

CMSC330 Spring 2010 Practice Problems 8 Solutions

1. Operational semantics

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

a. 1

$$\cdot \frac{}{1 \rightarrow 1} \cdot$$

b. + 3 7

$$\frac{3 \rightarrow 3 \quad 7 \rightarrow 7}{+ 3 7 \rightarrow 10}$$

c. + 1 (+ 2 3)

$$\frac{1 \rightarrow 1 \quad \frac{2 \rightarrow 2 \quad 3 \rightarrow 3}{(+ 2 3) \rightarrow 5}}{+ 1 (+ 2 3) \rightarrow 6}$$

d. (fun x = 4) 5

$$\frac{\begin{array}{l} \bullet ; (\text{fun } x = 4) \rightarrow (\bullet, \lambda x.4) \quad // \text{ evaluate function to produce a closure} \\ \bullet ; 5 \rightarrow 5 \quad // \text{ evaluate the argument} \\ \hline (x:5; 4) \rightarrow 4 \quad // \text{ evaluate body of closure, after extending} \\ // \text{ environment w/ binding for parameter} \end{array}}{(\text{fun } x = 4) 5 \rightarrow 4}$$

e. (fun x = + x 6) 7

$$\frac{\begin{array}{l} \bullet ; (\text{fun } x = + x 6) \rightarrow (\bullet, \lambda x. + x 6) \quad // \text{ evaluate function to produce a closure} \\ \bullet ; 7 \rightarrow 7 \quad // \text{ evaluate the argument} \\ \hline (x:7; + x 6) \rightarrow 13 \quad // \text{ evaluate body of closure, after extending} \\ // \text{ environment w/ binding for parameter} \end{array}}{\bullet ; (\text{fun } x = + x 6) 7 \rightarrow 13}$$

f. (fun x = (fun y = + y x)) 8 9

$$\frac{\begin{array}{l} \bullet ; (\text{fun } x = (\text{fun } y = + y x)) \rightarrow (\bullet, \lambda x. (\text{fun } y = + y x)) \quad // \text{ eval func} \\ \bullet ; 8 \rightarrow 8 \quad // \text{ eval arg} \\ \hline x:8; (\text{fun } y = + y x) \rightarrow (x:8, \lambda y. (+ y x)) \quad // \text{ eval body} \\ \bullet ; (\text{fun } x = (\text{fun } y = + y x)) 8 \rightarrow (x:8, \lambda y. (+ y x)) \quad // \text{ eval func} \\ \bullet ; 9 \rightarrow 9 \quad // \text{ eval arg} \\ \hline x:8, y:9; (+ y x) \rightarrow 17 \quad // \text{ eval body} \end{array}}{\bullet ; (\text{fun } x = (\text{fun } y = + y x)) 8 9 \rightarrow 17}$$

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:

```
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. `int count(Set<A> s) { ... } ... count(new TreeSet<A>());`

Legal. Actual parameter type (Set<A>) matches formal parameter type (Set<A>)

- b. `int count(Set<A> s) { ... } ... count(new TreeSet<B>());`

Illegal. Actual parameter type (Set) is not a subclass of formal parameter type (Set<A>), even though B is a subclass of A.

- c. `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. `int count(Set<?> s) { ... } ... count(new TreeSet<A>());`

Legal. Actual parameter type (Set<A>) matches formal parameter type (Set<?>), since ? matches A.

- e. `int count(Set<? extends A> s) { ... } ... count(new TreeSet<B>());`

Legal. Actual parameter type (Set) 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%.

```
<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>A VW</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>
```