Appendix A

Implementation of Integer

Here is an example implementation of **Integer**, Haskell's arbitrary precision integral value type: it is essentially a wrapper around Java's **BigInteger** class. The copious uses of underscores is explained in Section ??.

```
import java.math.BigInteger;
    public class _Integer extends Data {
3
        public BigInteger value;
4
        public static _Integer _make_Integer(BigInteger x) {
5
            _Integer i = new _Integer();
6
            i.value = x;
            return i;
8
        public static _Integer _make_Integer(String x) {
10
            return _make_Integer(new BigInteger(x));
11
12
13
        public static _Integer add(_Integer x, _Integer y) {
14
            return _make_Integer(x.value.add(y.value));
15
16
        ... // Analogous functions for subtraction and multiplication
17
18
        public static boolean eq(_Integer x, _Integer y) { ... }
19
20
        public static String show(_Integer x) { ... }
21
   }
22
```

The _make_Integer(String) function is used by the compiler to construct Integer literals: it allows a Java _Integer object to be constructed from a Java string representation. For example, the bytecode that creates the Haskell literal 2 would load the string "2" from the constant pool then invoke the creation method:

```
ldc 210 // String 2
invokestatic 16 // Method

tmp/_Integer._make_Integer:(Ljava/lang/String;)Ltmp/_Integer;
```

The add, eq, etc. methods are Java implementations of the functions required by Haskell's Num, Eq and Show typeclass instances for Integer. For 'builtin' types, the implementation of these typeclass functions need to be given in Java, as they cannot be expressed in Haskell. Section ?? on Hooks covers this aspect of code generation in more detail.

Appendix B

The Function class

```
import java.util.ArrayList;
    import java.util.function.BiFunction;
   public class Function extends HeapObject {
        private BiFunction<HeapObject[], HeapObject[], HeapObject> inner;
        private HeapObject[] freeVariables;
6
        private ArrayList<HeapObject> arguments;
        private int arity = 0;
        private HeapObject result = null;
9
10
        public Function(BiFunction<HeapObject[], HeapObject[], HeapObject> inner, int arity,
        → HeapObject[] freeVariables) {
            this.inner = inner;
12
            this.arity = arity;
13
            this.freeVariables = freeVariables;
14
            arguments = new ArrayList<>();
15
16
17
        @Override
18
        public HeapObject enter() {
19
            // Check if we've got a cached value
20
            if (result != null) {
21
                return result;
22
            }
23
24
            if (arguments.size() < arity) {</pre>
25
                return this;
27
            else if (arguments.size() > arity) {
28
                try {
29
                    Function fun = (Function)inner
30
                         .apply(arguments.subList(0, arity).toArray(new HeapObject[0]),
31

    freeVariables)

                         .enter()
32
                         .clone();
33
                     for (HeapObject arg : arguments.subList(arity, arguments.size()))
34
                         fun.addArgument(arg);
                     result = fun.enter();
36
                     return result;
37
                }
                catch (CloneNotSupportedException e) {
39
                     throw new RuntimeException(e);
40
            }
42
            else {
43
```

```
result = inner.apply(arguments.toArray(new HeapObject[0]),
44

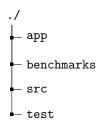
    freeVariables).enter();

                 return result;
45
            }
46
        }
47
48
        public void addArgument(HeapObject arg) {
            arguments.add(arg);
50
51
        @Override
53
        public Object clone() throws CloneNotSupportedException {
54
            Function f = (Function)super.clone();
55
            f.inner = inner;
56
            f.arity = arity;
57
            f.freeVariables = freeVariables.clone();
58
            f.arguments = new ArrayList<>(arguments);
59
            return f;
60
        }
61
   }
62
```

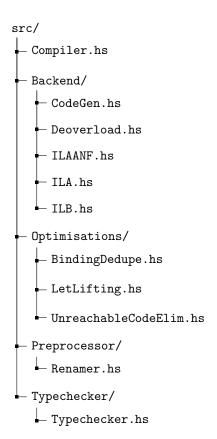
Appendix C

Repository Structure Diagrams

The top-level repository structure is within 4 directories:



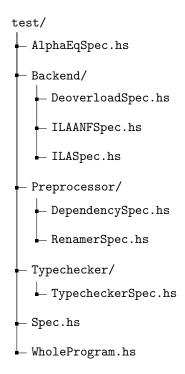
C.1 compiler



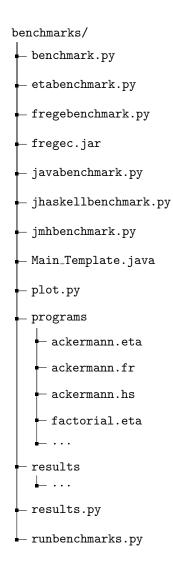
C.2 compiler-exe

```
app/
__Main.hs
```

C.3 Tests



C.4 Benchmarks



${ m C.5}$ hs-java

