`dependently-typed`

A programming languages club at Georgia Tech

Elton Pinto

April 19, 2023

Slides available at
https://github.com/dependently-typed/promo

Question for you

Have you ever wondered how your code magically runs?

```
1 def fibonacci(n):
2     if n <= 0:
3         return 1
4     else:
5         return fibonacci(n-1) + fibonacci(n-2)
6
```

Surprise, surprise. It's not magic :)

```
2           0 LOAD_FAST                0 (n)
            2 LOAD_CONST               1 (0)
            4 COMPARE_OP               1 (<=)
            6 POP_JUMP_IF_FALSE        6 (to 12)

3           8 LOAD_CONST               2 (1)
           10 RETURN_VALUE

5     >>   12 LOAD_GLOBAL              0 (fibonacci)
           14 LOAD_FAST                0 (n)
           16 LOAD_CONST               2 (1)
           18 BINARY_SUBTRACT
           20 CALL_FUNCTION            1
           22 LOAD_GLOBAL              0 (fibonacci)
           24 LOAD_FAST                0 (n)
           26 LOAD_CONST               3 (2)
           28 BINARY_SUBTRACT
           30 CALL_FUNCTION            1
           32 BINARY_ADD
           34 RETURN_VALUE
```

Figure: Python bytecode for `fibonacci`

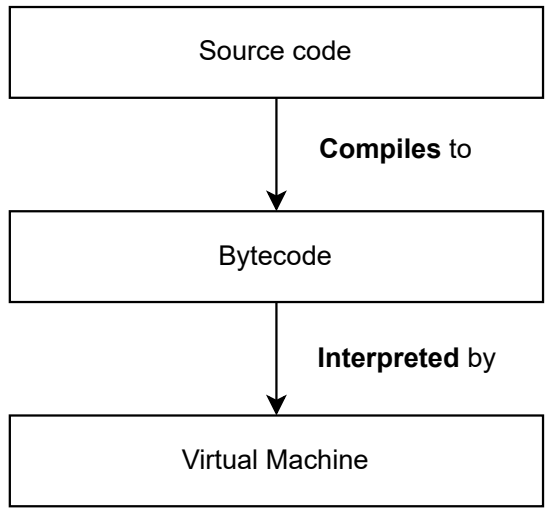

Figure: Python runtime overview

About the club

- Programming languages (PL) and compilers club
- Open to undergraduate and graduate students
- Weekly meetings on Tuesday, 6-7pm CCB 340

What do we do?

- Talks, workshops, and paper reading sessions
- Networking and community building
- Projects

Talks

- Student driven, with occasional guest speakers
- Member talks:
  - What the f*** is a monad?
  - Fast in-place interpreter for WebAssembly
  - Types and propositions / intro to constructive logic
- Guest talks:
  - Vivek Sarkar (chair of SCS) gave a talk about his research on parallelizing Python for large-scale data processing
  - Nathan Braswell gave a talk about his work on F-expression compilation
  - Sharjeel Khan gave a talk about his journey into PL research
- Schedule: https://dtyped.netlify.app/wiki/schedule

Why should I care?

- Programming languages and compilers are everywhere, and are here to stay.
- You will stand out. ‘Tis a very marketable set of skills to employers.
- The knowledge you will gain is broadly applicable to and is heavily used in other fields of CS.

If not for anything else...

You'll learn how to give *life* to the inanimate.

README.md

meow

An implementation of Gradual Typing for Functional Languages.

I started writing this on International Cat Day and decided to name the repo `meow` ;).

Some meow-difications from the paper:

- "meow", "😺", "😸", "😹", "😻", "😼", "😽", "🙀", "😿", "😾" are aliases for the `?` type
- "meow", "😺", "😸", "😹", "😻", "😼", "😽", "🙀", "😿", "😾" are also commands! They print out a random cat emoji ;)

You will need sedlex and menhirLib to build the project (in addition to the OCaml toolchain of course, >= 14.4.0).

Running the examples

Evaluate file:

```
dune exec bin/main.exe -- examples/<name-of-file>
```

Only run the typechecker

... and learn greek

$\underset{\uparrow}{\Delta} \mid \underset{\uparrow}{\Gamma} \vdash_s \underset{\uparrow}{e} \rightsquigarrow \underset{\downarrow}{e'}$ $\quad \Delta \cap \Gamma = \varnothing \quad \Gamma \subseteq \mathrm{fv}(e) \quad \mathrm{fv}(e) \subseteq \Delta, \Gamma \quad$ multiplicity of each member in Δ, Γ is 1

$$\frac{}{\Delta \mid x \vdash_s x \rightsquigarrow x}\ [\textsc{svar}] \qquad \frac{}{\Delta, x \mid \varnothing \vdash_s x \rightsquigarrow \mathrm{dup}\ x;\ x}\ [\textsc{svar-dup}]$$

$$\frac{\Delta, \Gamma_2 \mid \Gamma - \Gamma_2 \vdash_s e_1 \rightsquigarrow e_1' \quad \Delta \mid \Gamma_2 \vdash_s e_2 \rightsquigarrow e_2' \quad \Gamma_2 = \Gamma \cap \mathrm{fv}(e_2)}{\Delta \mid \Gamma \vdash_s e_1\ e_2 \rightsquigarrow e_1'\ e_2'}\ [\textsc{sapp}]$$

$$\frac{x \in \mathrm{fv}(e) \quad \varnothing \mid ys, x \vdash_s e \rightsquigarrow e' \quad ys = \mathrm{fv}(\lambda x.\ e) \quad \Delta_1 = ys - \Gamma}{\Delta, \Delta_1 \mid \Gamma \vdash_s \lambda x.\ e \rightsquigarrow \mathrm{dup}\ \Delta_1;\ \lambda^{ys} x.\ e'}\ [\textsc{slam}]$$

$$\frac{x \notin \mathrm{fv}(e) \quad \varnothing \mid ys \vdash_s e \rightsquigarrow e' \quad ys = \mathrm{fv}(\lambda x.\ e) \quad \Delta_1 = ys - \Gamma}{\Delta, \Delta_1 \mid \Gamma \vdash_s \lambda x.\ e \rightsquigarrow \mathrm{dup}\ \Delta_1;\ \lambda^{ys} x.\ (\mathrm{drop}\ x;\ e')}\ [\textsc{slam-d}]$$

$$\frac{x \in \mathrm{fv}(e_2) \quad x \notin \Delta, \Gamma \quad \Delta, \Gamma_2 \mid \Gamma - \Gamma_2 \vdash_s e_1 \rightsquigarrow e_1' \quad \Delta \mid \Gamma_2, x \vdash_s e_2 \rightsquigarrow e_2' \quad \Gamma_2 = \Gamma \cap (\mathrm{fv}(e_2) - x)}{\Delta \mid \Gamma \vdash_s \mathrm{val}\ x = e_1;\ e_2 \rightsquigarrow \mathrm{val}\ x = e_1';\ e_2'}\ [\textsc{sbind}]$$

$$\frac{x \notin \mathrm{fv}(e_2),\ \Delta,\ \Gamma \quad \Delta, \Gamma_2 \mid \Gamma - \Gamma_2 \vdash_s e_1 \rightsquigarrow e_1' \quad \Delta \mid \Gamma_2 \vdash_s e_2 \rightsquigarrow e_2' \quad \Gamma_2 = \Gamma \cap \mathrm{fv}(e_2)}{\Delta \mid \Gamma \vdash_s \mathrm{val}\ x = e_1;\ e_2 \rightsquigarrow \mathrm{val}\ x = e_1';\ \mathrm{drop}\ x;\ e_2'}\ [\textsc{sbind-d}]$$

$$\frac{\Delta \mid \Gamma_i \vdash_s e_i \rightsquigarrow e_i' \quad \Gamma_i = (\Gamma, \mathrm{bv}(p_i)) \cap \mathrm{fv}(e_i) \quad \Gamma_i' = (\Gamma, \mathrm{bv}(p_i)) - \Gamma_i}{\Delta \mid \Gamma, x \vdash_s \mathrm{match}\ x\ \{\ \overline{p_i \mapsto e_i}\ \} \rightsquigarrow \mathrm{match}\ x\ \{\ \overline{p_i \mapsto \mathrm{drop}\ \Gamma_i';\ e_i'}\ \}}\ [\textsc{smatch}]$$

$$\frac{\Delta, \Gamma_{i+1}, \ldots, \Gamma_n \mid \Gamma_i \vdash_s v_i \rightsquigarrow v_i' \quad 1 \leqslant i \leqslant n \quad \Gamma_i = (\Gamma - \Gamma_{i+1} - \ldots - \Gamma_n) \cap \mathrm{fv}(v_i)}{\Delta \mid \Gamma \vdash_s C\ v_1 \ldots v_n \rightsquigarrow C\ v_1' \ldots v_n'}\ [\textsc{scon}]$$

Fig. 8. Syntax-directed linear resource rules of λ^1.

... and maybe even get to geek about it

Neko: A quantum map-filter-reduce programming language

ELTON PINTO, Georgia Institute of Technology, USA

1 INTRODUCTION

Programming quantum computers is hard. One has to painstakingly write code that builds a circuit using low-level quantum gates [Svore et al. 2018] [Abraham et al. 2019] [Luo et al. 2020]. In a way, writing a quantum program is analogous to writing assembly: it is tedious, error-prone, and hard to debug. The gate-level abstraction, albeit universal, is non-intuitive and too primitive to be used for rapidly prototyping large-scale quantum applications. There is a need to develop high-level abstractions that enable programmers to productively leverage the idiosyncrasies of quantum computing: quantum parallelism, interference, and entanglement.

In this ongoing work, I present Neko, a high-level quantum programming language that exposes a map-filter-reduce interface for exploiting quantum parallelism through the notion of *first-class superpositions*.

Interested in getting involved?

Join the discord!

Contact: epinto6@gatech.edu
Website: dtyped.netlify.app