

Written Assignment 5

Due March 9, 2006

This assignment asks you to prepare written answers to questions on type checking. Each of the questions has a short answer. You may discuss this assignment with other students and work on the problems together. However, your write-up should be your own individual work. Remember that written assignments are to be turned in either at the start of lecture or in the CS164 homework box in 283 Soda by 12:30 PM on the due date.

Please write your name, your account name, your TA's name, and your section time on your homework! We need this information so that we can give you credit for the assignment and so that we can return it to you.

1. Show the full type derivation (as done in slide 49 in the lecture notes) for the following judgement:

$$O[\text{Bool}/x] \vdash x \leftarrow (\text{let } x:\text{Object} \leftarrow x \text{ in } x = x): \text{Bool}$$

2. Suppose we extend the grammar for Cool with a “**void**” keyword

$$\begin{array}{l} \text{expr} ::= \text{void} \\ \quad | \dots \end{array}$$

that is analogous to **null** in Java. (Currently objects are initialized to void if they have no other initializer specified, but there is no general-purpose **void** keyword.) We want to be able to use **void** wherever an object can be used, as in

```
let foo:Int <- if some_test
               then 5
               else void
               fi
in ...
```

Give a sound typing rule that we can add to the Cool specification to accommodate this new keyword.

3. Suppose we extend Cool with exceptions by adding two new constructs to the Cool language.

$$\begin{array}{l} \text{expr} ::= \text{try } \text{expr} \text{ catch } ID \Rightarrow \text{expr} \\ \quad | \text{throw } \text{expr} \\ \quad | \dots \end{array}$$

Here **try**, **catch** and **throw** are three new terminals. “**throw** *expr*” returns *expr* to the closest dynamically enclosing catch block. Note that since **throw** expression returns control to a different location, we do not really care about the context in which throw is used. For example, (**throw** *false*) + 2 is a valid Cool expression (However, note that (**throw** *false*) + (2 + *true*) is not a valid Cool expression). Following is an example that uses the try-catch and throw constructs.

```

try
  if some_test1 then throw 34
  else if some_test2 then throw ‘‘undefined error’’
  else do_something fi fi
catch x =>
  case x of
    x:Int => do_something1
    x:String => do_something2
  esac

```

The above program fragment executes “do_something1” (with x bound to the value 34) if “some_test1” evaluates to *true*. It executes “do_something2” (with x bound to the value “undefined error”) if “some_test1” evaluates to *false* but “some_test2” evaluates to *true*. It executes “do_something” if both “some_test1” and “some_test2” evaluate to *false*.

Give a set of new sound typing rules that we can add to the Cool specification to accomodate these two new constructs.

4. The Java programming language includes arrays. The Java language specification states that if s is an array of elements of class S , and t is an array of elements of class T , then the assignment $s = t$ is allowed as long as T is a subclass of S . This typing rule for array assignments turns out to be unsound. (Java works around the fact that this rule is not statically sound by inserting runtime checks to generate an exception if arrays are used unsafely. For this question, assume there are no special runtime checks.)

Consider the following Java program, which type checks according to the preceding rule:

```

class Mammal { String name; }

class Dog extends Mammal { void beginBarking() { ... } }

class Main {
  static public void main(String argv[]) {
    Dog x[] = new Dog[5];
    Mammal y[] = x;

    /* Insert code here */
  }
}

```

Add code to the `main` method so that the resulting program is a valid Java program (i.e., it type checks statically and so it will compile), but the program could result in an operation being applied to an inappropriate type when executed. Include a brief explanation of how your program exhibits the problem.