Concepts of Programming Language Design

Featherweight Java Exercises

Gabriele Keller

November 6, 2024

1. **Dynamic versus static method dispatch** In C#, in contrast to (Featherweight)Java, methods are dispatched statically. That is, if you have a method invocation, than the method is selected depending on the object's static type, not its actual, dynamic type.

   Which static and dynamic semantics rules would have to change if we want to implement static dispatch, without compromising type safety.

2. FeatherWeight Java Consider the following Featherweight Java code, where we assume the existence of integers and booleans, an equal (`==`) and modulo (`%`) operator on integer values, and the logic *and* operator (`&&`).

```
class Colour extends Object {
  int red;
  int green;
  int blue;

  Colour (int r, int g, int b) {
    super ();
    this.red   = r % 256;
    this.green = g % 256;
    this.blue  = b % 256;
 }

  bool isEqual (Colour col) {
     return (this.red == col.red && this.blue == col.blue && this.green == col.green);
  }

  bool isBoring () {return (this.blue == 0);}
}

class Point extends Object {
  int x;
  int y;

  Point (int x, int y) {
    super ();
    this.x = x;
    this.y = y;
  }

  bool isBoring () {return (this.x == 0);}

  bool isEqual (Point p) {return (p.x == this.x && p.y == this.y);}
}

class ColourPoint extends Point {
  Colour c;
```

```
  ColourPoint (int x, int y, Colour c) {
   super (x, y);
   this.c = c;
  }

  bool isBoring () {return (this.c.red == 0);}

  bool isEqual (Point p) {return (p.isEqual (this) && ((ColourPoint) p).c.isEqual (this.c));}
}
```

For each of the following four expressions, explain very briefly whether they are legal according to Featherweight Java's static semantics (if not, why?). If so, what would they evaluate to according the Featherweight Java's dynamic semantics? You do not need to provide a formal proof or derivation.

(a) `(new ColourPoint (10, 20, new Colour (0, 0, 0))).isBoring();`

(b) `((ColourPoint)(new Point (0, 0))).isBoring();`

(c) `(new Point (0,0)).isEqual (new ColourPoint (0, 1, new Colour (255, 255, 255)))`

(d) `(new ColourPoint (0, 0, Colour (255, 255, 255))).isEqual (new Point (0,0))`

What would happen if we replaced the `isEqual` method in the `ColourPoint` class with

```
bool isEqual (ColourPoint p) {return (this.c.isEqual (p.c));}
```

3. **Type safety and security** Software vulnerabilities like format string attack (uncontrolled format string) are not possbile in a truly type safe language. Why?

   What are the costs or down sides of type safety?