] // evaluate function to produce a closure
   \[ \cdot ; 5 \rightarrow 5 \] // evaluate the argument
   \[ (x:5; 4) \rightarrow 4 \] // evaluate body of closure, after extending
   \[ (\text{fun } x = 4) 5 \rightarrow 4 \] // environment w/ binding for parameter

\[ (\text{fun } x = + x 6) \]
   \[ \cdot ; (\text{fun } x = + x 6) \rightarrow (\cdot , \lambda x. + x 6) \] // evaluate function to produce a closure
   \[ \cdot ; 7 \rightarrow 7 \] // evaluate the argument
   \[ (x:7; + x 6) \rightarrow 13 \] // evaluate body of closure, after extending
   \[ (\text{fun } x = + x 6) 7 \rightarrow 13 \] // environment w/ binding for parameter

f. \[ (\text{fun } x = (\text{fun } y = + y x)) \]
   \[ \cdot ; (\text{fun } x = (\text{fun } y = + y x)) \rightarrow (\cdot , \lambda x. (\text{fun } y = + y x)) \] // eval func
   \[ \cdot ; 8 \rightarrow 8 \] // eval arg
   \[ x:8 ; (\text{fun } y = + y x) \rightarrow (x:8, \lambda y.(+ y x)) \] // eval body
   \[ \cdot ; (\text{fun } x = (\text{fun } y = + y x)) 8 \rightarrow (x:8, \lambda y.(+ y x)) \] // eval func
   \[ \cdot ; 9 \rightarrow 9 \] // eval arg
   \[ x:8, y : 9 ; (+ y x) \rightarrow 17 \] // eval body
   \[ \cdot ; (\text{fun } x = (\text{fun } y = + y x)) 8 9 \rightarrow 17 \] // eval body
2. Programming languages
   a. Describe the difference between ad-hoc and parametric polymorphism.
      Ad hoc polymorphism applies to code supporting a finite range of types whose combinations must be specified, parametric polymorphism applies to code written without mention to type that can transparently support an arbitrary number of types.
   b. Describe the difference between HTML and XML.
      HTML tags are predefined and presentation-oriented, whereas XML tags are user defined and are intended for describing data and metadata.
   c. Describe the difference between query languages and programming languages.
      Query languages are designed to make requests to a database or information system, whereas programming languages are designed to express computations that can be performed by a machine.

3. Polymorphism
   Consider the following Java classes:
   ```java
   class A { public void a( ) { … } }
   class B extends A { public void b( ) { … } }
   class C extends B { public void c( ) { … } }
   ```
   Explain why the following code is or is not legal
   a. ```java
      int count(Set<A> s) { … } … count(new TreeSet<A>( ));
   ```
      Legal. Actual parameter type (Set<A>) matches formal parameter type (Set<A>)
   b. ```java
      int count(Set<A> s) { … } … count(new TreeSet<B>( ));
   ```
      Illegal. Actual parameter type (Set<B>) is not a subclass of formal parameter type (Set<A>), even though B is a subclass of A.
   c. ```java
      int count(Set s) { … } … count(new TreeSet<A>( ));
   ```
      Legal. Type erasure will cause formal parameter type (TreeSet<A>) to become TreeSet, which matches actual parameter type (Set).
   d. ```java
      int count(Set<? s) { … } … count(new TreeSet<A>( ));
   ```
      Legal. Actual parameter type (Set<A>) matches formal parameter type (Set<?), since ? matches A.
   e. ```java
      int count(Set<? extends A> s) { … } … count(new TreeSet<B>( ));
   ```
      Legal. Actual parameter type (Set<B>) matches formal parameter type (Set<? extends A>), since “? extends A” can match A and its subclasses B & C (classes that extend A, including A)
f. int count(Set<? extends B> s) { … } … count(new TreeSet<A>());
    Illegal. Actual parameter type (Set<A>) does not match formal parameter type (Set<? extends B>), since “? extends B” can match only B and its subclass C (classes that extend B, including B)

g. int count(Set<? extends B> s) { for (A x : s) x.a(); … }
    Legal. The actual parameter type (Set<? extends B>) indicates s contains elements of class B or its subclasses. So any element of s may be treated as an object of class B or its subclasses (e.g., C). The for loop treats elements of s as objects of class A, which is a superclass of B, and thus is legal (can use subclass in place of superclass).

h. int count(Set<? extends B> s) { for (C x : s) x.c(); … }
    Illegal. The actual parameter type (Set<? extends B>) indicates s contains elements of class B or its subclasses. So any element of s may be treated as an object of class B or its subclasses (e.g., C). The for loop treats elements of s as objects of class C, and is illegal since elements of s may be objects of class B (cannot use superclass in place of subclass).

i. int count(Set<? super B> s) { for (A x : s) x.a(); … }
    Illegal. The actual parameter type (Set<? super B>) indicates s contains elements of class B or its superclasses. So any element of s may be treated as an object of class B or its superclasses (e.g., A, Object). The for loop treats elements of s as objects of class A, and is illegal since elements of s may be objects of class Object (cannot use superclass in place of subclass).

j. int count(Set<? super B> s) { for (C x : s) x.c(); … }
    Illegal. The actual parameter type (Set<? super B>) indicates s contains elements of class B or its superclasses. So any element of s may be treated as an object of class B or its superclasses (e.g., A, Object). The for loop treats elements of s as objects of class C, which is not included and thus illegal.
4. Markup languages
   a. Creating your own XML tags, write an XML document that organizes the following information: 1-hour test on Spanish Monday in Jiménez worth 15%. 1-hour test on Computers Tuesday in CSIC worth 10%. 30-minute test on Computers Friday in AVW worth 5%.

   ```xml
   <testList>
     <test>
       <length>1 hour</length>
       <subject>Spanish</subject>
       <date>Monday</date>
       <location>Jiménez</location>
       <value>15%</value>
     </test>
     <test>
       <length>1 hour</length>
       <subject>Computers</subject>
       <date>Tuesday</date>
       <location>CSIC</location>
       <value>10%</value>
     </test>
     <test>
       <length>30 minute</length>
       <subject>Computers</subject>
       <date>Friday</date>
       <location>AVW</location>
       <value>5%</value>
     </test>
   </testList>
   or
   <testList>
     <test subject="Spanish">
       <length unit="hour">1</length>
       ...
     </test>
     <test subject="Computers">
       <length unit="hour">1</length>
       ...
     </test>
     <test subject="Computers">
       <length unit="minute">30</length>
       ...
     </test>
   </testList>
   ```