EJCP presentation

Raphaël Proust – raphael.proust@cl.cam.ac.uk

May 2013

High-level languages and low-level programming

- Q: What do we use C for?

High-level languages and low-level programming

- Q: What do we use C for?
- A: nostalgia, legacy, &c.

High-level languages and low-level programming

- Q: What do we use C for?
- A: nostalgia, legacy, &c.
- A: operating system's code

High-level languages and low-level programming

- Q: What do we use C for?
- A: nostalgia, legacy, &c.
- A: operating system's code
  - *real-life* OS code

High-level languages and low-level programming

- Q: What do we use C for?
- A: nostalgia, legacy, &c.
- A: operating system's code
  - *real-life* OS code
- A: runtime code for high-level language

High-level languages and low-level programming

- Q: What do we use C for?
- A: nostalgia, legacy, &c.
- A: operating system's code
  - *real-life* OS code
- A: runtime code for high-level language
  - Allocator, GC, VM, &c.

High-level languages and low-level programming

We use C because (sometimes) we need

- **predicatability** and
- **control**

about

- data-layout and
- memory management.

I.e. We use C because (sometimes) we do low-level things.

High-level languages and low-level programming

> *Computer science is the art of pilling up abstractions on top of one another (and bootstrapping them "because we can"^{TM})*

(Any one knows a quote along those lines (preferrably by someone famous)?)

High-level languages and low-level programming

Abstractions are useful.

Abstractions in Programming Languages are very useful.

Thus we *want to* use ML (or go, or Scala, or Scheme, or Haskell, &c.)

What my PhD is about

Making low-level programming in a high-level language possible.

Making a predictable ML:

- predicatble: low-level friendly
- ML: high-level language

Making a runtime-free ML: `RTFML`!

(Anyone has a suggestion for a better name? MoRTeL? MoRTadeL?) (Pun can also involve "system".)

Here are my weapons

- linear typing

Here are my weapons

- linear typing
  - for static (i.e. GCless, thus tag-less), automatic (i.e. with no `alloc`/`free`) memory management

Here are my weapons

- linear typing
  - for static (i.e. GCless, thus tag-less), automatic (i.e. with no `alloc`/`free`) memory management
- monomorphisation

Here are my weapons

- linear typing
  - for static (i.e. GCless, thus tag-less), automatic (i.e. with no `alloc`/`free`) memory management
- monomorphisation
  - for tag-less polymorphism

Here are my weapons

- linear typing
  - for static (i.e. GCless, thus tag-less), automatic (i.e. with no `alloc`/`free`) memory management
- monomorphisation
  - for tag-less polymorphism
- (light, limited) dependent types

Here are my weapons

- linear typing
  - for static (i.e. GCless, thus tag-less), automatic (i.e. with no `alloc`/`free`) memory management
- monomorphisation
  - for tag-less polymorphism
- (light, limited) dependent types
  - for safe, complex data layout

What am I doing right now

Designing the intermediate representations that RTFML will be compiled through.

That's it

(In the meantime, here are cool things I use:

- OCaml and rc,
- dwm, st, dvtm, dtach (and other tools from "suckless.org, software that sucks less"),
- acme and vim (and no other editor),
- Archlinux, cyanogenmod (and plan9 for fun and Debian when I need to),
- pentadactyl, git, darcs, mk, ssh, aria2c, sldm, xbindkeys, mplayer, mpd, llpp, &c.

)