Research vision

Danel Ahman

University of Ljubljana

Balliol College, 12.11.2018

Programming Languages

```
let r = alloc 0 in r := !r + 1; r
```

Programming Languages

```
let r = alloc 0 in r := !r + 1; r
```

Typed Programming Languages

```
let r = alloc 0 in r := !r + 1; r : ref Nat
```

lightweight and modular

specification

verification

documentation

correct by construction

Typed Programming Languages

```
let r = alloc 0 in r := !r + 1; r : ref Nat
```

lightweight and modular

specification
verification
documentation

correct by construction

Typed Programming Languages

But what about behaviour?

```
let r = alloc 0 in r := !r + 1; r : ref Nat
```

- r is fresh
- !r > 0
- other effects like I/O ?
- user-defined effects ?
- ...

State of affairs in type-based reasoning

State of affairs in type-based reasoning

Values

Well-understood, uniform,

and thoroughly studied! :)

◉ Refinement types

```
Odd ⊑ Nat     Even ⊑ Nat

Vec A n = { l:List A | len l = n }
```

◉ Dependent types

```
Vec a n = Σ l:List A.(len l = n)
```

◉ Homogeneous implementations

- Agda, Coq, Idris, F*, L.Haskell, ...

State of affairs in type-based reasoning

Values

Well-understood, uniform, and thoroughly studied! :)

- **Refinement types**

```
Odd ⊑ Nat     Even ⊑ Nat

Vec A n = { l:List A | len l = n }
```

- **Dependent types**

```
Vec a n = Σ l:List A.(len l = n)
```

- **Homogeneous implementations**

  - Agda, Coq, Idris, F*, L.Haskell, ...

Effects and behaviour

Scattered landscape, effect-specific, little uniformity! :(

- **Hoare Type Theory** (state)

```
M : Ψ.X.{P}x:A{Q}
```

- **F*** (state, exceptions, but no I/O)

```
M : ST A wpST
```

~~`M : IO A wpIO`~~

- **Session Types** (I/O & channels)

```
c : ?Nat.!String.!Nat.T
```

- **Graded monads, param. monads**

My vision: no need for this non-uniformity!

My vision: no need for this non-uniformity!

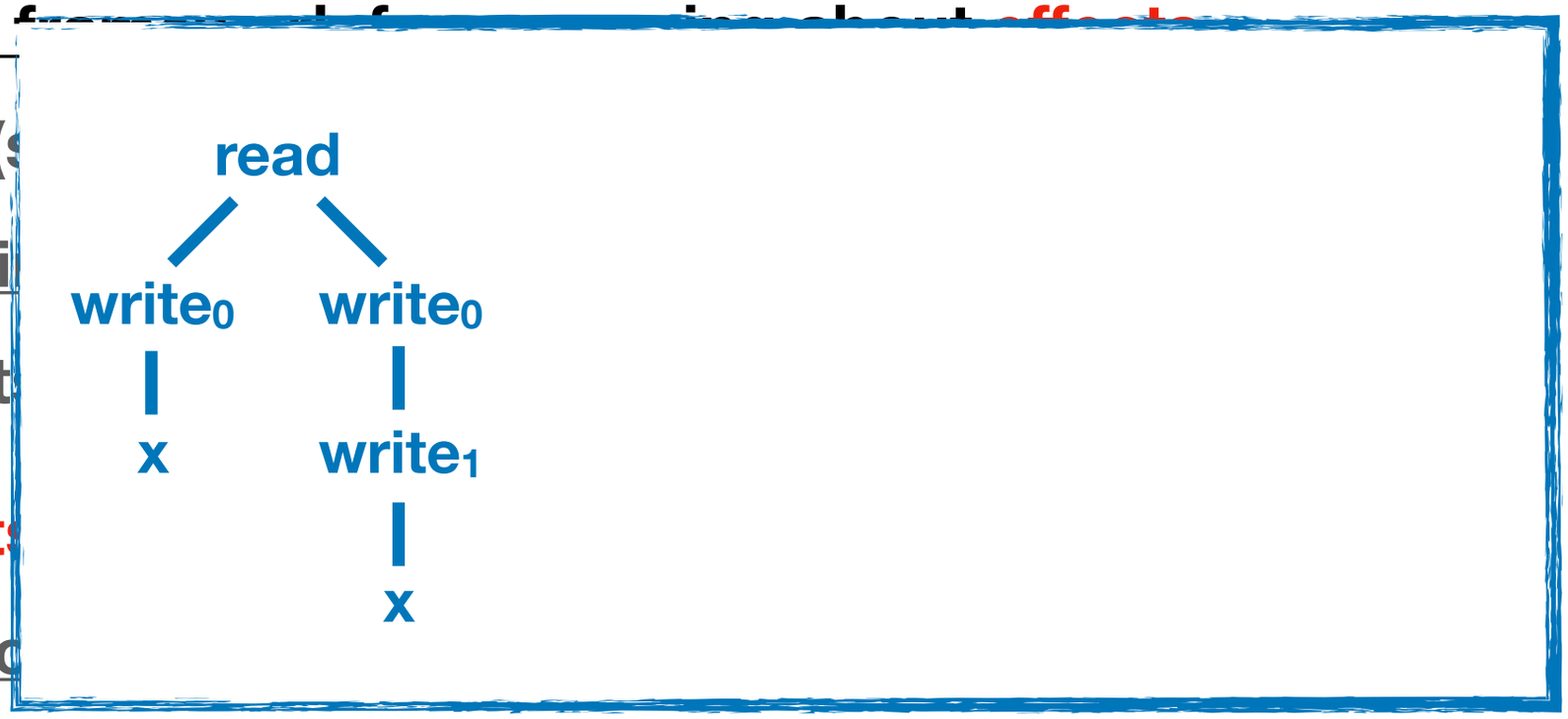
- **Goal:** a general, uniform framework for reasoning about effects
  - wide range of effects (state, I/O, exceptions, probability, ...)
  - primitive and user-defined effects
  - combinations of effects

My vision: no need for this non-uniformity!

- **Goal:** a general, uniform framework for reasoning about effects
  - wide range of effects (state, I/O, exceptions, probability, ...)
  - primitive and user-defined effects
  - combinations of effects
- **Answer:** algebraic effects and effect handlers (rather than just monads)
  - operations and equations
  - reveal the fundamental underlying tree-like structure of effects
  - effect handlers are algebras; handling is homomorphism application

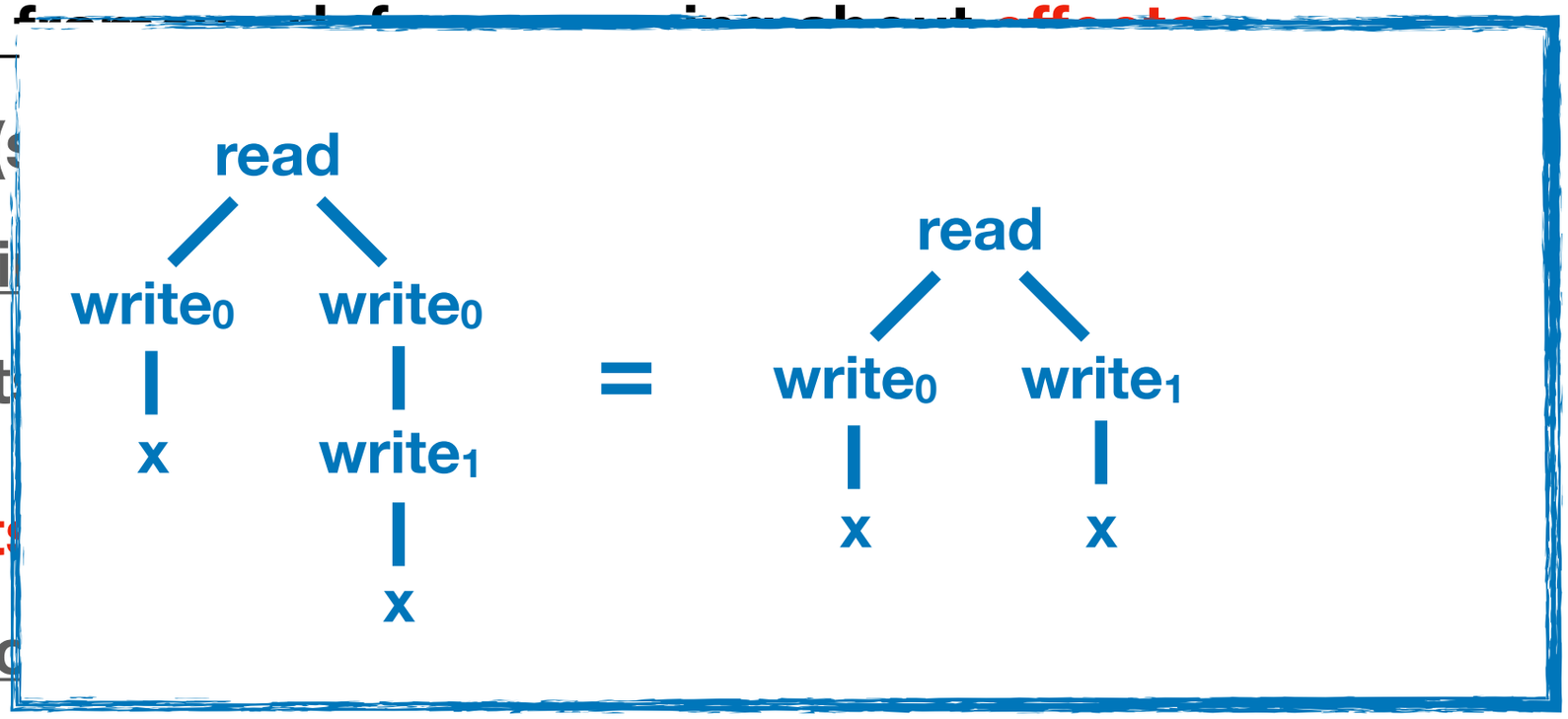
My vision: no need for this non-uniformity!

- **Goal:** a general, uniform f… …about effects…
  - wide range of effects (s…
  - primitive and user-defi…
  - combinations of effect…
- **Answer:** algebraic effects…
  - operations and equatio…
  - reveal the **fundamental underlying tree-like structure** of effects
  - **effect handlers** are algebras; **handling** is homomorphism application



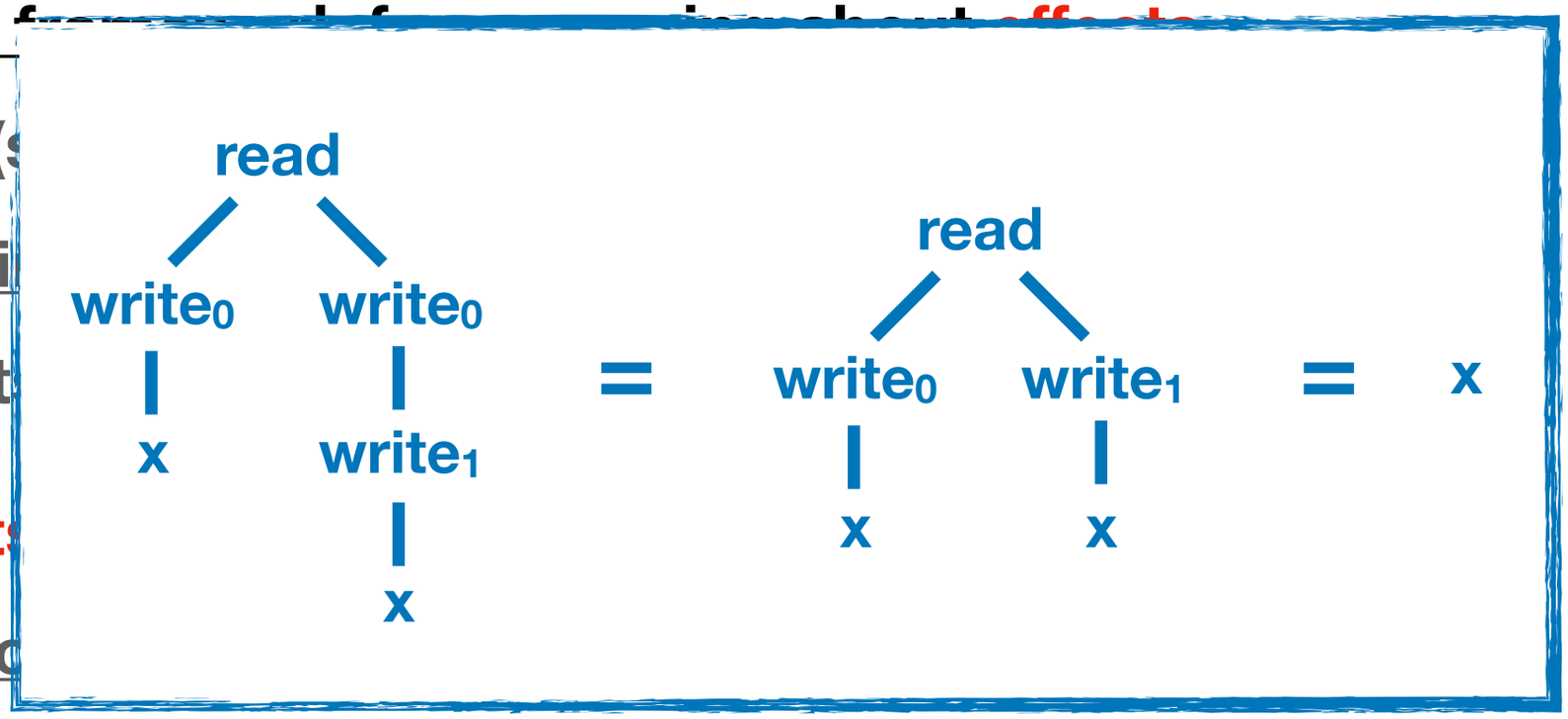
My vision: no need for this non-uniformity!

- **Goal:** a general, uniform f… …about effects…
  - wide range of effects (s…
  - primitive and user-defi…
  - combinations of effect…
- **Answer:** algebraic effects…
  - operations and equatio…
  - reveal the **fundamental underlying tree-like structure** of effects
  - **effect handlers** are algebras; **handling** is homomorphism application



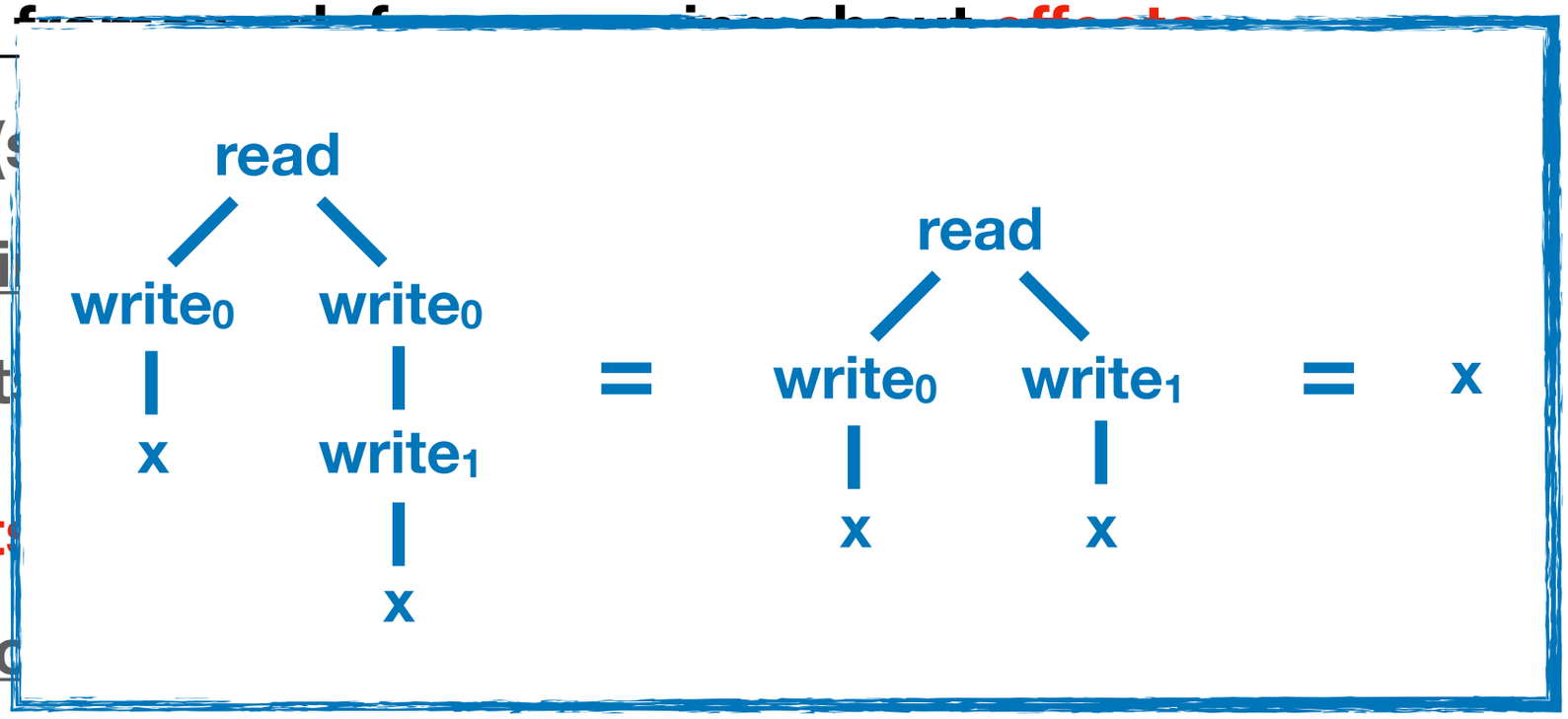
My vision: no need for this non-uniformity!

- **Goal:** a general, uniform f… …about effects…
  - wide range of effects (s…
  - primitive and user-defi…
  - combinations of effect…
- **Answer:** algebraic effects…
  - operations and equatio…
  - reveal the **fundamental underlying tree-like structure** of effects
  - **effect handlers** are algebras; **handling** is homomorphism application



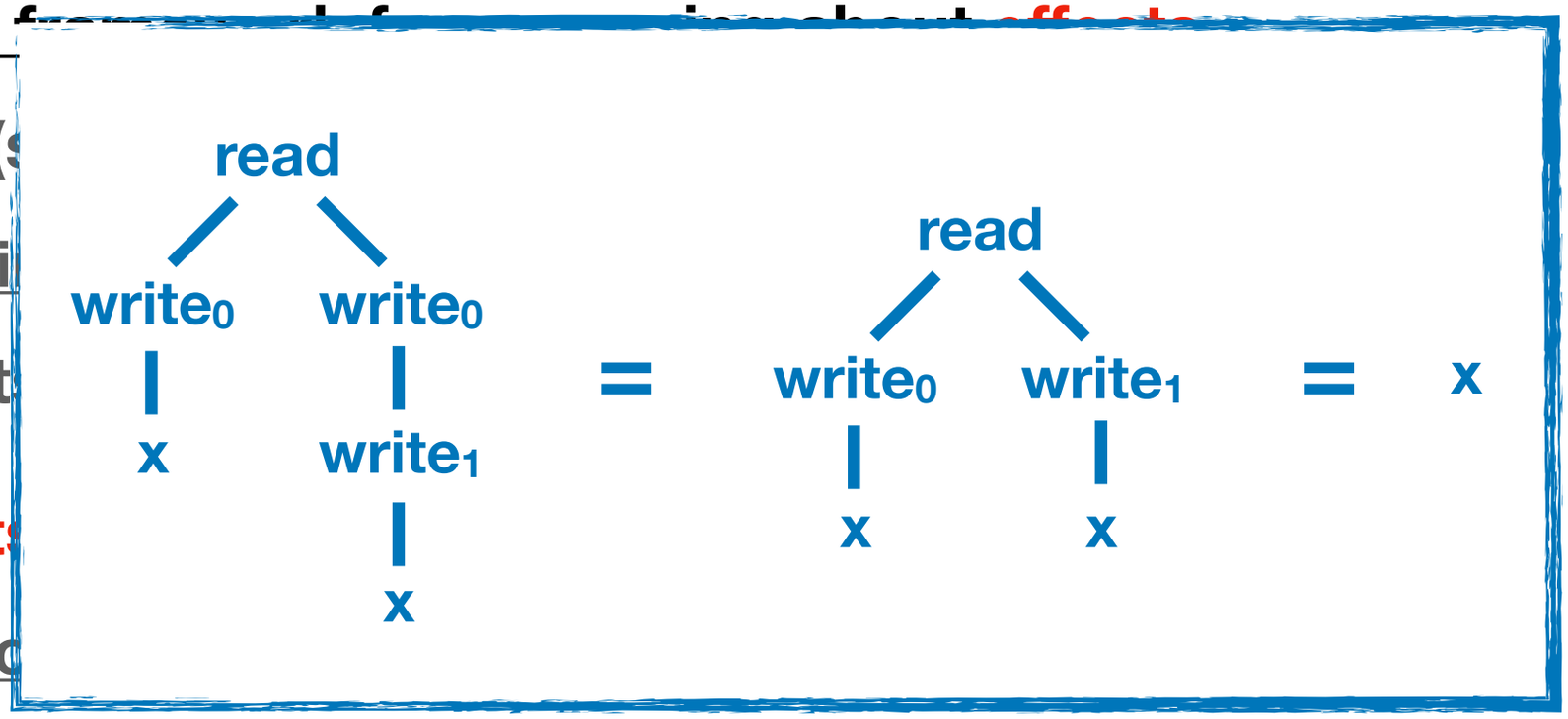
My vision: no need for this non-uniformity!

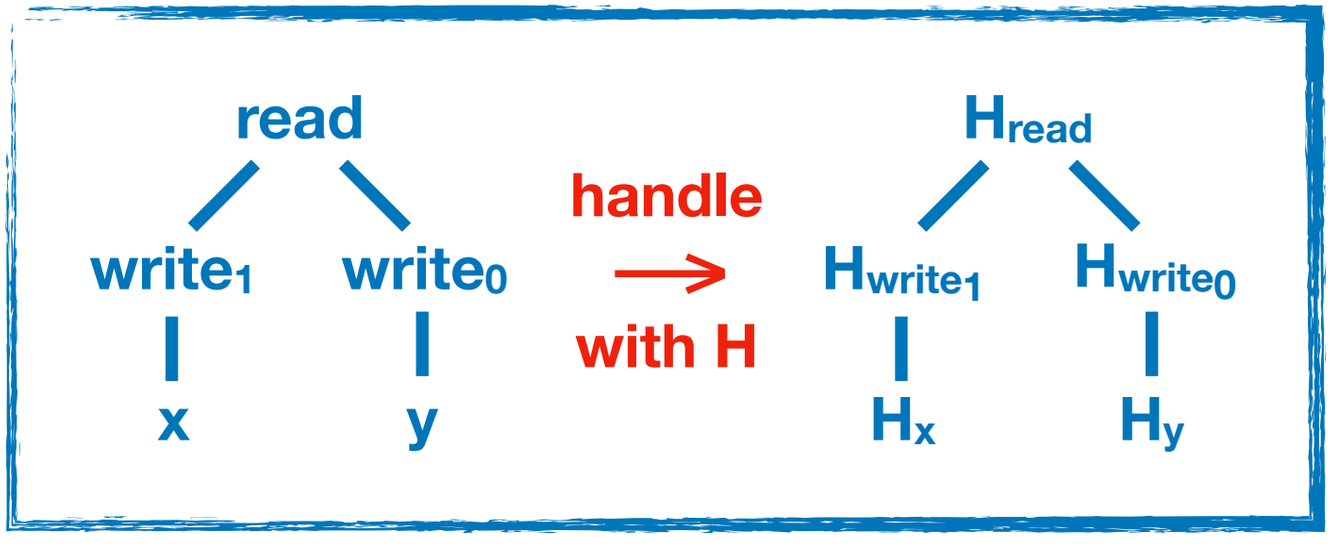
- **Goal:** a general, uniform f… …about effects…
  - wide range of effects (s…
  - primitive and user-defi…
  - combinations of effect…
- **Answer:** algebraic effect…
  - operations and equatio…
  - reveal the **fundamental underlying tree-like structure** of effects
  - **effect handlers** are algebras; **handling** is homomorphism application



My vision: no need for this non-uniformity!

- **Goal:** a general, uniform f… …about effects…
  - wide range of effects (s…
  - primitive and user-defi…
  - combinations of effect…
- **Answer:** algebraic effects…
  - operations and equatio…
  - reveal the **fundamental underlying tree-like structure** of effects
  - **effect handlers** are algebras; **handling** is homomorphism application





My vision: no need for this non-uniformity!

- **Goal:** a general, uniform framework for reasoning about effects
  - wide range of effects (state, I/O, exceptions, probability, ...)
  - primitive and user-defined effects
  - combinations of effects
- **Answer:** algebraic effects and effect handlers (rather than just monads)
  - operations and equations
  - reveal the fundamental underlying tree-like structure of effects
  - effect handlers are algebras; handling is homomorphism application

My vision: no need for this non-uniformity!

- **Goal:** a general, uniform framework for reasoning about effects
  - wide range of effects (state, I/O, exceptions, probability, ...)
  - primitive and user-defined effects
  - combinations of effects
- **Answer:** algebraic effects and effect handlers (rather than just monads)
  - operations and equations
  - reveal the fundamental underlying tree-like structure of effects
  - effect handlers are algebras; handling is homomorphism application
- **State of the art:** very popular (!) but effect systems too coarse grained (!)
  - concurrency, probability, delimited control, monadic reflection, ...
  - Multicore OCaml, Uber's Pyro tool, Eff, Koka, Frank, ...
  - `M : A ! { read , write, throw }`

The plan: modal logic based c. ref. types

The plan: modal logic based c. ref. types

- **Simple idea: exploit the underlying tree-like structure of effects!**
  - $\langle op \rangle(\psi_1, \ldots, \psi_n)$ for each n-ary operation symbol (cf TYPES'15)

The plan: modal logic based c. ref. types

- **Simple idea: exploit the underlying tree-like structure of effects!**
  - $\langle op \rangle(\psi_1, \ldots, \psi_n)$ for each n-ary operation symbol (cf TYPES'15)

$$\langle op \rangle(\psi_1, \ldots, \psi_n) = \left\{ \begin{array}{ccccc} & & op & & \\ & \swarrow & & \searrow & \\ t_1 & & \cdots & & t_n \end{array} \;\middle|\; t_1 \in \psi_1 \wedge \cdots \wedge t_n \in \psi_n \right\}$$

$$M : A \,!\, \psi$$

The plan: modal logic based c. ref. types

- **Simple idea: exploit the underlying tree-like structure of effects!**
  - $\langle op \rangle(\psi_1, \ldots, \psi_n)$ for each n-ary operation symbol (cf TYPES'15)

The plan: modal logic based c. ref. types

- **Simple idea: exploit the underlying tree-like structure of effects!**
  - $\langle op \rangle(\psi_1, \ldots, \psi_n)$ for each n-ary operation symbol (cf TYPES'15)
- **Major pros:**
  - uniform across all algebraic effects
  - can already encode **Hoare Logic**, Session Types, HL $\otimes$ ST, ...

The plan: modal logic based c. ref. types

- **Simple idea: exploit the underlying tree-like structure of effects!**
  - $\langle op \rangle(\psi_1, \ldots, \psi_n)$ for each n-ary operation symbol (cf TYPES'15)
- **Major pros:**
  - uniform across all algebraic effects
  - can already encode **Hoare Logic**, Session Types, HL $\otimes$ ST, ...

$$\{\{1\}\}\ A\ \{Q\} = A\ !\ \bigvee_{i\in\{0,1\},q\in Q} \langle rd\rangle(\langle wr_i\rangle(ret), \langle wr_q\rangle(ret))$$

The plan: modal logic based c. ref. types

- **Simple idea: exploit the underlying tree-like structure of effects!**
  - $\langle op \rangle(\psi_1, \ldots, \psi_n)$ for each n-ary operation symbol (cf TYPES'15)
- **Major pros:**
  - uniform across all algebraic effects
  - can already encode **Hoare Logic**, Session Types, HL $\otimes$ ST, ...

The plan: modal logic based c. ref. types

- **Simple idea: exploit the underlying tree-like structure of effects!**
  - $\langle op \rangle(\psi_1, \ldots, \psi_n)$ for each n-ary operation symbol (cf TYPES'15)
- **Major pros:**
  - uniform across all algebraic effects
  - can already encode Hoare Logic, Session Types, HL ⊗ ST, ...
- **Challenges:**
  - non-linear effect equations (read x x = x)
  - operations with value params. and variable binding
  - effect instances, generativity, and locality (my current focus in LJ)
  - dynamic nature of handlers

Many possible applications

Many possible applications

- **Separation Logic (state, I/O, state ⊗ I/O, ...)**
  - HL + **generative instances** and **locality**

Many possible applications

- **Separation Logic (state, I/O, state ⊗ I/O, ...)**
  - HL + **generative instances** and **locality**
- **Big Data Computations**
  - commutative monoid structure (**an algebraic effect**)
  - partitioning, spatial layout, ...

Many possible applications

- **Separation Logic (state, I/O, state ⊗ I/O, ...)**
  - HL + **generative instances** and **locality**
- **Big Data Computations**
  - commutative monoid structure (**an algebraic effect**)
  - partitioning, spatial layout, ...
- **Concurrency**
  - (multi-)**handlers** based concurrency
  - Scala's `promises` and `futures` as a **monotonic state effect**

Many possible applications

- **Separation Logic (state, I/O, state ⊗ I/O, ...)**
  - HL + **generative instances** and **locality**
- **Big Data Computations**
  - commutative monoid structure (**an algebraic effect**)
  - partitioning, spatial layout, ...
- **Concurrency**
  - (multi-)**handlers** based concurrency
  - Scala's `promises` and `futures` as a **monotonic state effect**
- **Probabilistic programming**
  - `sample` as an **algebraic effect**; `condition` as a **handler** (cf Pyro)

Temporal view

- **Year 1**

  - modal logic (design, model and proof theory)
  - instances, generativity, locality
  - case studies and applications

- **Year 2**

  - declarative PL design (type-and-effect system)
  - meta-theory (denotational and operational)
  - case studies and applications

- **Year 3**

  - algorithmic PL design (type-and-effect inference)
  - implementation
  - case studies and applications

Temporal view

- **Year 1**
  - modal logic (design, model and proof theory)
  - instances, generativity, locality
  - case studies and applications
- **Year 2**
  - declarative PL design (type-and-effect system)
  - meta-theory (denotational and operational)
  - case studies and applications
- **Year 3**
  - algorithmic
  - implementa
  - case studies

- **In parallel**
  - continue collaborations with the F* team
  - continue collaborations on container datatypes
  - forge new collaborations in Oxford (and elsewhere)

Conclusions

- **Software is everywhere!**
- **We had better know what it does!**
- **General and uniform frameworks already exist for values!**
- **But only scattered, effect-specific frameworks for behaviour!**
- **My research will seek to rectify this situation**
  - a uniform and widely applicable approach
  - inspired by algebraic effects and effect handlers
  - both foundational theory and exciting applications