Gabriel Scherer
2017 - : Parsifal, INRIA Saclay
2016 - 2017: Northeastern University – with Amal Ahmed
2012 - 2015: Gallium, INRIA Rocquencourt – with Didier Rémy

Search for Program Structure

The Unreasonable Effectiveness of Mathematics in the Natural Sciences
Eugene Wigner, 1960

The miracle of the appropriateness of the language of mathematics for the formulation of the laws of physics is a wonderful gift which we neither understand nor deserve. We should be grateful for it and hope that it will remain valid in future research and that it will extend, [..] to our bafflement, to wide branches of learning.

The Unreasonable Effectiveness of Mathematics in the Natural Sciences
Eugene Wigner, 1960

The miracle of the appropriateness of the language of mathematics for the formulation of the laws of physics is a wonderful gift which we neither understand nor deserve. We should be grateful for it and hope that it will remain valid in future research and that it will extend, [..] to our bafflement, to wide branches of learning.

The Unreasonable Effectiveness of Mathematics in the Natural Sciences
Eugene Wigner, 1960

The miracle of the appropriateness of the language of mathematics for the study of programming languages is a wonderful gift which we neither understand nor deserve. We should be grateful for it and hope that it will remain valid in future research and that it will extend, [..] to our bafflement, to wide branches of learning.

Programming languages, formally

Model a program as a mathematical object.

Formal definitions of: execution, compilation, typing, errors...

Programming languages are “spaces” of programs.

Study the formal properties of these spaces.

Applications

Programming languages, features, and tools.

- develop new languages, features, tools
- study existing languages, features
- evolve existing languages, features

Expected benefits:

- correctness
- clarity
- simplicity

Test your design with theorems

Formalism lets us capture usability properties as theorem statements.

Test your design with theorems

Formalism lets us capture usability properties as theorem statements.

Examples:

- Determinism.
- Memory soundness.
- Type soundness.
- Type erasure.

Test your design with theorems

Formalism lets us capture usability properties as theorem statements.

Examples:

- Determinism.
- Memory soundness.
- Type soundness.
- Type erasure.

Guide language designers and tool authors.

Code is for the machine *and* humans. Theorems are the same.

Blind spots

No empirical evaluation.

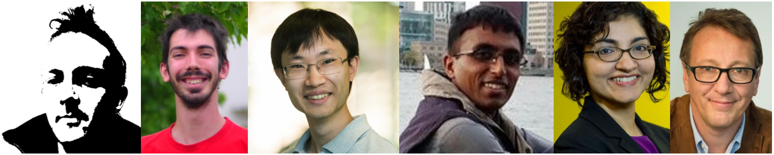
No study of cognitive aspects. (Surface syntax?)

No study of social factors. (Project management? Company adoption?)

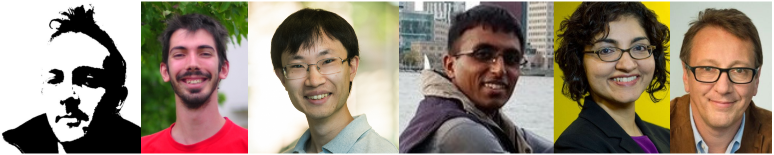
Plus the blind blind spots.

Yet: surprisingly, unreasonably effective.

Recent work (2017): JIT compilation



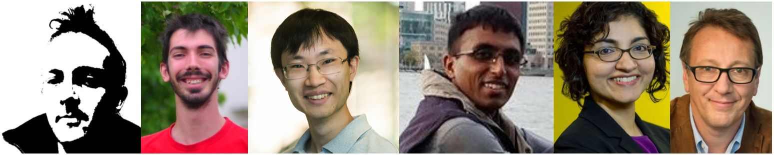
Recent work (2017): JIT compilation



Just-in-time (JIT) compilation for dynamic languages:

- code generation as the program is running
- speculative optimization
- deoptimization

Recent work (2017): JIT compilation

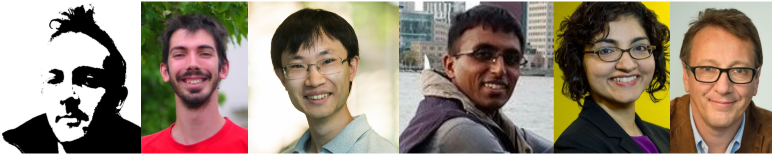


Just-in-time (JIT) compilation for dynamic languages:

- code generation as the program is running
- speculative optimization
- deoptimization

Is deoptimization correct? Many bugs in industrial implementations.

Recent work (2017): JIT compilation



Just-in-time (JIT) compilation for dynamic languages:

- code generation as the program is running
- speculative optimization
- deoptimization

Is deoptimization correct? Many bugs in industrial implementations.

Our approach:

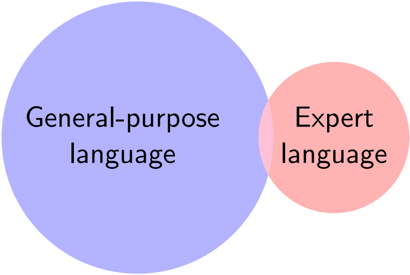
- formal model: small language with minimal features
- correctness proofs
- for humans: invariants, proof techniques

Recent work (2016-2017): graceful interoperation

What does it mean for two languages to "interact well together"?

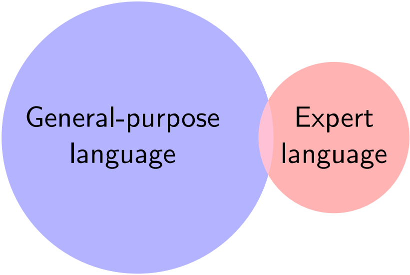
Recent work (2016-2017): graceful interoperation

What does it mean for two languages to “interact well together”?



Recent work (2016-2017): graceful interoperation

What does it mean for two languages to "interact well together"?



Full abstraction:

$$\forall (t_1, t_2 \in G), \quad t_1 \simeq_G t_2 \quad \Longrightarrow \quad t_1 \simeq_{G+E} t_2$$

Practice (2010-): OCaml

Typed functional programming language.

Good for manipulating symbolic representations.

Small but active community: thousands of programmers, research software, open source projects, companies, etc.

I co-maintain the language implementation
and some tools (`batteries`, `ocamlbuild`, `opam-repository`...).

It takes work, but keeps us programming.

Which mathematics?

We reuse the methodology of (some) mathematicians.

But few of their theories.
No analysis. Small bits of algebra, topology and category theory.

Mostly new mathematical objects.
(λ-calculi, type derivations, type theories)

Interactions with constructive logic and proof theory.

Why proof theory?

Study mathematical proofs as mathematical objects.

Logics: spaces of proofs.

Why proof theory?

Study mathematical proofs as mathematical objects.

Logics: spaces of proofs.

Curry-Howard correspondence:

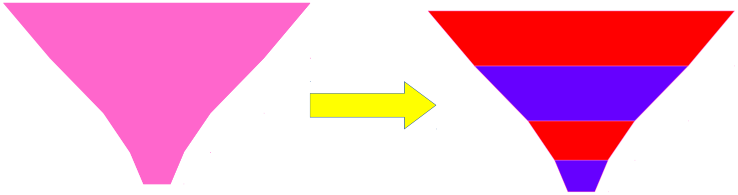
Logic	$\Longleftrightarrow$	Typed functional language
Formula	$\Longleftrightarrow$	Type
Proof	$\Longleftrightarrow$	Program
Cut elimination	$\Longleftrightarrow$	Execution

Logicians think about the structure of proofs a lot.

They design new representations to reduce redundancies.

(Redundancy: different syntaxes for “morally the same” proof).

Focusing



$\Gamma \vdash A$ $\qquad$ $\Gamma \vdash_{\mathrm{foc}} A$

Recent work (2015-): focusing on equivalence

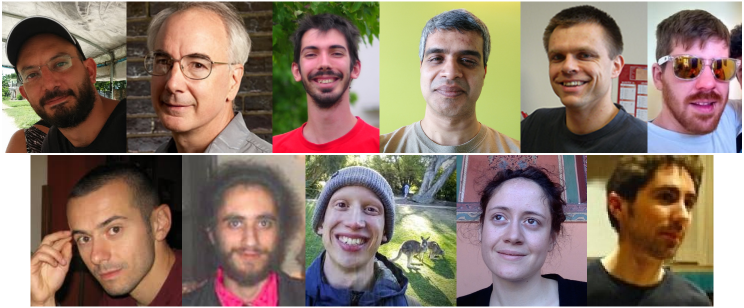
Design “focused” type system from these ideas.

Put programs in canonical (multi-)focused form.

Solved an open problem on decidability of program equivalence.

Applications: equivalence checking, type-directed program synthesis.

Parsifal



Proof theory, focusing, automated theorem proving, proof assistants.

Applied mostly to proof systems so far.

Me: expertise and application goals in programming languages.

Programming projects (Abella, Psyche, Bedwyr, Mætning...).
$\leftrightarrow$ OCaml expertise.