Rich types, Tractable typing – Overview –

Yann Régis-Gianas
yrg@irif.fr

2017-12-08

What have you learnt for now?

Yann Régis-Gianas
yrg@irif.fr

What have you learnt for now?

Well typed programs never go wrong!

Yann Régis-Gianas
yrg@irif.fr

What have you learnt for now?

Yann Régis-Gianas
yrg@irif.fr

Well typed programs never go wrong!

A precise account on key concepts behind this slogan

- Typed functional programming
- Operational semantics
- Type soundness

The slogan is a bit strong, isn't it?

```
1 List.map2 ( / ) l1 l2
```

Type safety, more precisely

Type safety is **a deal between the type system and the runtime**: to be safe, the well-formedness of each term must be checked.
There are **implicit assumptions** about which of these checks are the responsability of the typechecker and which part is the responsability of the runtime system.
Of course, we prefer static checks: they improve both the robustness and the performance of our programs! But how far can we go? Can we totally remove the aforementionned implicit assumptions?

Yann Régis-Gianas
yrg@irif.fr

Richer types, richer guarantees

```
1 List.map2 ( / ) l1 l2
```

Is there a type for that (guarantee)?

- `l1` and `l2` the same length.
- `l2` only contains strictly positive integers.
- This is a terminating computation.
- `List.map2` enjoys a linear complexity.
- `map2` is productive [1].
- This is a pure computation.
- ...

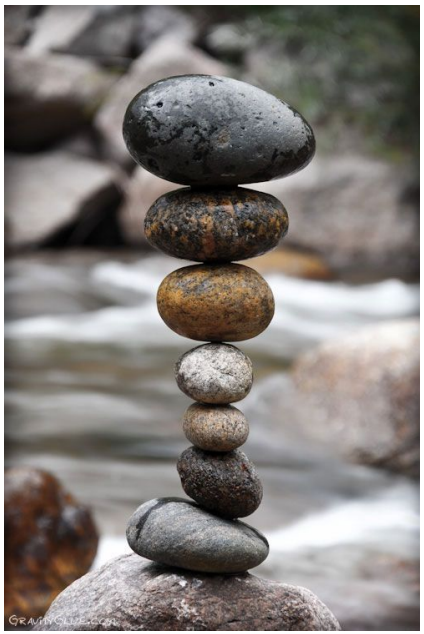
[1] Even if `l1` and `l2` are infinite, any finite prefix of the result can be observed.

Yann Régis-Gianas
yrg@irif.fr

Richer types, complex trade-offs

```
1  (* f : 'a list -> 'b list -> ('a * 'b) list *)
2  let f l1 l2 = List.map2 mkPair l1 l2
```

Type systems designers are like rock balancing artists



Principality
Efficient type checking
Usefulness
Decidability
Soundness
Expressivity

and, according to them, ML sits at a sweet spot in the design space.

Yann Régis-Gianas
yrg@irif.fr

Expressivity vs Tractability

Yann Régis-Gianas
yrg@irif.fr

As a programming tool, a type system must not only be expressive and sound but it must also fulfill some (maybe more informal) practical properties: typechecking must be efficient, types must be palatable, type inference should be predictable, types should be erasable, etc.

Motto

> Make type systems as rich as possible
> while keeping an eye on their implementation.

Something to chew on

Yann Régis-Gianas
yrg@irif.fr

More technically, we focus on type systems equipped with rules like:

$$\frac{\Gamma \vdash t : T_1 \qquad \Gamma \vdash T_1 \mathrel{\mathcal{R}} T_2}{\Gamma \vdash t : T_2}$$

where $\mathcal{R}$ is typically some equivalence or partial order over types.

This part of the course

- Act 1: ML and type inference.
- Act 2: Subtyping.
- Act 3: Modules.
- Act 4: Infinite computations.
- Act 5: Effects and resources.
- Act 6: Dependently-typed systems for programming.
- Act 7: Functional correctness.

Yann Régis-Gianas
yrg@irif.fr