CS131 Compilers: Writing Assignment 3
Due Sunday, May 20, 2018 at 11:59pm

Rong Yuyang - 69850764

This assignment asks you to prepare written answers to questions on semantic analysis. 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 and you should indicate in your submission who you worked with, if applicable. Written assignments are turned in at the start of lecture. You should use the Latex template provided at the course web site to write your solution.

I worked with: (Name,ID), (Name,ID)...

Example for type rule in tex:

$$\frac{\dfrac{\dfrac{O[\mathrm{Bo/x}][\mathrm{Ob/x}](\mathrm{x}) = \mathrm{Ob} \quad O[\mathrm{Bo/x}][\mathrm{Ob/x}](\mathrm{x}) = \mathrm{Ob}}{O[\mathrm{Bo/x}][\mathrm{Ob/x}] \vdash \mathrm{x{:}\ Ob}}}{} \quad \dfrac{\dfrac{Reason1}{\text{Conclusion1}} \quad \dfrac{Reason2}{\text{Conclusion2}}}{\text{Conclusion}}}{} \tag{1}$$

1. (10 pts) Consider the following class definitions.

```
  class A {
i: Int
o: Object
b: B <- new B
x: SELF_TYPE
f(): SELF_TYPE {x}
  }
  class B inherits A {
g(b: Bool): Object { (* EXPRESSION *) }
  }
```

Assume that the type checker implements the rules described in the lectures and in the Cool Reference Manual. For each of the following expressions, occurring in place of (* EXPRESSION *) in the body of the method g, show the static type inferred by the type checker for the expression. If the expression causes a type error, give a brief explanation of why the appropriate type checking rule for the expression cannot be applied.

(a) `x`

(b) `self = x`

(c) `self = i`

(d) `let x: B <- x in x`

(e)
```
case o of
o: Int => b;
o: Bool => o;
o: Object => true;
esac
```

(a)

$$\frac{O(\mathrm{x}) = \text{SELF\_TYPE}_B}{O, M, C \vdash \mathrm{x{:}\ SELF\_TYPE}_B} \tag{2}$$

(b)

$$\frac{\begin{array}{l} O, M, C \vdash \text{x: SELF\_TYPE} \\ O, M, C \vdash \text{self: SELF\_TYPE} \\ \text{SELF\_TYPE = SLEF\_TYPE} \end{array}}{O, M, C \vdash \text{x = self: Bool}} \tag{3}$$

(c) Error, as var i has type Int while var x has type SELF_TYPE

(d)

$$\frac{\begin{array}{l} T_0' = \text{B} \\ O[\text{SELF\_TYPE/x}],\ M, C \vdash \text{x: } SELF\_TYPE_B \\ \text{B} \leq \text{B} \Rightarrow SELF\_TYPE_B \leq \text{B} \\ O[T_0'/\text{x}],\ M, C \vdash \text{x: B} \end{array}}{O, M, C \vdash \text{let x: B} \leftarrow \text{x in x: B}} \tag{4}$$

(e)

$$\frac{\begin{array}{l} O, M, C \vdash \text{o : Object} \\ O\ [\text{Int/o}],\ M, C \vdash \text{b : B} \\ O\ [\text{Bool/o}],\ M, C \vdash \text{o: Object} \\ O\ [\text{Object/o}],\ M, C \vdash \text{true: Bool} \end{array}}{O, M, C \vdash \text{case o of ... esac: Object}} \tag{5}$$

2. (10 pts) Show the full type *derivation tree* for the following judgement: (You can use Bo as the type Bool and Ob as the type Object)

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

To save space, let's identify some labels first:

$$e_0\text{: x} \rightarrow \text{let x:Object} \leftarrow \text{x in x = x}$$

$$\frac{\dfrac{\dfrac{\dfrac{\begin{array}{l} O[\text{Bool/x}][\text{Object/x}] \vdash \text{x : Bool} \\ O[\text{Bool/x}][\text{Object/x}] \vdash \text{x : Bool} \\ Bool = Bool \end{array}}{O[\text{Bool/x}][\text{Object/x}] \vdash \text{x=x : Bool}} \quad \begin{array}{l} T_0' = \text{Object} \\ O[\text{Bool/x}] \vdash \text{x: Bool} \\ \text{Bool} \leq \text{Object} \end{array}}{O[\text{Bool/x}] \vdash \text{x} \rightarrow \text{let x:Object} \leftarrow \text{x in x = x: Bool}}}{O[\text{Bool/x}] \vdash (\text{x} \rightarrow \text{let x:Object} \leftarrow \text{x in x = x}) \text{: Bool}} \quad \begin{array}{l} O(\text{x}) = \text{Bool} \\ \text{Bool} \leq \text{Bool} \end{array}}{O[\text{Bool/x}] \vdash \texttt{x} \rightarrow \texttt{(x} \rightarrow \texttt{let x:Object} \leftarrow \texttt{x in x = x):}\ \ \texttt{Bool}} \tag{6}$$

3. (10 pts) Suppose we extend the grammar for Cool with a "**void**" keyword

$$\begin{array}{rcl} expr & ::= & \textbf{void} \\ & | & \ldots \end{array}$$

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** whereever 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 accomodate this new keyword.

$$\frac{\forall \text{ T, Void} \leq \text{T}}{O, M, C \vdash \textbf{void} : \text{Void}} \tag{7}$$

4. (10 pts) Suppose we extend Cool with exceptions by adding two new constructs to the Cool language.

$$\begin{array}{rcl} \mathit{expr} & ::= & \textbf{try } \mathit{expr} \textbf{ catch } ID => \mathit{expr} \\ & | & \textbf{throw } \mathit{expr} \\ & | & ... \end{array}$$

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 accommodate these two new constructs.

$$\frac{\begin{array}{c} O, M, C \vdash e_0 : T_0 \\ O, M, C \vdash e_1 : T_1 \end{array}}{\textbf{try } e_0 \textbf{ catch } \text{id} \Rightarrow e_1 : T_0 \sqcup T_1} \tag{8}$$

$$\frac{O, M, C \vdash e : \text{T}}{\textbf{throw } e\text{: T}} \tag{9}$$

5. (10 pts) 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 preceeding 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 error being applied to an inappropriate type when executed. Include a brief explanation of how your program exhibits the problem.

```
1  class Main {
2      static public void main(String argv[]) {
3          Dog x[] = new Dog[5];
4          Mammal y[] = x;
5          y[0] = new Mammal();  // Crash. java.lang.ArrayStoreException.
6      }
7  }
8  class Mammal {
9      String name = "Name";
10 }
11
12 class Dog extends Mammal {
13     void beginBarking() {
14         System.out.println("Rua!");
15     }
16 }
```

Explaination:

y[0] and x[0] are referencing to the same piece of memory. Saving an $Mammal$ instance there is fine if it were y to judge, since $Mammal \leq Mammal$. But x may have a problem as $Dog \leq Mammal$ can not be satisfied, as a result, an Exception is thrown.

Further more, this problem cannot be static checked because line 5 writes:

```
        y[0] = new Mammal();
```

which works perfectly fine.

A easier way to think of this problem is to think y[0] as a duplicate of x[0] and replace y[0] with x[0] in that line:

```
        x[0] = new Mammal();
```

Then immediately everyone can tell the problem.

6. (10 pts) Now that you know why Java arrays are problematic, you decide to add an array construct to Cool with sound typing rules. An array containing objects of type A is declared as being of type Array(A) and one can create arrays in Cool using the new Array[A][e] construct, where e is an expression of type Int, specifying the size of the array. One can access elements in the array using the construct e1[e2] which yields the e2'th element in array e1, and one can insert elements into the array using the notation e1[e2] $\leftarrow$ e3. Finally, as in Java, an assignment from one array a to an array b does not make copies of the elements contained in a, but addresses of elements.

   (a) (2 pts) Give a sound subtype relation for arrays in Cool, i.e., state the conditions under which the subtype relation Array(T)$\leq$ T' is valid.

$$\frac{\begin{array}{l}\mathrm{T}' = \mathrm{Array}(\mathrm{T}'') \\ \mathrm{T}'' = \mathrm{T}\end{array}}{\mathsf{Array(T)} \leq \mathsf{T'}} \tag{10}$$

   (b) Give sound typing rules that are as permissive as possible for the following constructs:

      i. (2 pts) new Array[A][e]

$$\frac{\mathrm{O, M, C} \vdash \mathrm{e: Int}}{\mathrm{O, M, C} \vdash \mathrm{new\ Array[A][e]{:}\ Array[A]}} \tag{11}$$

      ii. (2 pts) e1[e2]

$$\frac{\begin{array}{l}\mathrm{O, M, C} \vdash \mathrm{e2{:}\ Int} \\ \mathrm{O, M, C} \vdash \mathrm{e1{:}\ Array[A]}\end{array}}{\mathrm{O, M, C} \vdash \mathrm{e1[e2]{:}\ A}} \tag{12}$$

      iii. (4 pts) e1[e2] $\rightarrow$ e3. Assume the type of the whole expression is the type of e1.

$$\frac{\begin{array}{l}\mathrm{O, M, C} \vdash \mathrm{e1{:}\ Array[A]} \\ \mathrm{O, M, C} \vdash \mathrm{e2{:}\ Int} \\ \mathrm{O, M, C} \vdash \mathrm{e3{:}\ A}\end{array}}{\mathrm{O, M, C} \vdash \mathrm{e1[e2]} \rightarrow \mathrm{e3{:}\ Array[A]}} \tag{13}$$