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**

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

**Dynamic typing**

```python
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*

<table>
<thead>
<tr>
<th>in object-oriented languages</th>
<th>in functional languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incomplete, ad-hoc, often <strong>unsound</strong></td>
<td>Solid <strong>formal foundations</strong></td>
</tr>
<tr>
<td>Still require lots of annotations</td>
<td>Applies on <strong>limited type systems</strong></td>
</tr>
</tbody>
</table>

**my work: bridge the gap**
Type Inference State of the Art

Many directions to push

- object orientation
- subtyping
- dependent types
- overloading
- dynamic languages
- higher-kindred types
- first-class polymorphism
Dynamic languages are moving towards static typing

Type Inference for Dynamic Languages

JavaScript

TypeScript

Python

MatLab
Example: The **MLscript** language

**Goal: be a better TypeScript**

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

[github.com/hkust-taco/mlscript](https://github.com/hkust-taco/mlscript)

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

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

github.com/hkust-taco/mlscript

web demo:

```typescript
x = "oops"
f y = succ y
f x
```

```
[ERROR] Type mismatch in application:
1.4: f x
  string literal of type "oops" does not match type `int`
1.0: x = "oops."
  but it flows into reference with expected type `int`
1.4: f x
  Note: constraint arises from reference:
1.2: 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