Applied research in Programming Language Theory

Gabriel Scherer – INRIA Saclay, France

GT Scalp
Friday November 5th, 2021

Introduction

A less-technical talk.

Applied research
(applying research to real-world problems)

in Programming Languages Theory
(formal design of programming languages)

1. What are we talking about?
   Some concrete examples.
2. Is this a good idea?
3. How would one get started?

Disclaimer

Policitians set up bad incentives and terrible metrics to favor applications: patents, software licensing, startups...

In the context of this political pressure,
any mention of applied research deserves a warning.

I'm not saying that people "should" do applied research.
I do it because it is *fun*.
Applied is not "better" than theoretical. and worse in several ways.

Be clear about what is theory and what is application,
and ask for good working conditions for both.

What are we talking about?

OCaml example: ambiguous pattern variables

```
| (Add (n, Zero) | Add (Zero, n)) -> n

| Add (n, z) when is_zero z -> n

| (Add (n, z) | Add (z, n)) when is_zero z -> n
```

Question: are some variables bound in different places in the or-alternatives of a when-guarded clause?

(Joint work with Luc Maranget and Thomas Refis)

OCaml example: recursive values

```
let rec fac = fun n ->
  if n = 0 then 1
  else n * fac (n - 1)
```

OCaml example: recursive values

```
let rec fac = fun n ->
  if n = 0 then 1
  else n * fac (n - 1)
```

```
type 'a stream = { head: 'a; tail: (unit -> 'a stream) }
let forever v =
  let rec s = { head = v; tail = tail }
  and tail () = s
  in s
```

OCaml example: recursive values

```
let rec fac = fun n ->
  if n = 0 then 1
  else n * fac (n - 1)
```

```
type 'a stream = { head: 'a; tail: (unit -> 'a stream) }
let forever v =
  let rec s = { head = v; tail = tail }
  and tail () = s
  in s
```

```
let rec wrong = wrong + 1
```

Recursive values

Idea: use a modal type system where the mode captures the way a value is used in a term.

$$x : \mathrm{Guard}, y : \mathrm{Dereference} \vdash t : \mathrm{Return}$$

(Joint work with Alban Reynaud and Jeremy Yallop)

OCaml example: constructor unboxing

$$A + B := (\{0\} \times A) \cup (\{1\} \times B) \qquad A + B \stackrel{?}{\simeq} (\{0\} \times A) \cup B$$

$$\mathsf{Int} := \{\mathsf{Imm}\} \times \mathbb{Z}/N\mathbb{Z} \qquad \mathsf{String} := \{\mathsf{Str}\} \times \dots$$

```
type bignum =
  | Large of Gmp.t
  | Small of int [@unboxed]
```

(joint work with Nicolas Chataing)

Constructor unboxing: normalization?

To detect conflit, you need to approximate the values of a type. Need: unfold type definitions.

```
type 'a foo = (int * 'a) bar
```

foo $:= \lambda\alpha.$ bar (prod int α)

Normalization in presence of mutually-recursive definitions?

(Help from Stephen Dolan and Irene Waldspurger)

Many other works and communities

Programming: OCaml, Haskell, SML, Scala, maybe Rust, Ceylon

Proving: Isabelle, Coq, Agda, Idris, Lean...

Verification: Why3, F$\star$...

Is this a good idea?

Pros

- students / collaborators
- advantages of perceived usefulness
  (some reward systems, funding (eg. CIFRE))
- gratifying feedback from users (including yourself)

Also: keeping in touch with computing.

Cons

- time-consuming
- theoretical tools not always that interesting
  (not impressive to yourself or others)
- risk of ugliness

Difficult balance

A natural idea to get the “best of both worlds”
(theoretical and applied research)
is to split your time between the two.

Note: no need for a clear connection between your applied and theoretical work.

My experience: difficult to do in practice.
(Requires willpower and organization.)

How would one get started?

Common misconceptions.

1. "I don't know enough about "real" programming."
   $\Rightarrow$ most master students don't either, yet they can contribute.

2. "This is for people who know what they are doing."
   $\Rightarrow$ most people don't know what they are doing... and we manage.

3. "I'm stronger at theory."
   How can you tell?

4. "Theory has a longer-lasting impact."
   True.

A recipe

1. Get some programming experience.
   (simple trick: write code for your own research)

2. Contribute to an existing project with users.
   (Follow user requests/needs.)

3. Jump in when a problem is calling for research.

Thanks!

Discussion ?