The *Next Frontier* in *Type Inference*

Lionel Parreaux, HKUST CSE
Research and Technology Forum 2022

About Myself

Lionel Parreaux Origin: France

PhD at **EPFL**, Lausanne

Sep 2014—Jun 2020

Thesis: ***Type-Safe Metaprogramming*** *and* ***Compilation Techniques*** *For Designing* ***Efficient Systems*** *in* ***High-Level Languages***

Assistant Professor at **HKUST**, Hong Kong

Since Feb 2021

Current focus:

- **Type inference** with advanced features
- Compiler optimization
- Dependent type systems, metaprogramming
- Performance-oriented software systems

Problem of Type Inference

An old dilemma

Static typing

```
List<Integer> foo(Integer init) {
     List<Integer> xs =
          List.of<Integer>(init);
     System.out.println(xs);
     return xs;
}
```

Dynamic typing

```
def foo(init):
     xs = List.of(init)
     System.out.println(xs)
     return xs
```

✅ more concise, readable
❌ error-prone
❌ slower to execute

Problem of Type Inference

The best of both worlds

Static typing + type inference

Infer type annotations at **compilation time**

Report possible **errors** to users early on

✅ more concise, readable
✅ type checked at compile time
✅ can compile to efficient code

Type Inference State of the Art

Two schools of type inference

in object-oriented languages

Incomplete, ad-hoc, often **unsound**

Still **require lots of annotations**

in functional languages

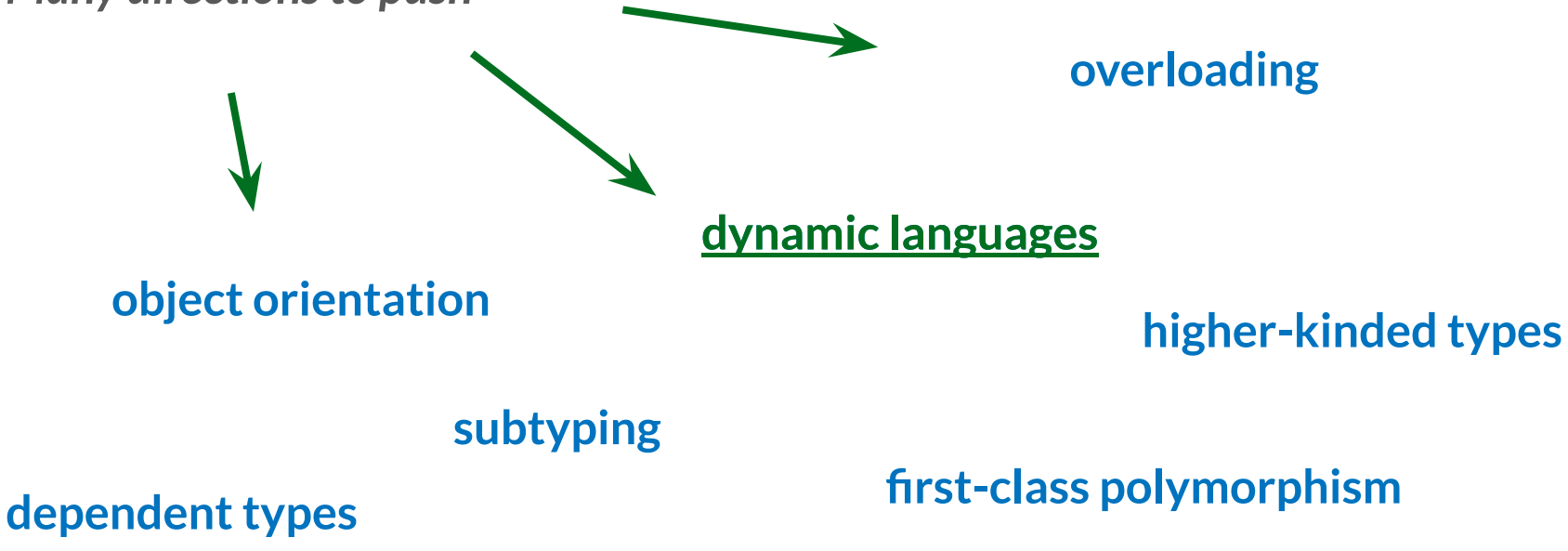
Solid **formal foundations**

Applies on **limited type systems**

my work: *bridge the gap*

Type Inference State of the Art

Many directions to push



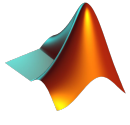
Type Inference for Dynamic Languages

Dynamic languages are moving towards static typing

JavaScript

TypeScript

Python



MatLab

Example: The MLscript language

Goal: be a better TypeScript

github.com/hkust-taco/mlscript

interoperable type system, with

- **sound type system**
- **formally-proven full type inference**
- **concise, functional syntax**

current contributors:

Lionel Parreaux
Assistant Professor

Luyu Cheng
PhD Student

Tony Chau
MPhil Student

Élise Rouille
MPhil Student

Example: The MLscript language

Goal: be a better TypeScript

github.com/hkust-taco/mlscript

interoperable type system, with

- **sound type system**
- **formally-proven full type inference**
- **concise, functional syntax**

web demo:

```
x = "oops"

f y = succ y

f x
```

```
val x: "oops"
val f: int -> int
╔══ [ERROR] Type mismatch in application:
╟── l.4: f x
╟── string literal of type `"oops"` does not match type `int`
╟── l.0: x = "oops"
╟── but it flows into reference with expected type `int`
╟── l.4: f x
╟── Note: constraint arises from reference:
╟── l.1: f y = succ y
╚══
```

Challenges of Type Inference

complexity, decidability

find sweet spot between ***expressiveness*** and ***complexity***

predictability

should be ***intuitive*** for users, easy to ***understand***

error messages

explain type errors in terms of user-level concepts