

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

In today's world
software is
everywhere!

Typed Programming Languages

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

lightweight and modular

specification

verification

documentation

correct by construction

In today's world
software is
everywhere!

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

$\text{Odd} \sqsubseteq \text{Nat}$ $\text{Even} \sqsubseteq \text{Nat}$

$\text{Vec } A \ n = \{ \ell : \text{List } A \mid \text{len } \ell = n \}$

- Dependent types

$\text{Vec } a \ h =$

| $\text{nil} : \text{Vec } a \ 0$

| $\text{cons} : \dots \rightarrow \text{Vec } a \ (n+1)$

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

State of affairs in type-based reasoning

Values

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

● Refinement types

$\text{Odd} \sqsubseteq \text{Nat}$ $\text{Even} \sqsubseteq \text{Nat}$

$\text{Vec } A \ n = \{ \ell : \text{List } A \mid \text{len } \ell = n \}$

● Dependent types

$\text{Vec } a \ h =$

| $\text{nil} : \text{Vec } a \ 0$

| $\text{cons} : \dots \rightarrow \text{Vec } a \ (n+1)$

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

Effects and behaviour

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

● Hoare Type Theory (state)

$M : \Psi.X.\{P\}x:A\{Q\}$

● F* (state, exceptions, but no I/O)

$M : \text{ST } A \ \text{wp}_{\text{ST}}$

~~$M : \text{IO } A \ \text{wp}_{\text{IO}}$~~

● Session Types (I/O & channels)

$c : ?\text{Nat}.\text{!String}.\text{!Nat}.T$

● 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

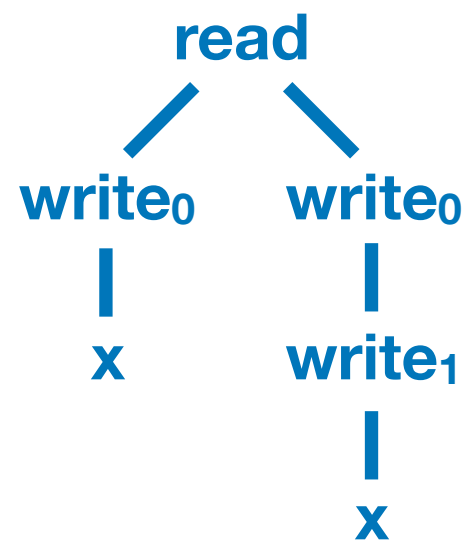
My vision: **no need for this non-uniformity!**

- © **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 homomorphic tree transformers

My vision: no need for this non-uniformity!

- Goal: a general, uniform framework for reasoning about effects

- wide range of effects (stateful computations)
- primitive and user-defined effects
- combinations of effects



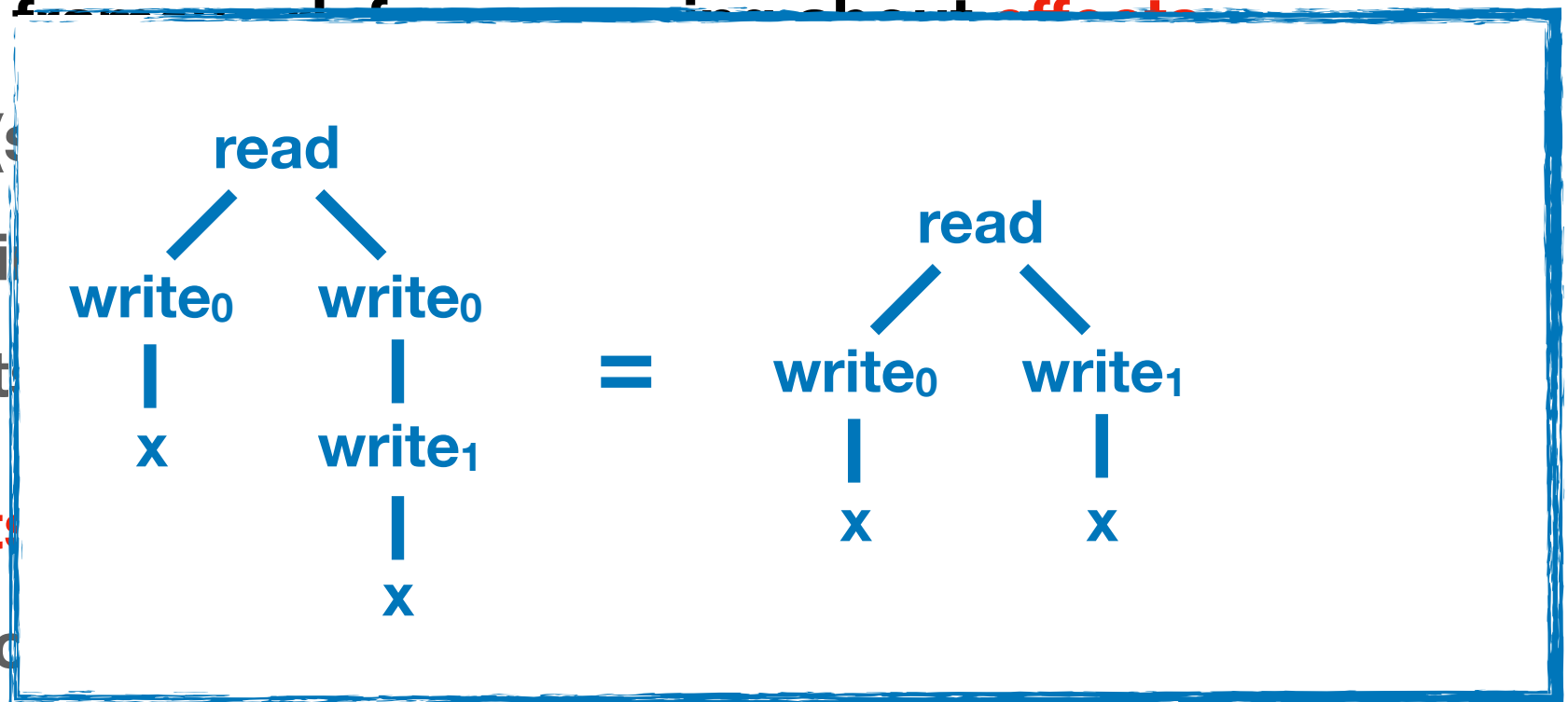
- Answer: algebraic effects

- operations and equations
- reveal the fundamental underlying tree-like structure of effects
- effect handlers are homomorphic tree transformers

My vision: no need for this non-uniformity!

- Goal: a general, uniform framework for reasoning about effects

- wide range of effects (e.g.,
- primitive and user-defined
- combinations of effects



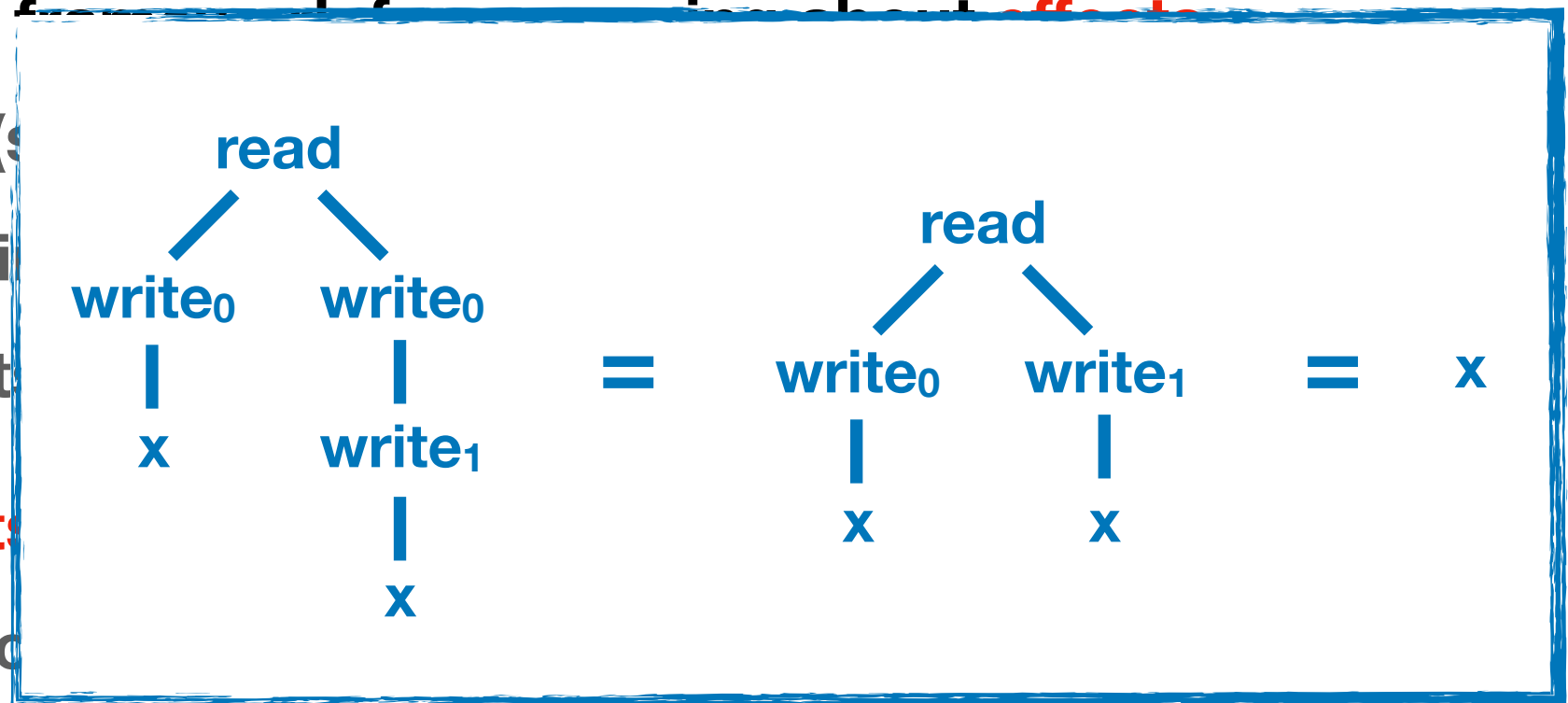
- Answer: algebraic effects

- operations and equations
- reveal the fundamental underlying tree-like structure of effects
- effect handlers are homomorphic tree transformers

My vision: no need for this non-uniformity!

- Goal: a general, uniform framework for reasoning about effects

- wide range of effects (stateful, non-terminating, etc.)
- primitive and user-defined effects
- combinations of effects



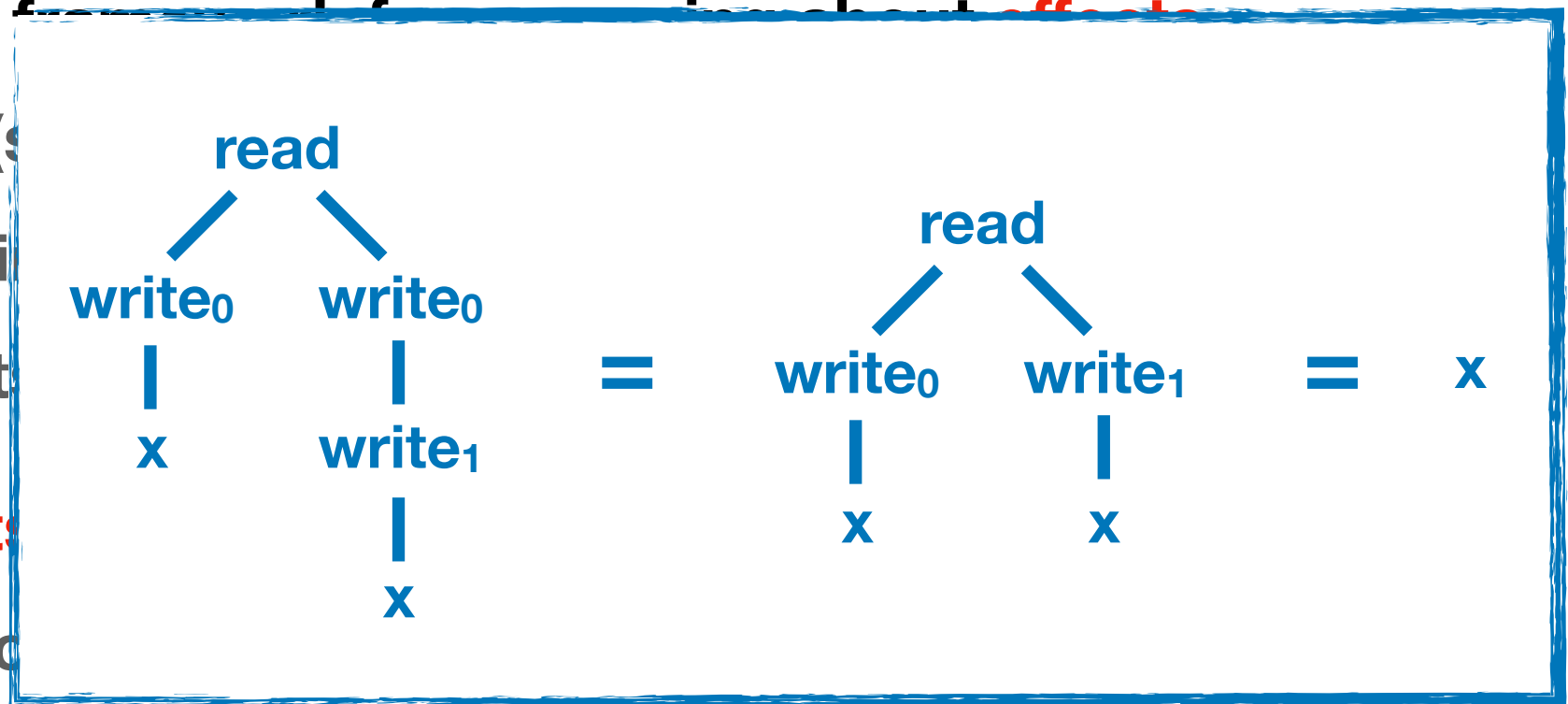
- Answer: algebraic effects

- operations and equations
- reveal the fundamental underlying tree-like structure of effects
- effect handlers are homomorphic tree transformers

My vision: no need for this non-uniformity!

- Goal: a general, uniform framework for reasoning about effects

- wide range of effects (stateful, non-terminating, etc.)
- primitive and user-defined effects
- combinations of effects



- Answer: algebraic effects

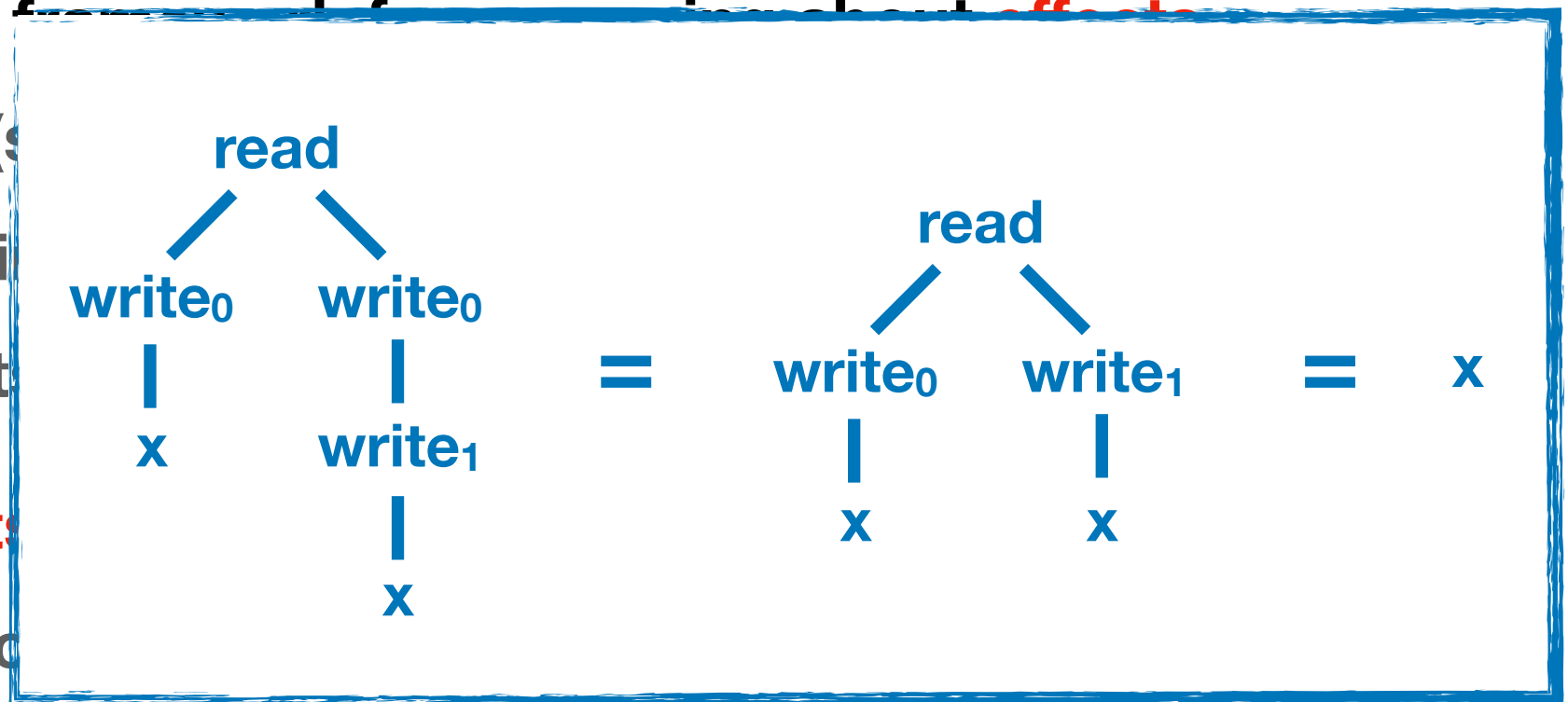
- operations and equations
- reveal the fundamental underlying tree-like structure of effects
- effect handlers are homomorphic tree transformers



My vision: no need for this non-uniformity!

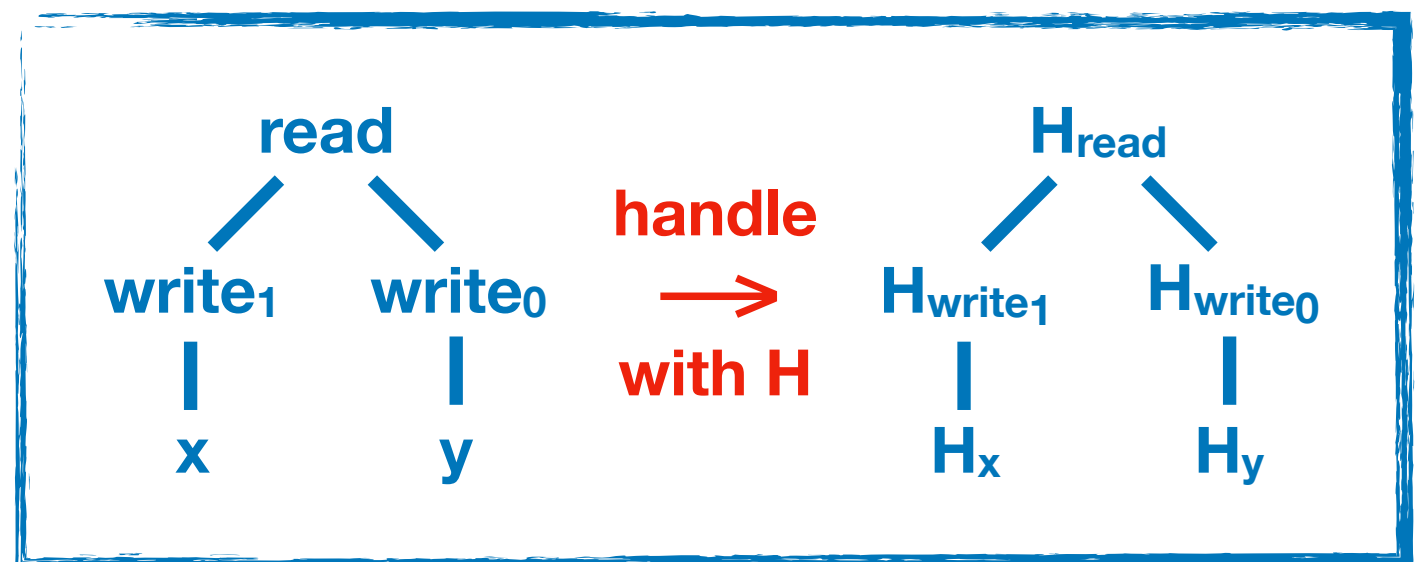
Goal: a general, uniform framework for reasoning about effects

- wide range of effects (stateful, non-terminating, etc.)
- primitive and user-defined effects
- combinations of effects



Answer: algebraic effects

- operations and equations
- reveal the fundamental underlying tree-like structure of effects
- **effect handlers** are homomorphic tree transformers



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 homomorphic tree transformers

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 homomorphic tree transformers
- ◎ **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, ...
 - $M : A ! \{ \text{read}, \text{write}, \text{throw} \}$

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**!
 - $\langle \text{op} \rangle(\psi_1, \dots, \psi_n)$ 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!

- $\langle \text{op} \rangle(\psi_1, \dots, \psi_n)$ for each n-ary operation symbol (cf TYPES'15)

$$\langle \text{op} \rangle(\psi_1, \dots, \psi_n) = \left\{ \begin{array}{c} \text{op} \\ \swarrow \quad \searrow \\ t_1 \quad \dots \quad t_n \end{array} \mid t_1 \in \psi_1 \wedge \dots \wedge t_n \in \psi_n \right\}$$

$M : A ! \psi$

The plan: modal logic based c. ref. types

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

The plan: modal logic based c. ref. types

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

$$\{\{1\}\} A \{Q\} = A ! \bigvee_{i \in \{0,1\}, q \in Q} \langle \text{rd} \rangle (\langle \text{wr}_i \rangle (\text{ret}), \langle \text{wr}_q \rangle (\text{ret}))$$

The plan: modal logic based c. ref. types

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

The plan: modal logic based c. ref. types

- ◎ **Simple idea:** exploit the **underlying tree-like structure** of **effects**!

- $\langle \text{op} \rangle(\psi_1, \dots, \psi_n)$ for each n -ary operation symbol (cf TYPES'15)

- ◎ **Major pros:**

- uniform across all algebraic effects
- can already encode **Hoare Logic**, Session Types, $\text{HL} \otimes \text{ST}$, ...

- ◎ **Challenges:**

- non-linear effect equations
- 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 \otimes I/O, ...)
 - generative instances and locality

Many possible applications

- ◎ Separation Logic (state, I/O, state \otimes I/O, ...)
 - **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 \otimes I/O, ...)
 - **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 \otimes I/O, ...)
 - **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 planning

◎ Year 1

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

◎ Year 2

- declarative PL design (type-and-effect system)
- meta-theory (denotational and operational)
- encodings of existing specification styles

◎ Year 3

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

Temporal planning



◎ Year 1

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

◎ Year 2

- declarative PL design (type-and-effect system)
- meta-theory (denotational and operational)
- encodings of existing specification styles

◎ Year 3

- algorithmic PL
- implementation
- case studies a

◎ In parallel

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

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
 - inspired by algebraic effects and effect handlers
 - exploiting modalities in computational refinement types
 - both foundational theory and exciting applications