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 h =
| nil : Vec a 0
| cons : ... -> 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**

```
Odd ⊑ Nat     Even ⊑ Nat
Vec A n = {l:List A | len l = n}
```

- **Dependent types**

```
Vec a h =
| nil  : Vec a 0
| cons : ... -> 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)**

$M : ST\ A\ wp_{ST}$

$\cancel{M : IO\ A\ wp_{IO}}$

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

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

- **Graded monads, param. monads**

Research vision: let's be uniform!

Research vision: let's be uniform!

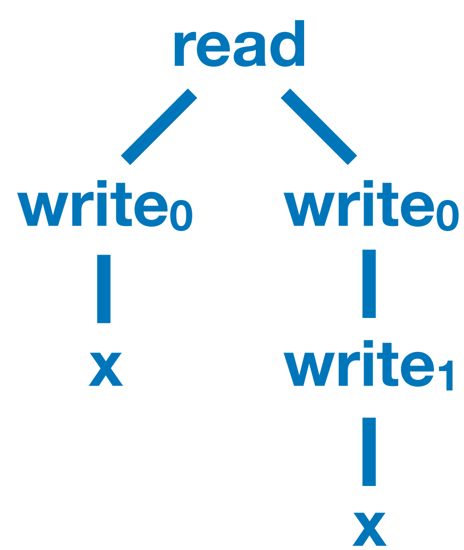
- **Goal:** a general, uniform framework for reasoning about effects
  - wide range of effects (state, I/O, exceptions, probability, ...)
  - primitive and user-defined effects
  - combinations of effects

Research vision: let's be uniform!

- **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
  - given by operations and equations
  - reveal the fundamental underlying tree-like structure
  - handlers are homomorphic tree transformers

Research vision: let's be uniform!

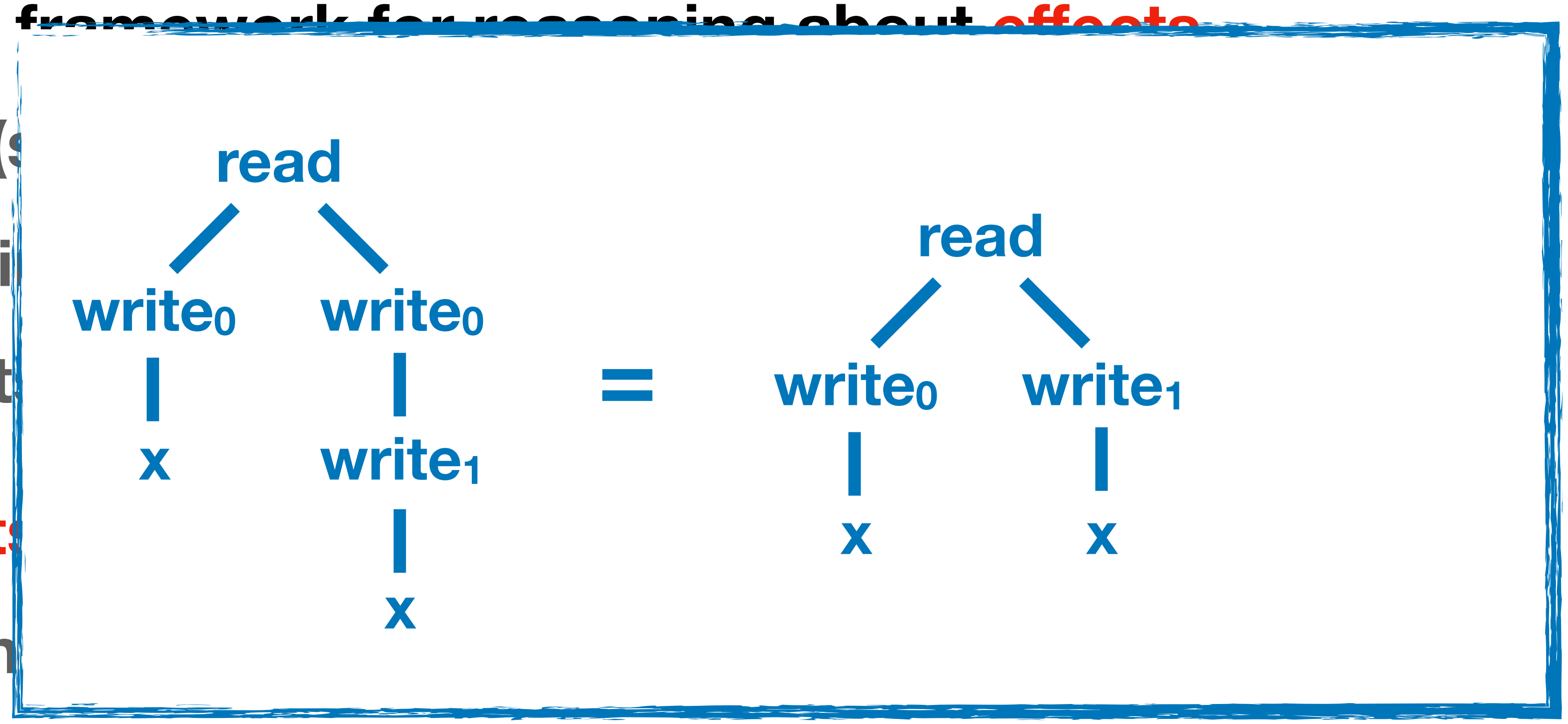
- **Goal:** a general, uniform framework for reasoning about effects
  - wide range of effects (s
  - primitive and user-defi
  - combinations of effect



- **Answer: algebraic effect**
  - given by operations an
  - reveal the **fundamental underlying tree-like structure**
  - handlers are homomorphic tree transformers

Research vision: let's be uniform!

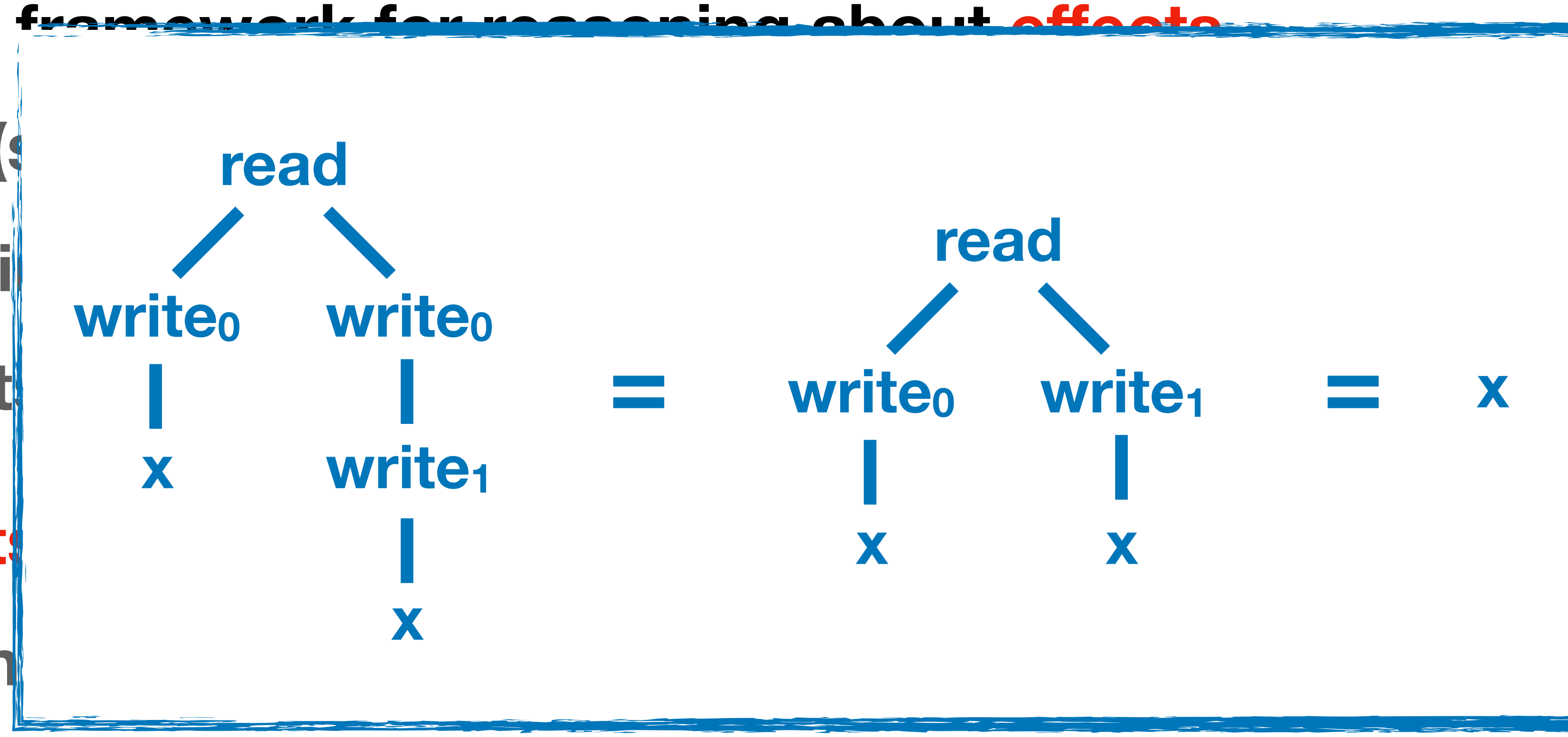
- **Goal:** a general, uniform framework for reasoning about effects
  - wide range of effects (s
  - primitive and user-defi
  - combinations of effect
- **Answer:** algebraic effect



  - given by operations an
  - reveal the fundamental underlying tree-like structure
  - handlers are homomorphic tree transformers

Research vision: let's be uniform!

- **Goal:** a general, uniform framework for reasoning about effects
  - wide range of effects (s
  - primitive and user-defi
  - combinations of effect
- **Answer: algebraic effects**
  - given by operations an
  - reveal the **fundamental underlying tree-like structure**
  - handlers are homomorphic tree transformers



Research vision: let's be uniform!

- **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
  - given by operations and equations
  - reveal the fundamental underlying tree-like structure
  - handlers are homomorphic tree transformers

Research vision: let's be uniform!

- **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
  - given by operations and equations
  - reveal the fundamental underlying tree-like structure
  - handlers are homomorphic tree transformers
- **State of the art:** very popular (!) but effect systems too coarse grained (!)
  - concurrency, delimited control, monadic reflection, ...
  - Multicore OCaml , Uber's Pyro tool , Eff , Koka , Frank , ...
  - `M : A ! { read , write, throw }`

Specifics: modal logic based c. ref. types

Specifics: modal logic based c. ref. types

- **Simple idea: exploit the underlying tree-like structure of effects!**
  - comp. ref. types should denote sets of comp. trees
  - <op>($\psi_1, \ldots, \psi_n$) for each n-ary operation symbol
  - I presented a proof-of-concept experiment back at TYPES'15

Specifics: modal logic based c. ref. types

- **Simple idea: exploit the underlying tree-like structure of effects!**
  - comp. ref. types should denote sets of comp. trees
  - <op>(ψ_1 , ... , ψ_n) for each n-ary operation symbol
  - I presented a proof-of-concept experiment back at TYPES'15

$$\langle op \rangle(\psi_1 , \dots , \psi_n) = \left\{ \begin{array}{ccc} & op & \\ \diagup & & \diagdown \\ t_1 & \cdots & t_n \end{array} \;\middle|\; t_1 \in \psi_1 \wedge \cdots \wedge t_n \in \psi_n \right\}$$

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

Specifics: modal logic based c. ref. types

- **Simple idea: exploit the underlying tree-like structure of effects!**
  - comp. ref. types should denote sets of comp. trees
  - <op>(ψ_1 , ... , ψ_n) for each n-ary operation symbol
  - I presented a proof-of-concept experiment back at TYPES'15

Specifics: modal logic based c. ref. types

- **Simple idea: exploit the underlying tree-like structure of effects!**
  - comp. ref. types should denote sets of comp. trees
  - <op>(ψ_1 , ... , ψ_n) for each n-ary operation symbol
  - I presented a proof-of-concept experiment back at TYPES'15
- **Major pros:**
  - uniform across all algebraic effects
  - can encode Hoare Logic, Session Types, HL ⊗ ST, ...

Specifics: modal logic based c. ref. types

- **Simple idea: exploit the underlying tree-like structure of effects!**
  - comp. ref. types should denote sets of comp. trees
  - <op>(ψ_1 , ... , ψ_n) for each n-ary operation symbol
  - I presented a proof-of-concept experiment back at TYPES'15
- **Major pros:**
  - uniform across all algebraic effects
  - can encode Hoare Logic, Session Types, HL ⊗ ST, ...
- **Challenges:**
  - operations taking arguments and binding values
  - lifting non-linear effect equations to such refinement types
  - effect instances, generativity, and locality (my current focus in LJ)

Many possible applications

Many possible applications

- **Separation Logic (state, I/O, state ⊗ I/O, ...)**
    - generative instances of algebraic effects

Many possible applications

- **Separation Logic (state, I/O, state $\otimes$ I/O, ...)**
  - generative instances of algebraic effects
- **Big Data Computations**
  - commutative monoid structure (an algebraic effect)
  - partitioning, spatial layout, ...

Many possible applications

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

Many possible applications

- **Separation Logic (state, I/O, state $\otimes$ I/O, ...)**
    - generative instances of algebraic effects
- **Big Data Computations**
    - commutative monoid structure (an algebraic effect)
    - partitioning, spatial layout, ...
- **Concurrency**
    - (multi-)handlers based concurrency
    - Scala's promises and futures as an algebraic effect
- **Probabilistic programming**
    - algebraic effects and handlers as in Uber's Pyro tool

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

Conclusions

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
  - with a wide range of potential application areas