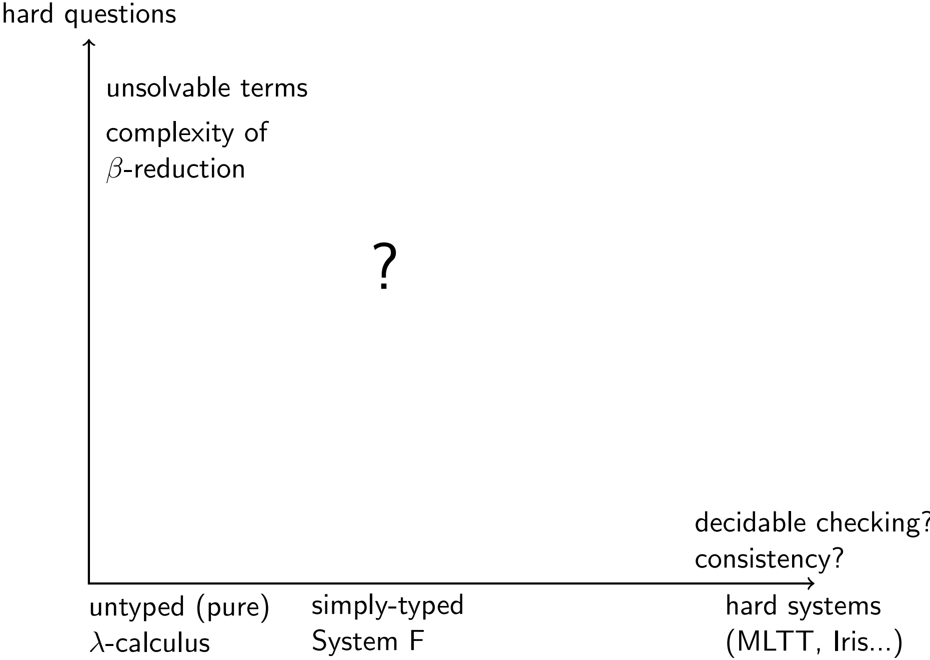
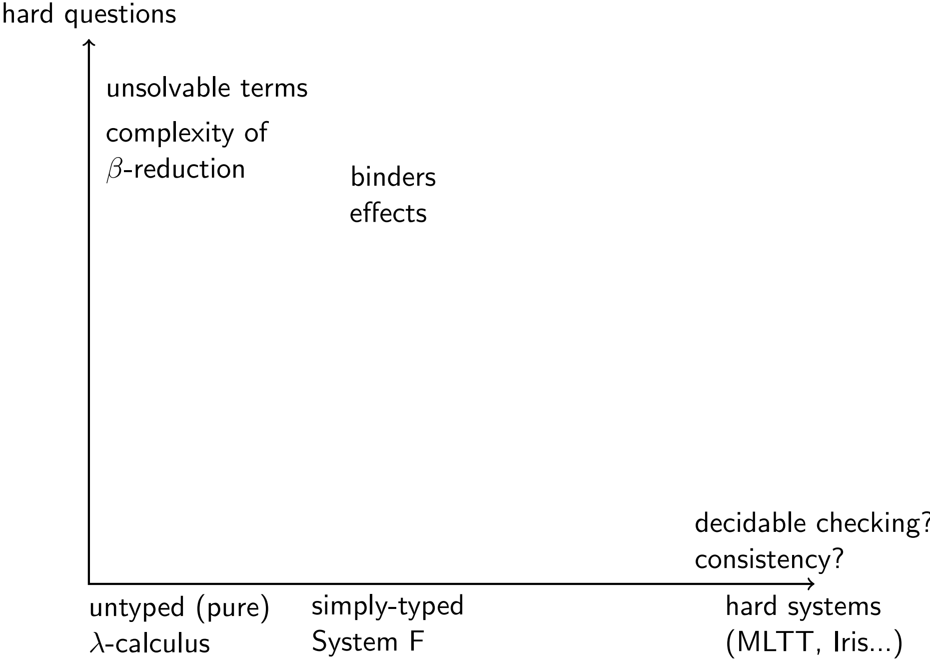
Balance in programming research

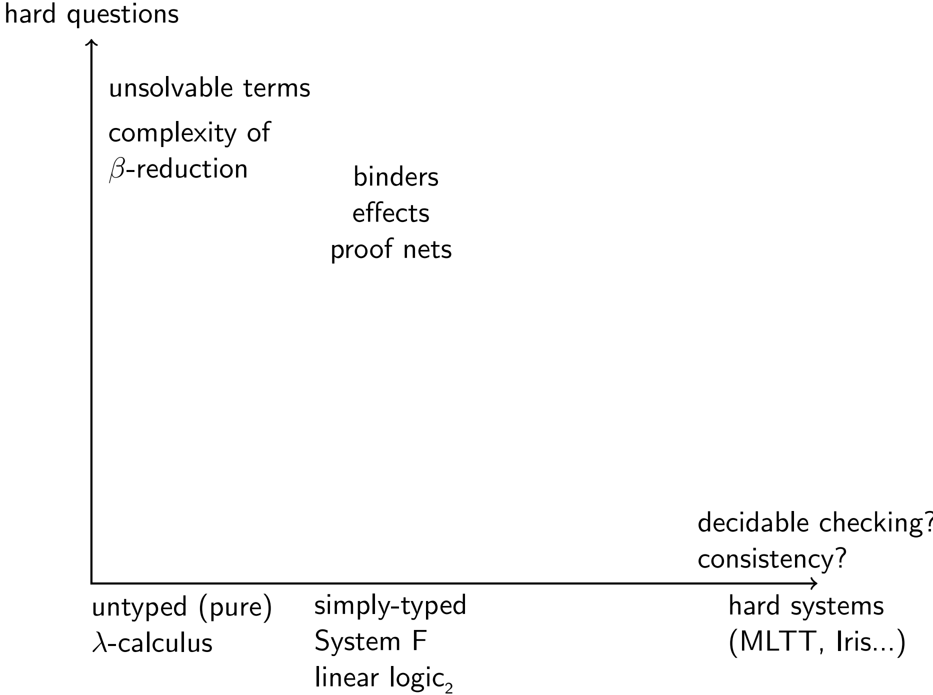
Gabriel Scherer

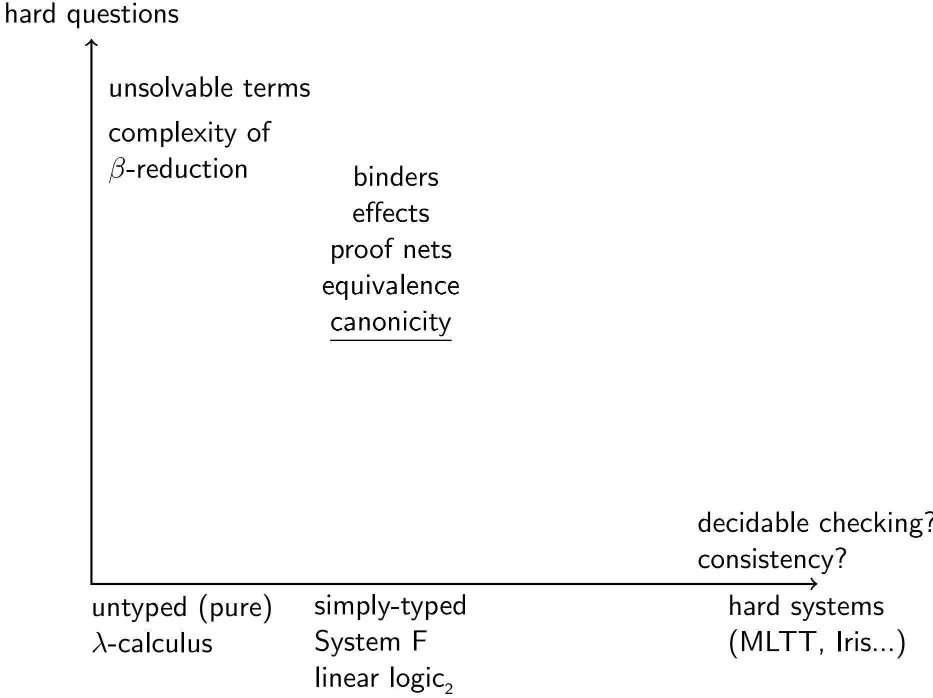
Parsifal, INRIA Saclay

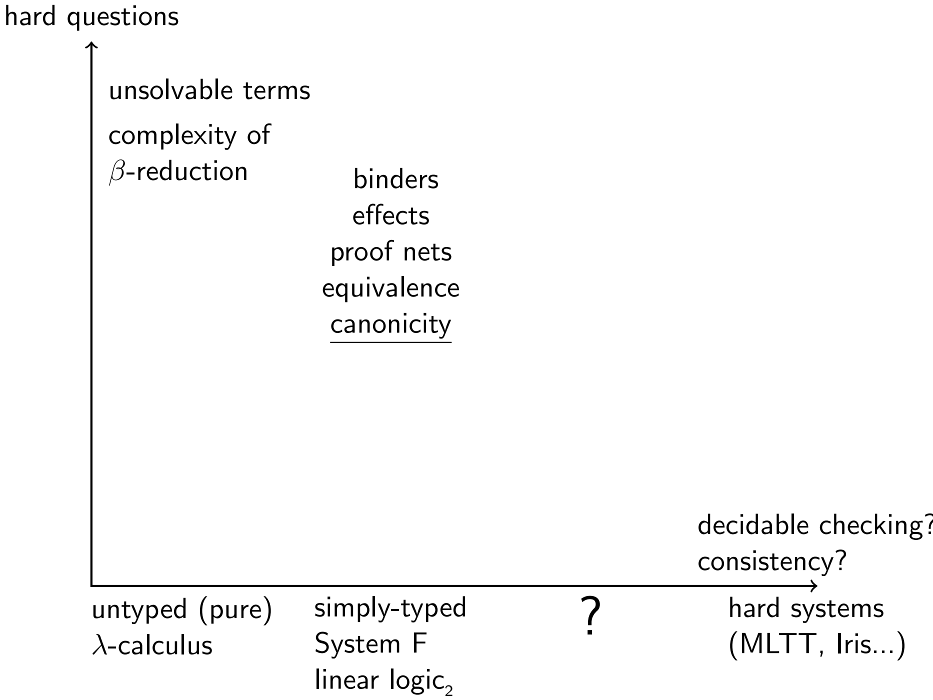
March 21, 2019

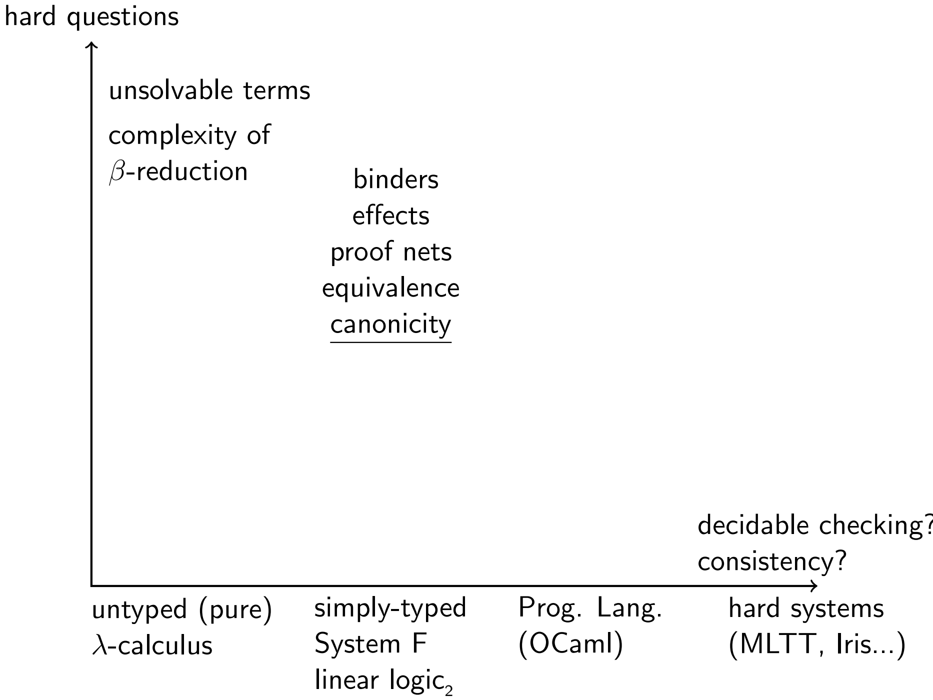

hard questions
unsolvable terms
complexity of
β-reduction
?
decidable checking?
consistency?
untyped (pure)
λ-calculus
simply-typed
System F
hard systems
(MLTT, Iris...)

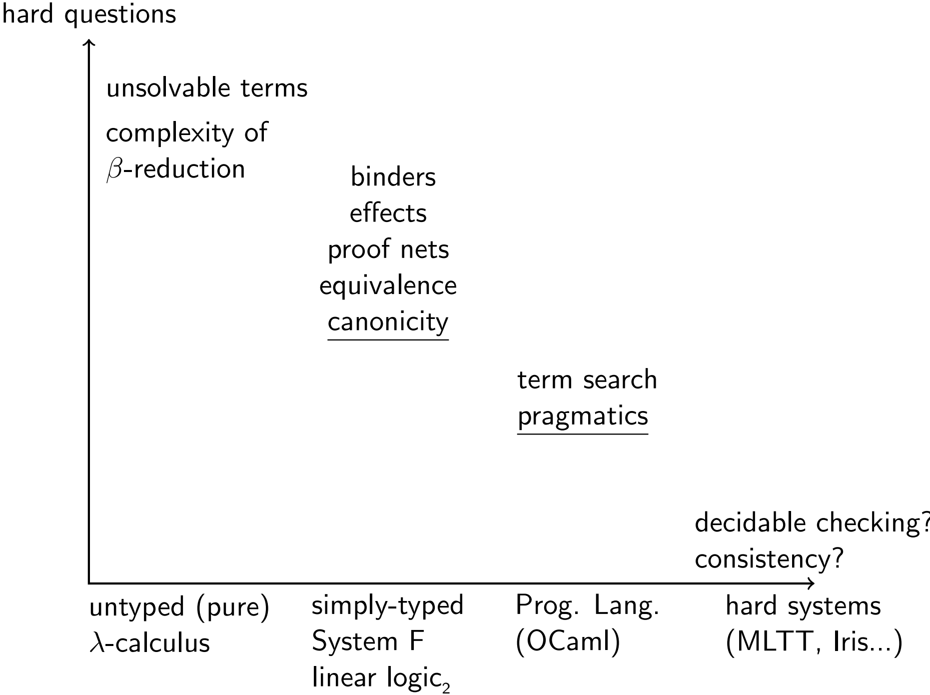

hard questions
unsolvable terms
complexity of
β-reduction
binders
effects
decidable checking?
consistency?
untyped (pure)
λ-calculus
simply-typed
System F
hard systems
(MLTT, Iris...)


hard questions
unsolvable terms
complexity of
β-reduction
binders
effects
proof nets
decidable checking?
consistency?
untyped (pure)
λ-calculus
simply-typed
System F
linear logic[2]
hard systems
(MLTT, Iris...)


hard questions
unsolvable terms
complexity of
β-reduction
binders
effects
proof nets
equivalence
canonicity
decidable checking?
consistency?
untyped (pure)
λ-calculus
simply-typed
System F
linear logic$_2$
hard systems
(MLTT, Iris...)


hard questions
unsolvable terms
complexity of
β-reduction
binders
effects
proof nets
equivalence
canonicity
decidable checking?
consistency?
untyped (pure)
λ-calculus
simply-typed
System F
linear logic$_2$
?
hard systems
(MLTT, Iris...)


hard questions
unsolvable terms
complexity of
β-reduction
binders
effects
proof nets
equivalence
canonicity
decidable checking?
consistency?
untyped (pure)
λ-calculus
simply-typed
System F
linear logic$_2$
Prog. Lang.
(OCaml)
hard systems
(MLTT, Iris...)


hard questions
unsolvable terms
complexity of
β-reduction
binders
effects
proof nets
equivalence
canonicity
term search
pragmatics
decidable checking?
consistency?
untyped (pure)
λ-calculus
simply-typed
System F
linear logic[2]
Prog. Lang.
(OCaml)
hard systems
(MLTT, Iris...)

OCaml

Strongly typed functional language – ML family.
Widely used in our research communities, niche outside.

Research successes: Coq, Why3, Frama-C, HOL-light, CIL, slam/sdv, $F^\star$...

Industrial successes: languages (Rust, Webassembly), finance (at Jane Street), program analysis (at Facebook), blockchain (Tezos), unikernels (at Docker).

Important common infrastructure.

Free Software project, maintained by a distributed group of 17 volunteers. (France, UK, Japan)
I'm one of the most active maintainers.

OCaml *research*

Active project: more applied research for OCaml.
(Inspiration: what SPJ does beautifully for Haskell)

Last year:

- internship: safely unboxing mutually-recursive declarations
- internship: a type system for recursive value declarations
- collaboration: a paper on Merlin (ICFP Experience Report)

Focus: recursive value declarations

```
let rec x(t) = x(t)
let rec x = 1 + x
let rec x = 1 :: x
```

Focus: recursive value declarations

```
let rec x(t) = x(t)          fun t -> (x:Delay)(t)
let rec x = 1 + x            1 :: (x : Guard)
let rec x = 1 :: x           1 + (x : Dereference)
```

Focus: recursive value declarations

```
let rec x(t) = x(t)          fun t -> (x:Delay)(t)
let rec x = 1 + x            1 :: (x : Guard)
let rec x = 1 :: x           1 + (x : Dereference)
```

$m ::= \text{Ignore} \mid \text{Delay} \mid \text{Guard} \mid \text{Return} \mid \text{Dereference} \qquad \Gamma ::= (x \mapsto m)^*$

$$\Gamma \vdash t : m$$

How to check a declaration?

`let rec` x_1 `=` e_1 `... and` x_n `=` e_n `in body`

Focus: recursive value declarations

```
let rec x(t) = x(t)          fun t -> (x:Delay)(t)
let rec x = 1 + x            1 :: (x : Guard)
let rec x = 1 :: x           1 + (x : Dereference)
```

$m ::= \mathsf{Ignore} \mid \mathsf{Delay} \mid \mathsf{Guard} \mid \mathsf{Return} \mid \mathsf{Dereference} \qquad \Gamma ::= (x \mapsto m)^*$

$$\Gamma \vdash t : m$$

How to check a declaration?

`let rec` x_1 `=` e_1 `... and` x_n `=` e_n `in body`

$$? \vdash e_i : \mathsf{Return}$$

Focus: recursive value declarations

```
let rec x(t) = x(t)          fun t -> (x:Delay)(t)
let rec x = 1 + x            1 :: (x : Guard)
let rec x = 1 :: x           1 + (x : Dereference)
```

$m ::= \text{Ignore} \mid \text{Delay} \mid \text{Guard} \mid \text{Return} \mid \text{Dereference} \qquad \Gamma ::= (x \mapsto m)^*$

$$\Gamma \vdash t : m$$

How to check a declaration?

let rec x_1 = e_1 ... and x_n = e_n in body

$$? \vdash e_i : \text{Return}$$

$$\Gamma_i \vdash e_i : \text{Return}$$

Focus: recursive value declarations

```
let rec x(t) = x(t)            fun t -> (x:Delay)(t)
let rec x = 1 + x              1 :: (x : Guard)
let rec x = 1 :: x             1 + (x : Dereference)
```

$m ::= \text{Ignore} \mid \text{Delay} \mid \text{Guard} \mid \text{Return} \mid \text{Dereference} \qquad \Gamma ::= (x \mapsto m)^*$

$$\Gamma \vdash t : m$$

How to check a declaration?

let rec x_1 = e_1 ... and x_n = e_n in body

$$? \vdash e_i : \text{Return}$$

$$\Gamma_i \vdash e_i : \text{Return}$$

$$\forall \Gamma_i, \forall x_j, \quad \Gamma_i(x_j) \leq \text{Guard}$$

Transition slide.

Canonicity

What is the **identity** of programs (λ-terms)?

Canonical representation: a syntactic description of the representatives of the (contextual) equivalence classes.

$$t, u \text{ canonical} \implies t \neq_\alpha u \implies t \neq_{\text{ctx}} u$$

Application: deciding equivalence, program synthesis (maybe?).
Just darn interesting.

Canonicity

What is the **identity** of programs (λ-terms)?

Canonical representation: a syntactic description of the representatives of the (contextual) equivalence classes.

$$t, u \text{ canonical} \implies t \neq_\alpha u \implies t \neq_{\mathsf{ctx}} u$$

Application: deciding equivalence, program synthesis (maybe?).
Just darn interesting.

> $\Lambda C(\alpha, \to, \times)$: β-short η-long normal forms.
> $\Lambda C(\alpha, \to, \times, +)$: ...
> $\Lambda C(\alpha, \to, \times, 1, +, 0)$: ?

Solution proposed in 2017 using (maximal multi-)**focusing**.

Canonicity

What is the **identity** of programs (λ-terms)?

Canonical representation: a syntactic description of the representatives of the (contextual) equivalence classes.

$$t, u \text{ canonical} \implies t \neq_\alpha u \implies t \neq_{\text{ctx}} u$$

Application: deciding equivalence, program synthesis (maybe?).
Just darn interesting.

> $\Lambda C(\alpha, \to, \times)$: β-short η-long normal forms.
> $\Lambda C(\alpha, \to, \times, +)$: ...
> $\Lambda C(\alpha, \to, \times, 1, +, 0)$: ?

Solution proposed in 2017 using (maximal multi-)**focusing**.

Goal: richer types.

Canonicity: future work

System F: no subformula property.

$$\frac{\Gamma, A[B/\alpha] \vdash C}{\Gamma \ni \forall \alpha.\, A \vdash C}$$

Equivalence is undecidable in F: no decidable canonical forms.

Could we have a partial algorithm that works sometimes?

Eliminating polymorphism

Idea: probe the structure of $\forall\alpha.\, A$ through (canonical) proof search.

$$\dfrac{\dfrac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \overset{\text{def}}{=} A \to B \to \alpha \vdash \alpha}}{\vdash \forall\alpha.\,(A \to B \to \alpha) \to \alpha}$$

Eliminating polymorphism

Idea: probe the structure of $\forall\alpha.\,A$ through (canonical) proof search.

$$\dfrac{\dfrac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \stackrel{\text{def}}{=} A \to B \to \alpha \vdash \alpha}}{\vdash \forall\alpha.\,(A \to B \to \alpha) \to \alpha} \qquad \dfrac{\dfrac{\Gamma \vdash A \quad \oplus \quad \Gamma \vdash B}{\Gamma \stackrel{\text{def}}{=} A \to \alpha, B \to \alpha \vdash \alpha}}{\vdash \forall\alpha.\,(A \to \alpha) \to (B \to \alpha) \to \alpha}$$

Eliminating polymorphism

Idea: probe the structure of $\forall\alpha.\, A$ through (canonical) proof search.

$$\dfrac{\dfrac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \overset{\text{def}}{=} A \to B \to \alpha \vdash \alpha}}{\vdash \forall\alpha.\,(A \to B \to \alpha) \to \alpha}$$

$$\dfrac{\dfrac{\Gamma \vdash A \quad \oplus \quad \Gamma \vdash B}{\Gamma \overset{\text{def}}{=} A \to \alpha, B \to \alpha \vdash \alpha}}{\vdash \forall\alpha.\,(A \to \alpha) \to (B \to \alpha) \to \alpha}$$

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \alpha} \quad \oplus \quad \dfrac{\dfrac{}{\Gamma \vdash \alpha \to \alpha} \qquad \Gamma \vdash \alpha}{\Gamma \vdash \alpha}}{\Gamma \overset{\text{def}}{=} \alpha \to \alpha, \alpha \vdash \alpha}}{\vdash \forall\alpha.\,(\alpha \to \alpha) \to \alpha \to \alpha}$$

Eliminating polymorphism

Idea: probe the structure of $\forall\alpha.\, A$ through (canonical) proof search.

$$\dfrac{\dfrac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \stackrel{\text{def}}{=} A \to B \to \alpha \vdash \alpha}}{\vdash \forall\alpha.\,(A \to B \to \alpha) \to \alpha}
\qquad
\dfrac{\dfrac{\Gamma \vdash A \quad \oplus \quad \Gamma \vdash B}{\Gamma \stackrel{\text{def}}{=} A \to \alpha, B \to \alpha \vdash \alpha}}{\vdash \forall\alpha.\,(A \to \alpha) \to (B \to \alpha) \to \alpha}$$

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \alpha} \quad \oplus \quad \dfrac{\dfrac{}{\Gamma \vdash \alpha \to \alpha} \qquad \Gamma \vdash \alpha}{\Gamma \vdash \alpha}}{\Gamma \stackrel{\text{def}}{=} \alpha \to \alpha, \alpha \vdash \alpha}}{\vdash \forall\alpha.\,(\alpha \to \alpha) \to \alpha \to \alpha}$$

On which fragments can this idea work?

Zooming out

Goal: balance between applied and theoretical research.

Zooming out

Goal: balance between applied and theoretical research.