Appendix A

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;
    private ArrayList<HeapObject> arguments;
    private int arity = 0;
    private HeapObject result = null;

    public Function(BiFunction<HeapObject[], HeapObject[], HeapObject> inner, int arity,
    ↪  HeapObject[] freeVariables) {
        this.inner = inner;
        this.arity = arity;
        this.freeVariables = freeVariables;
        arguments = new ArrayList<>();
    }

    @Override
    public HeapObject enter() {
        // Check if we've got a cached value
        if (result != null) {
            return result;
        }

        if (arguments.size() < arity) {
            return this;
        }
        else if (arguments.size() > arity) {
            try {
                Function fun = (Function)inner
                    .apply(arguments.subList(0, arity).toArray(new HeapObject[0]),
                    ↪  freeVariables)
                    .enter()
                    .clone();
                for (HeapObject arg : arguments.subList(arity, arguments.size()))
                    fun.addArgument(arg);
                result = fun.enter();
                return result;
            }
            catch (CloneNotSupportedException e) {
                throw new RuntimeException(e);
            }
        }
        else {
```

```
44              result = inner.apply(arguments.toArray(new HeapObject[0]),
                ↪  freeVariables).enter();
45              return result;
46          }
47      }
48
49      public void addArgument(HeapObject arg) {
50          arguments.add(arg);
51      }
52
53      @Override
54      public Object clone() throws CloneNotSupportedException {
55          Function f = (Function)super.clone();
56          f.inner = inner;
57          f.arity = arity;
58          f.freeVariables = freeVariables.clone();
59          f.arguments = new ArrayList<>(arguments);
60          return f;
61      }
62  }
```

Appendix B

Repository Structure

The top-level repository structure is within 4 directories:

```
./
├─ app
├─ benchmarks
├─ src
└─ test
```

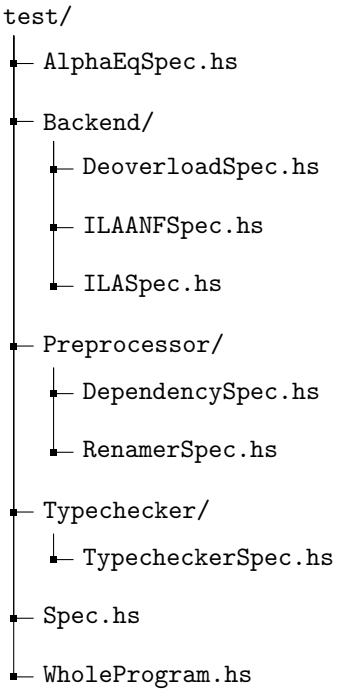
B.1 `compiler`

```
src/
├─ Compiler.hs
├─ Backend/
│  ├─ CodeGen.hs
│  ├─ Deoverload.hs
│  ├─ ILAANF.hs
│  ├─ ILA.hs
│  └─ ILB.hs
├─ Optimisations/
│  ├─ BindingDedupe.hs
│  ├─ LetLifting.hs
│  └─ UnreachableCodeElim.hs
├─ Preprocessor/
│  └─ Renamer.hs
└─ Typechecker/
   └─ Typechecker.hs
```

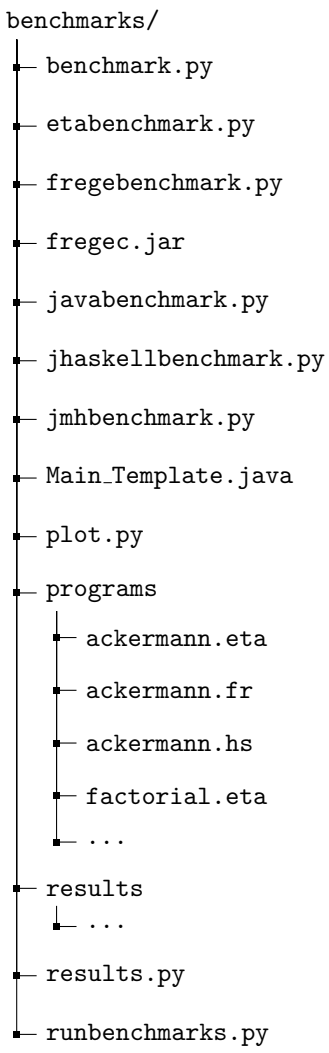
B.2 `compiler-exe`

```
app/
└─ Main.hs
```

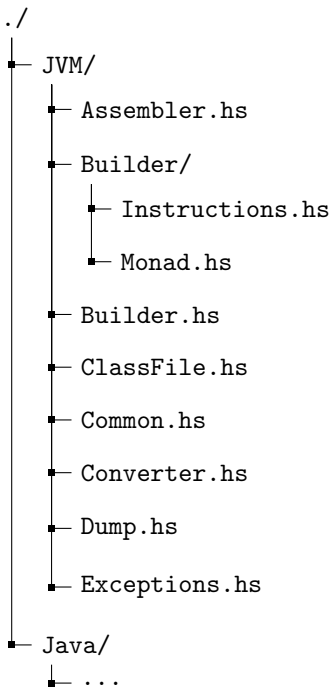
B.3 Tests


test/
AlphaEqSpec.hs
Backend/
DeoverloadSpec.hs
ILAANFSpec.hs
ILASpec.hs
Preprocessor/
DependencySpec.hs
RenamerSpec.hs
Typechecker/
TypecheckerSpec.hs
Spec.hs
WholeProgram.hs

B.4 Benchmarks



B.5 hs-java


./
JVM/
Assembler.hs
Builder/
Instructions.hs
Monad.hs
Builder.hs
ClassFile.hs
Common.hs
Converter.hs
Dump.hs
Exceptions.hs
Java/
...