Assignment 3/4 Review

Verification

- Static Dataflow Graph :
  - The stack layout must be the same for entry into a basic block
- General Wellformed-ness :
  - Referring to proper values in Constant Pool
  - Number of arguments in entry function
  - Out of bounds checks when referring to locals
  - Reference to labels outside of the local function
  - etc…
- Language Semantics :
  - Functions return single value
  - Etc…
- Type Checking :
  - Feeny is a dynamically-typed language. Not possible in general.

Environment Lookup

- Global and local variables have been preprocessed so that lookup is done via integer index!
- Lookup by name is reserved for the SLOT and CALL_SLOT operations.
- Give the user *useful* expressive power! Otherwise, don't design yourself into trouble.

Control Flow

- Implementing fine control flow operators are easy with a bytecode interpreter. Only the compiler changes.
- AST Interpreters are particularly easy to write only when the host language is close to isomorphic to the target language. (Eg. consider writing a Prolog AST Interpreter).
- For PL prelim: AST interpreter vs Bytecode Interpreter is analogous to Big-step semantics vs Small-step semantics.

Reverse Engineering

- It's surprising how much information you can recover isn't it!
- How much information to be retained is highly dependent upon the source language semantics. (E.g. method calls).
- What information is lost :
  - Local variable names.
  - High level control flow constructs. (Replaced with labels and gotos).
  - Any concept of scoping.

Semantic Differences

- What does this program do?

```
defn f () :
    val i = 0
    val i = 2
    println(“i = %d\n”, i)

println(“Hello World”)
```

Semantic Differences

- What does this program do?

```
f()
defn f () :
    println(“Hello World”)
```

- Seems like nitpicking. But stuff like this is what causes real portability issues in practice.
- Covering all the corner cases is not trivial:
  “The top level is hopeless” - Matthew Flatt
- Notice that any program that runs on bytecode interpreter will run on AST interpreter (but not vice versa).

Exposing the Bytecode IR

- What else needs to be specified if we expose the bytecode IR to the user?
  - How is it stored on disk? (Trivial)
  - What is valid and what is invalid bytecode? (Requires careful spec writing).

Labels vs Offsets

- Labels:
  - Avoids need for verification of offsets.
  - Compiler is easier to write.
- Offsets:
  - Can be interpreted directly?

- Takeaway: The semantics of the bytecode is orthogonal to how the bytecode is stored on disk. Don't make life hard for yourself by prioritizing the disk format over your semantics.

- Holy Grail of Language Design: *Any* syntactically correct program is both *correct* and *useful*.

Garbage Collector

- How's it coming?