Complications

June 2, 2017

The Standard Library

Polymorphic Comparison, Hashing, Serialization

We need to do our own polymorphic comparison, which must act exactly the same as that of the OCaml runtime. Two approaches: (a) Write our own, which would traverse the `Tinyocaml.t` datatype (b) Convert the data structures to be compared into the OCaml heap representation and call `compare_val` directly.

Interfacing with C

OCaml has a system for interfacing with C-like languages using `external` calls. We need to be able to (a) Convert any `Tinyocaml.t` into an OCaml heap object and (b) Convert any OCaml heap object back into `Tinyocaml.t`. Biggest complication: what happens when a closure is passed or returned?

Interfacing with the expectation of a Runtime

C code linked to OCaml code can create OCaml heap items, interact with the OCaml runtime, and make callbacks. So, there will be another OCaml runtime running (in addition to the interpreter's). How will we communicate with it? Or can we make a runtime shim and capture all the calls? Our own `lib(asm)run.a` effectively?

Threads and Processes

Calling the debugger from any build system