Towards refined notions of computation: the global state example

Danel Ahman

LFCS, University of Edinburgh

20 December 2012

joint work with Gordon Plotkin and Alex Simpson

THE UNIVERSITY of EDINBURGH informatics

lfcs Laboratory for Foundations of Computer Science

Overview

- Moggi's monadic approach to computational effects
- Lawvere theories

  and the computational effects they identify
- Refinement types

  and adding more detailed specifications
- Refinement types + Lawvere theories = ?

  on an example of refined global state

Moggi's monadic approach

Moggi's monadic approach

- Semantics of pure simply-typed lambda calculus:
  - take a cartesian-closed category $\mathcal{C}$
  - interpret base types $\alpha, \beta, \ldots$ as objects $[\![\alpha]\!], [\![\beta]\!], \ldots$
  - interpret product type as finite product structure on $\mathcal{C}$
  - interpret (pure) function type $\sigma \to \tau$

    as the exponential $[\![\sigma]\!] \Rightarrow [\![\tau]\!]$
  - interpret value terms $\Gamma \vdash t : \sigma$ as morphisms $[\![\Gamma]\!] \longrightarrow [\![\sigma]\!]$

- Moggi's insight for impure languages:
  - use a strong monad $T : \mathcal{C} \longrightarrow \mathcal{C}$

    to model computational effects
  - $T[\![\sigma]\!]$ stands for computations returning values from $[\![\sigma]\!]$
  - interpret impure function type $\sigma \rightharpoonup \tau$

    as the Kleisli exponential $[\![\sigma]\!] \Rightarrow T[\![\tau]\!]$
  - interpret computations as Kleisli maps $[\![\Gamma]\!] \longrightarrow T[\![\sigma]\!]$

Moggi's monadic approach

- Semantics of pure simply-typed lambda calculus:
  - take a cartesian-closed category $\mathcal{C}$
  - interpret base types $\alpha, \beta, \ldots$ as objects $[\![\alpha]\!], [\![\beta]\!], \ldots$
  - interpret product type as finite product structure on $\mathcal{C}$
  - interpret (pure) function type $\sigma \to \tau$
    as the exponential $[\![\sigma]\!] \Rightarrow [\![\tau]\!]$
  - interpret value terms $\Gamma \vdash t : \sigma$ as morphisms $[\![\Gamma]\!] \longrightarrow [\![\sigma]\!]$

- Moggi's insight for impure languages:
  - use a strong monad $T : \mathcal{C} \longrightarrow \mathcal{C}$
    to model computational effects
  - $T[\![\sigma]\!]$ stands for computations returning values from $[\![\sigma]\!]$
  - interpret impure function type $\sigma \rightharpoonup \tau$
    as the Kleisli exponential $[\![\sigma]\!] \Rightarrow T[\![\tau]\!]$
  - interpret computations as Kleisli maps $[\![\Gamma]\!] \longrightarrow T[\![\sigma]\!]$

Moggi's monadic approach

- Example monads proposed by Moggi
  - exceptions - $TX = X + E$
  - global state - $TX = (S \times X)^S$
    - *(stateful computations $S \times X \longrightarrow S \times Y$)*
  - local state - $(TX)_n = (\int^{m \in (n/I)} (S_m \times X_m))^{S_n}$
  - finite nondeterminism - $TX = \mathcal{F}^+ X$
  - continuations - $TX = R^{R^X}$
- Also possible to combine different monads, e.g.,
  - state plus exceptions - $TX = (S \times (X + E))^S$

Moggi's monadic approach

- Moggi's work gives us an elegant denotational semantics of computational effects
- However, this denotation does not tell us much about how to construct such effects
- We have to note their operational meaning and how such effects (e.g., state) are implemented in programming languages

Lawvere theories

Lawvere theories

- A countable Lawvere theory consists of:
  - a small category $\mathcal{L}$ with countable products
  - an id. on objects countable-product preserving functor

    $$J : \aleph_1^{op} \longrightarrow \mathcal{L}$$
  - *(where $\aleph_1$ is the skeleton of the category of countable sets)*

- Think of the hom $\mathcal{L}(n, 1)$ (abbrv. $\mathcal{L}(J(n), J(1))$) as a set of n-ary operations in the theory

- Then it suffices to give an algebraic theory as:
  - operations of are given by morphisms $op : O \longrightarrow I$
    - *(equivalently a family of operations $op_{i \in I} : O \longrightarrow 1$)*
  - equations are given by commuting diagrams

Models of Lawvere theories

- A model of a Lawvere theory $(\mathcal{L}, J)$ in a category $\mathcal{C}$ with countable products
  - is a countable product preserving functor $M : \mathcal{L} \longrightarrow \mathcal{C}$
- The models of $\mathcal{L}$ together with nat. transfs. :
  - form a category $Mod(\mathcal{L}, \mathcal{C})$ with $U : Mod(\mathcal{L}, \mathcal{C}) \longrightarrow \mathcal{C}$
  - having a left adjoint $F : \mathcal{C} \longrightarrow Mod(\mathcal{L}, \mathcal{C})$
  - the adjoint functors induce a monad $T = UF$
- For the purposes of this talk, we let $\mathcal{C} = \mathrm{Set}$
- To give a model M of $\mathcal{L}$ is equivalent to
  - giving a set $X = M1$
  - for every operation $op : O \longrightarrow I$ a morphism $X^O \longrightarrow X^I$
- Because
  - $M1$ determines MO up to coherent isomorphism
  - $MO \cong M(\prod_{o \in O} 1) \cong \prod_{o \in O} (M1) \cong (M1)^O$

Models of Lawvere theories

- A model of a Lawvere theory $(\mathcal{L}, J)$ in a category $\mathcal{C}$ with countable products
  - is a countable product preserving functor $M : \mathcal{L} \longrightarrow \mathcal{C}$
- The models of $\mathcal{L}$ together with nat. transfs. :
  - form a category $Mod(\mathcal{L}, \mathcal{C})$ with $U : Mod(\mathcal{L}, \mathcal{C}) \longrightarrow \mathcal{C}$
  - having a left adjoint $F : \mathcal{C} \longrightarrow Mod(\mathcal{L}, \mathcal{C})$
  - the adjoint functors induce a monad $T = UF$
- For the purposes of this talk, we let $\mathcal{C} = \mathrm{Set}$
- To give a model M of $\mathcal{L}$ is equivalent to
  - giving a set $X = M1$
  - for every operation $op : O \longrightarrow I$ a morphism $X^O \longrightarrow X^I$
- Because
  - $M1$ determines MO up to coherent isomorphism
  - $MO \cong M(\prod_{o \in O} 1) \cong \prod_{o \in O} (M1) \cong (M1)^O$

Global state example

- Plotkin and Power noticed that the global state monad is determined by the following countable Lawvere theory
- Countable set of values V and a finite set of locations Loc
- Take the set of states to be $S = V^{Loc}$
- The theory is freely generated by operations
  - $lookup : V \longrightarrow Loc$
  - $update : 1 \longrightarrow Loc \times V$

  subject to commuting diagrams expressed set-theoretically

  1. $lookup_{loc}(update_{loc,v}(x))_v = x$
  2. $lookup_{loc}(lookup_{loc}(x_{vv'})_v)_{v'} = lookup_{loc}(x_{vv})_v$
  3. $update_{loc,v}(update_{loc,v'}(x)) = update_{loc,v'}(x)$
  4. $update_{loc,v}(read_{loc}(x'_v)'_v) = update_{loc,v}(x_v)$
  5. $update_{loc,v}(update_{loc',v'}(x)) =$ $update_{loc',v'}(update_{loc,v}(x)) \qquad (loc \neq loc')$
  6. ...

Global state example

- Plotkin and Power noticed that the global state monad is determined by the following countable Lawvere theory
- Countable set of values V and a finite set of locations Loc
- Take the set of states to be $S = V^{Loc}$
- The theory is freely generated by operations
  - $lookup : V \longrightarrow Loc$
  - $update : 1 \longrightarrow Loc \times V$

subject to commuting diagrams expressed set-theoretically

1. $lookup_{loc}(update_{loc,v}(x))_v = x$
2. $lookup_{loc}(lookup_{loc}(x_{vv'})_v)_{v'} = lookup_{loc}(x_{vv})_v$
3. $update_{loc,v}(update_{loc,v'}(x)) = update_{loc,v'}(x)$
4. $update_{loc,v}(read_{loc}(x'_v)'_v) = update_{loc,v}(x_v)$
5. $update_{loc,v}(update_{loc',v'}(x)) =$
   $update_{loc',v'}(update_{loc,v}(x)) \qquad (loc \neq loc')$
6. ...

Global state example

- Plotkin and Power noticed that the global state monad is determined by the following countable Lawvere theory
- Countable set of values V and a finite set of locations Loc
- Take the set of states to be $S = V^{Loc}$
- The theory is freely generated by operations
  - $lookup : V \longrightarrow Loc$
  - $update : 1 \longrightarrow Loc \times V$

  subject to commuting diagrams expressed set-theoretically

  1. $lookup_{loc}(update_{loc,v}(x))_v = x$
  2. $lookup_{loc}(lookup_{loc}(x_{vv'})_v)_{v'} = lookup_{loc}(x_{vv})_v$
  3. $update_{loc,v}(update_{loc,v'}(x)) = update_{loc,v'}(x)$
  4. $update_{loc,v}(read_{loc}(x'_v)'_v) = update_{loc,v}(x_v)$
  5. $update_{loc,v}(update_{loc',v'}(x)) =$
     $update_{loc',v'}(update_{loc,v}(x)) \qquad (loc \neq loc')$
  6. ...

Global state example

- Plotkin and Power noticed that the global state monad is determined by the following countable Lawvere theory
- Countable set of values V and a finite set of locations Loc
- Take the set of states to be $S = V^{Loc}$
- The theory is freely generated by operations
  - $lookup : V \longrightarrow Loc$
  - $update : 1 \longrightarrow Loc \times V$

  subject to commuting diagrams expressed set-theoretically

  1. $lookup_{loc}(update_{loc,v}(x))_v = x$
  2. $lookup_{loc}(lookup_{loc}(x_{vv'})_v)_{v'} = lookup_{loc}(x_{vv})_v$
  3. $update_{loc,v}(update_{loc,v'}(x)) = update_{loc,v'}(x)$
  4. $update_{loc,v}(read_{loc}(x'_v)'_v) = update_{loc,v}(x_v)$
  5. $update_{loc,v}(update_{loc',v'}(x)) = update_{loc',v'}(update_{loc,v}(x)) \qquad (loc \neq loc')$
  6. $\ldots$

Global state example

- Plotkin and Power noticed that the global state monad is determined by the following countable Lawvere theory
- Countable set of values V and a finite set of locations Loc
- Take the set of states to be $S = V^{Loc}$
- The theory is freely generated by operations
    - $lookup : V \longrightarrow Loc$
    - $update : 1 \longrightarrow Loc \times V$

  subject to commuting diagrams expressed set-theoretically

  1. $lookup_{loc}(update_{loc,v}(x))_v = x$
  2. $lookup_{loc}(lookup_{loc}(x_{vv'})_v)_{v'} = lookup_{loc}(x_{vv})_v$
  3. $update_{loc,v}(update_{loc,v'}(x)) = update_{loc,v'}(x)$
  4. $update_{loc,v}(read_{loc}(x'_v)'_v) = update_{loc,v}(x_v)$
  5. $update_{loc,v}(update_{loc',v'}(x)) = update_{loc',v'}(update_{loc,v}(x)) \qquad (loc \neq loc')$
  6. ...

Small detour into local state

- $(TX)_n = (\int^{m \in (n/Inj)} (S_m \times X_m))^{S_n}$
- Monad and algebra are given in category Set^{Inj}
  - *(Inj is the category of finite sets and injections)*
- $L_n = Inj(1, n), \qquad V_n = V, \qquad S_n = V^n$
- The algebra is given by
  - $lookup : X^V \longrightarrow X^{Loc}$
  - $update : X \longrightarrow X^{Loc \times V}$
  - $block : [L, X] \longrightarrow X^V$
  - subject to appropriate diagrams commuting
- $(Y^X)_n = [Inj, \mathrm{Set}](X - \times Inj(n, -), Y-)$
- $[X, Y]_n = [Inj, \mathrm{Set}](X-, Y(n + -))$
- See also work by Power (cotensoring models with comodels) and Staton (completeness via nominal sets)

Small detour into local state

- $(TX)_n = (\int^{m \in (n/Inj)} (S_m \times X_m))^{S_n}$
- Monad and algebra are given in category Set^{Inj}
  - *(Inj is the category of finite sets and injections)*
- $L_n = Inj(1, n), \qquad V_n = V, \qquad S_n = V^n$
- The algebra is given by
  - $lookup : X^V \longrightarrow X^{Loc}$
  - $update : X \longrightarrow X^{Loc \times V}$
  - $block : [L, X] \longrightarrow X^V$
  - subject to appropriate diagrams commuting
- $(Y^X)_n = [Inj, \mathrm{Set}](X - \times Inj(n, -), Y-)$
- $[X, Y]_n = [Inj, \mathrm{Set}](X-, Y(n + -))$
- See also work by Power (cotensoring models with comodels) and Staton (completeness via nominal sets)

Small detour into local state

- $(TX)_n = (\int^{m \in (n/Inj)} (S_m \times X_m))^{S_n}$
- Monad and algebra are given in category Set^{Inj}
  - *(Inj is the category of finite sets and injections)*
- $L_n = Inj(1, n), \qquad V_n = V, \qquad S_n = V^n$
- The algebra is given by
  - $lookup : X^V \longrightarrow X^{Loc}$
  - $update : X \longrightarrow X^{Loc \times V}$
  - $block : [L, X] \longrightarrow X^V$
  - subject to appropriate diagrams commuting
- $(Y^X)_n = [Inj, \mathrm{Set}](X - \times Inj(n, -), Y-)$
- $[X, Y]_n = [Inj, \mathrm{Set}](X-, Y(n + -))$
- See also work by Power (cotensoring models with comodels) and Staton (completeness via nominal sets)

Small detour into local state

- $(TX)_n = (\int^{m \in (n/Inj)} (S_m \times X_m))^{S_n}$
- Monad and algebra are given in category Set^{Inj}
    - *(Inj is the category of finite sets and injections)*
- $L_n = Inj(1, n), \qquad V_n = V, \qquad S_n = V^n$
- The algebra is given by
    - $lookup : X^V \longrightarrow X^{Loc}$
    - $update : X \longrightarrow X^{Loc \times V}$
    - $block : [L, X] \longrightarrow X^V$
    - subject to appropriate diagrams commuting
- $(Y^X)_n = [Inj, \mathrm{Set}](X - \times Inj(n, -), Y-)$
- $[X, Y]_n = [Inj, \mathrm{Set}](X-, Y(n + -))$
- See also work by Power (cotensoring models with comodels) and Staton (completeness via nominal sets)

Refinement types

Refinement types

- Also known as predicate subtyping

- Assume we are given some simple types
  - $Nat, Loc, \ldots$

- But often we want to talk about refined versions of them
  - even natural numbers
  - odd natural numbers
  - open locations
  - closed locations

- Refinement types provide us with such a framework

  - "equipping your existing type system with suitable logic"

Refinement types

- Well-formedness of refinement types

$$\frac{}{\Gamma \vdash \sigma : \mathit{Ref}(\sigma)} \qquad \frac{\Gamma \vdash \phi : \mathit{Ref}(\sigma) \quad \Gamma, x : \phi \vdash P : \mathit{wf}}{\Gamma \vdash (x : \phi)P : \mathit{Ref}(\sigma)}$$

$$\frac{\Gamma \vdash \phi : \mathit{Ref}(\sigma_1) \quad \Gamma, x : \phi \vdash \psi : \mathit{Ref}(\sigma_2)}{\Gamma \vdash \Sigma_{x:\phi}\psi : \mathit{Ref}(\sigma_1 \times \sigma_2)} \qquad \frac{\Gamma \vdash \phi : \mathit{Ref}(\sigma) \quad \Gamma, x : \phi \vdash \psi : \mathit{Ref}(\tau)}{\Gamma \vdash \Pi_{x:\phi}\psi : \mathit{Ref}(\sigma \to \tau)}$$

- Examples of typing rules

$$\frac{\Gamma \vdash t : \phi \quad \Gamma \vdash P[t/x]}{\Gamma \vdash t : (x : \phi)P}$$

$$\frac{\Gamma, x : \phi \vdash t : \psi}{\Gamma \vdash \lambda x : \phi . t : \Pi_{x:\phi}\psi} \qquad \frac{\Gamma \vdash t_1 : \Pi_{x:\phi}\psi \quad \Gamma \vdash t_2 : \phi}{\Gamma \vdash t_1 t_2 : \psi[t_2/x]}$$

Refinement types

- Well-formedness of refinement types

$$\frac{}{\Gamma \vdash \sigma : \mathit{Ref}(\sigma)} \qquad \frac{\Gamma \vdash \phi : \mathit{Ref}(\sigma) \quad \Gamma, x : \phi \vdash P : \mathit{wf}}{\Gamma \vdash (x : \phi)P : \mathit{Ref}(\sigma)}$$

$$\frac{\Gamma \vdash \phi : \mathit{Ref}(\sigma_1) \quad \Gamma, x : \phi \vdash \psi : \mathit{Ref}(\sigma_2)}{\Gamma \vdash \Sigma_{x:\phi}\psi : \mathit{Ref}(\sigma_1 \times \sigma_2)} \quad \frac{\Gamma \vdash \phi : \mathit{Ref}(\sigma) \quad \Gamma, x : \phi \vdash \psi : \mathit{Ref}(\tau)}{\Gamma \vdash \Pi_{x:\phi}\psi : \mathit{Ref}(\sigma \to \tau)}$$

- Examples of typing rules

$$\frac{\Gamma \vdash t : \phi \quad \Gamma \vdash P[t/x]}{\Gamma \vdash t : (x : \phi)P}$$

$$\frac{\Gamma, x : \phi \vdash t : \psi}{\Gamma \vdash \lambda x : \phi . t : \Pi_{x:\phi}\psi} \quad \frac{\Gamma \vdash t_1 : \Pi_{x:\phi}\psi \quad \Gamma \vdash t_2 : \phi}{\Gamma \vdash t_1\, t_2 : \psi[t_2/x]}$$

Refinement types

- Set-theoretic semantics (ala. Denney)
  - Interpret refinement type $\Gamma \vdash \phi : \mathit{Ref}(\sigma)$
    as a family of PERs $[\![\Gamma]\!] \longrightarrow \mathit{PER}([\![\sigma]\!])$
  - other type constructors (sums,products) are interpreted straightforwardly
  - terms $\Gamma \vdash t : \phi$ are interpreted as $[\![\Gamma]\!] \longrightarrow \mathcal{P}([\![\sigma]\!])$
    (subsets denoting the 'total realizers')

- Categorical semantics (ala. Jacobs)
  - based on fibrations and comprehension categories

$$\begin{tikzcd}
\mathbb{P} \arrow[dr] & \mathbb{T} \arrow[r] \arrow[d] & \mathbb{C}^{\to} \arrow[dl, "cod"] \\
 & \mathbb{C} &
\end{tikzcd}$$

Refinement types

- Set-theoretic semantics (ala. Denney)
  - Interpret refinement type $\Gamma \vdash \phi : \mathit{Ref}(\sigma)$
    as a family of PERs $[\![\Gamma]\!] \longrightarrow \mathit{PER}([\![\sigma]\!])$
  - other type constructors (sums,products) are interpreted straightforwardly
  - terms $\Gamma \vdash t : \phi$ are interpreted as $[\![\Gamma]\!] \longrightarrow \mathcal{P}([\![\sigma]\!])$
    (subsets denoting the 'total realizers')

- Categorical semantics (ala. Jacobs)
  - based on fibrations and comprehension categories

$$
\begin{tikzcd}
\mathbb{P} \arrow[dr] & \mathbb{T} \arrow[r] \arrow[d] & \mathbb{C}^{\to} \arrow[dl, "\mathit{cod}"] \\
& \mathbb{C} &
\end{tikzcd}
$$

Refining global state

Refining global state

- We had the finite set of locations Loc
- Assume that we now have predicates $Open(Loc)$ and $Closed(Loc) = \neg Open(loc)$ on the locations Loc
- Conceptually they denote subsets of Loc
- We should only be able to read from and write to locations that are open
  - $lookup : X^V \longrightarrow X^{Open(Loc)}$
  - $update : X \longrightarrow X^{Open(Loc) \times V}$
- However, notice that this requires an a priori given collection of open locations

Refining global state

- We had the finite set of locations Loc
- Assume that we now have predicates $Open(Loc)$ and $Closed(Loc) = \neg Open(loc)$ on the locations Loc
- Conceptually they denote subsets of Loc
- We should only be able to read from and write to locations that are open
  - $lookup : X^V \longrightarrow X^{Open(Loc)}$
  - $update : X \longrightarrow X^{Open(Loc) \times V}$
- However, notice that this requires an a priori given collection of open locations

Refining global state

- We had the finite set of locations Loc
- Assume that we now have predicates $Open(Loc)$ and $Closed(Loc) = \neg Open(loc)$ on the locations Loc
- Conceptually they denote subsets of Loc
- We should only be able to read from and write to locations that are open
  - $lookup : X^V \longrightarrow X^{Open(Loc)}$
  - $update : X \longrightarrow X^{Open(Loc) \times V}$
- However, notice that this requires an a priori given collection of open locations

Refining global state

- So we should also add operations for opening and closing locations
  - $lookup : X^V \longrightarrow X^{Open(Loc)}$
  - $update : X \longrightarrow X^{Open(Loc) \times V}$
  - $open : X \longrightarrow X^{Closed(Loc)}$
  - $close : X \longrightarrow X^{Open(Loc)}$
- But we should be able to distinguish between computations able to use different locations
- We could take inspiration from the algebra for local state
  - work in the category Set^W
- However, we don't yet know what the appropriate non-discrete world category and the corresponding (monoidal) closed structure should be

Refining global state

- So we should also add operations for opening and closing locations
  - $lookup : X^V \longrightarrow X^{Open(Loc)}$
  - $update : X \longrightarrow X^{Open(Loc) \times V}$
  - $open : X \longrightarrow X^{Closed(Loc)}$
  - $close : X \longrightarrow X^{Open(Loc)}$
- But we should be able to distinguish between computations able to use different locations
- We could take inspiration from the algebra for local state
  - work in the category Set^W
- However, we don't yet know what the appropriate non-discrete world category and the corresponding (monoidal) closed structure should be

Refining global state

- So we should also add operations for opening and closing locations
    - $lookup : X^V \longrightarrow X^{Open(Loc)}$
    - $update : X \longrightarrow X^{Open(Loc) \times V}$
    - $open : X \longrightarrow X^{Closed(Loc)}$
    - $close : X \longrightarrow X^{Open(Loc)}$
- But we should be able to distinguish between computations able to use different locations
- We could take inspiration from the algebra for local state
    - work in the category Set^W
- However, we don't yet know what the appropriate non-discrete world category and the corresponding (monoidal) closed structure should be

Refining global state

- So we should also add operations for opening and closing locations
  - $lookup : X^V \longrightarrow X^{Open(Loc)}$
  - $update : X \longrightarrow X^{Open(Loc) \times V}$
  - $open : X \longrightarrow X^{Closed(Loc)}$
  - $close : X \longrightarrow X^{Open(Loc)}$
- But we should be able to distinguish between computations able to use different locations
- We could take inspiration from the algebra for local state
  - work in the category Set^W
- However, we don't yet know what the appropriate non-discrete world category and the corresponding (monoidal) closed structure should be

Refining global state (W-sorted theories)

- We don't know the definition in a single sorted theory
- So let's try to work in W-sorted algebraic theories
- A W-sorted algebraic theory consists of:
  - a set of sorts W (we think of them as worlds)
  - a small category $\mathcal{L}$ with countable products
  - an id. on objects countable-product preserving functor

    $$J : W^* \longrightarrow \mathcal{L}$$

  - *(where W^* has as objects words $w_0, \ldots, w_n$ over W)*
- A model of a W-sorted theory is given by
  - a countable product preserving functor $M : \mathcal{L} \longrightarrow \mathrm{Set}$
- The forgetful functor $U : Mod(\mathcal{L}, \mathrm{Set}) \longrightarrow \mathrm{Set}^W$ again has a left adjoint F inducing a monad $T = UF$

Refining global state (W-sorted theories)

- We don't know the definition in a single sorted theory
- So let's try to work in W-sorted algebraic theories

- A W-sorted algebraic theory consists of:
  - a set of sorts W (we think of them as worlds)
  - a small category $\mathcal{L}$ with countable products
  - an id. on objects countable-product preserving functor

    $$J : W^* \longrightarrow \mathcal{L}$$
  - *(where W^* has as objects words $w_0, \ldots, w_n$ over W)*

- A model of a W-sorted theory is given by
  - a countable product preserving functor $M : \mathcal{L} \longrightarrow \mathrm{Set}$

- The forgetful functor $U : Mod(\mathcal{L}, \mathrm{Set}) \longrightarrow \mathrm{Set}^W$ again has a left adjoint F inducing a monad $T = UF$

Refining global state (W-sorted theories)

- We don't know the definition in a single sorted theory
- So let's try to work in W-sorted algebraic theories

- A W-sorted algebraic theory consists of:
  - a set of sorts W (we think of them as worlds)
  - a small category $\mathcal{L}$ with countable products
  - an id. on objects countable-product preserving functor

    $$J : W^* \longrightarrow \mathcal{L}$$

  - *(where W^* has as objects words $w_0, \ldots, w_n$ over W)*

- A model of a W-sorted theory is given by
  - a countable product preserving functor $M : \mathcal{L} \longrightarrow \mathrm{Set}$

- The forgetful functor $U : Mod(\mathcal{L}, \mathrm{Set}) \longrightarrow \mathrm{Set}^W$ again has a left adjoint F inducing a monad $T = UF$

Refining global state (W-sorted theories)

- We don't know the definition in a single sorted theory
- So let's try to work in W-sorted algebraic theories

- A W-sorted algebraic theory consists of:
  - a set of sorts W (we think of them as worlds)
  - a small category $\mathcal{L}$ with countable products
  - an id. on objects countable-product preserving functor

$$J : W^* \longrightarrow \mathcal{L}$$

  - *(where W^* has as objects words $w_0, \ldots, w_n$ over W)*

- A model of a W-sorted theory is given by
  - a countable product preserving functor $M : \mathcal{L} \longrightarrow \mathrm{Set}$

- The forgetful functor $U : Mod(\mathcal{L}, \mathrm{Set}) \longrightarrow \mathrm{Set}^W$ again has a left adjoint F inducing a monad $T = UF$

Refining global state (W-sorted theories)

- Let the worlds be $W = Bool^W$

- We have families of operations in the theory
  - $lookup_{w \in W, loc \in Open_w(Loc)} : w, \ldots., w \longrightarrow w$
  - $update_{w \in W, loc \in Open_w(Loc), v \in V} : w \longrightarrow w$
  - $open_{w \in W, loc \in Open_w(Loc)} : w \longrightarrow w[loc \mapsto \perp]$
  - $close_{w \in W, loc \in Closed_w(Loc)} : w \longrightarrow w[loc \mapsto \top]$
  - satisfying appropriate commuting diagrams

- Giving us the algebra
  - $lookup_{w \in W, loc \in Open_w(Loc)} : (X^V)_w \longrightarrow X_w$
  - $update_{w \in W, loc \in Open_w(Loc), v \in V} : X_w \longrightarrow X_w$
  - $open_{w \in W, loc \in Open_w(Loc)} : X_w \longrightarrow X_{w[loc \mapsto \perp]}$
  - $close_{w \in W, loc \in Closed_w(Loc)} : X_w \longrightarrow X_{w[loc \mapsto \top]}$

Refining global state (W-sorted theories)

- Let the worlds be $W = Bool^W$

- We have families of operations in the theory
  - $lookup_{w \in W, loc \in Open_w(Loc)} : w, ..., w \longrightarrow w$
  - $update_{w \in W, loc \in Open_w(Loc), v \in V} : w \longrightarrow w$
  - $open_{w \in W, loc \in Open_w(Loc)} : w \longrightarrow w[loc \mapsto \bot]$
  - $close_{w \in W, loc \in Closed_w(Loc)} : w \longrightarrow w[loc \mapsto \top]$
  - satisfying appropriate commuting diagrams

- Giving us the algebra
  - $lookup_{w \in W, loc \in Open_w(Loc)} : (X^V)_w \longrightarrow X_w$
  - $update_{w \in W, loc \in Open_w(Loc), v \in V} : X_w \longrightarrow X_w$
  - $open_{w \in W, loc \in Open_w(Loc)} : X_w \longrightarrow X_{w[loc \mapsto \bot]}$
  - $close_{w \in W, loc \in Closed_w(Loc)} : X_w \longrightarrow X_{w[loc \mapsto \top]}$

Refining global state (W-sorted theories)

- So we have the algebra
  - $lookup_{w \in W, loc \in Open_w(Loc)} : (X^V)_w \longrightarrow X_w$
  - $update_{w \in W, loc \in Open_w(Loc), v \in V} : X_w \longrightarrow X_w$
  - $open_{w \in W, loc \in Open_w(Loc)} : X_w \longrightarrow X_{w[loc \mapsto \bot]}$
  - $close_{w \in W, loc \in Closed_w(Loc)} : X_w \longrightarrow X_{w[loc \mapsto \top]}$
- Inducing monad $TX_w = UFX_w = (\sum_{w' \in W} (S_{w'} \times X_{w'}))^{S_w}$
- With the unit $\eta_X : X \longrightarrow UFX$ of the adjunction given by:

  $\eta_{X,w}\, \gamma = \lambda s \,.\, \mathrm{inj}_w\, (s\, ,\, \gamma)$
- And the counit $\varepsilon_A : FUA \longrightarrow A$ of the adjunction:

$$\varepsilon_{A,w} = (\coprod(S \times A_{w'}))^S \xrightarrow{(\coprod(S \times \overrightarrow{close}))^S} (\coprod(S \times A_{w^\top}))^S \xrightarrow{\cong} (S \times A_{w^\top})^S \xrightarrow{(\overrightarrow{write})^S} (A_{w^\top})^S \xrightarrow{\overrightarrow{read}} A_{w^\top} \xrightarrow{\overrightarrow{open}} A_w$$

- And the Kleisli extension is given by $(-)^* = U \varepsilon F$

Refining global state (W-sorted theories)

- So we have the algebra
  - $lookup_{w \in W, loc \in Open_w(Loc)} : (X^V)_w \longrightarrow X_w$
  - $update_{w \in W, loc \in Open_w(Loc), v \in V} : X_w \longrightarrow X_w$
  - $open_{w \in W, loc \in Open_w(Loc)} : X_w \longrightarrow X_{w[loc \mapsto \bot]}$
  - $close_{w \in W, loc \in Closed_w(Loc)} : X_w \longrightarrow X_{w[loc \mapsto \top]}$
- Inducing monad $TX_w = UFX_w = (\sum_{w' \in W} (S_{w'} \times X_{w'}))^{S_w}$
- With the unit $\eta_X : X \longrightarrow UFX$ of the adjunction given by:

  $$\eta_{X,w}\,\gamma = \lambda s\,.\,\mathrm{inj}_w\,(s\,,\,\gamma)$$

- And the counit $\varepsilon_A : FUA \longrightarrow A$ of the adjunction:

  $$\varepsilon_{A,w} = (\coprod(S \times A_{w'}))^S \xrightarrow{(\coprod(S \times \overrightarrow{close}))^S} (\coprod(S \times A_{w^\top}))^S \xrightarrow{\cong} (S \times A_{w^\top})^S \xrightarrow{(\overrightarrow{write})^S} (A_{w^\top})^S \xrightarrow{\overrightarrow{read}} A_{w^\top} \xrightarrow{\overrightarrow{open}} A_w$$

- And the Kleisli extension is given by $(-)^* = U\varepsilon F$

Refining global state (W-sorted theories)

- So we have the algebra
  - $lookup_{w \in W, loc \in Open_w(Loc)} : (X^V)_w \longrightarrow X_w$
  - $update_{w \in W, loc \in Open_w(Loc), v \in V} : X_w \longrightarrow X_w$
  - $open_{w \in W, loc \in Open_w(Loc)} : X_w \longrightarrow X_{w[loc \mapsto \bot]}$
  - $close_{w \in W, loc \in Closed_w(Loc)} : X_w \longrightarrow X_{w[loc \mapsto \top]}$
- Inducing monad $TX_w = UFX_w = (\sum_{w' \in W} (S_{w'} \times X_{w'}))^{S_w}$
- With the unit $\eta_x : X \longrightarrow UFX$ of the adjunction given by:

  $$\eta_{x,w}\,\gamma = \lambda s\,.\,\mathrm{inj}_w\,(s\,,\,\gamma)$$
- And the counit $\varepsilon_A : FUA \longrightarrow A$ of the adjunction:

  $$\varepsilon_{A,w} = (\coprod (S \times A_{w'}))^S \xrightarrow{(\coprod(S \times \overrightarrow{close}))^S} (\coprod (S \times A_{w^\top}))^S \xrightarrow{\cong} (S \times A_{w^\top})^S \xrightarrow{(\overrightarrow{write})^S} (A_{w^\top})^S \xrightarrow{\overrightarrow{read}} A_{w^\top} \xrightarrow{\overrightarrow{open}} A_w$$
- And the Kleisli extension is given by $(-)^* = U\varepsilon F$

Another example of a straightforward theory

- Inspiration from McBride's work on file operations
- Take the simple set of worlds $W = Bool$
- We are interested in axiomatizing logging in to and logging off from some system
- We have the theory
  - $LogIn_{p \in Password} : true, false \longrightarrow false$
  - $DoSomething : true \longrightarrow true$
  - $LogOut : false \longrightarrow true$
- And the algebra
  - $LogIn_{p \in Password} : X_{true} \times X_{false} \longrightarrow X_{false}$
  - $DoSomething : X_{true} \longrightarrow X_{true}$
  - $LogOut : X_{false} \longrightarrow X_{true}$
- However, *LogIn* not captured by Atkey's parametrized monads as the arguments live in different worlds!

Another example of a straightforward theory

- Inspiration from McBride's work on file operations
- Take the simple set of worlds $W = Bool$
- We are interested in axiomatizing logging in to and logging off from some system
- We have the theory
  - $LogIn_{p \in Password} : true, false \longrightarrow false$
  - $DoSomething : true \longrightarrow true$
  - $LogOut : false \longrightarrow true$
- And the algebra
  - $LogIn_{p \in Password} : X_{true} \times X_{false} \longrightarrow X_{false}$
  - $DoSomething : X_{true} \longrightarrow X_{true}$
  - $LogOut : X_{false} \longrightarrow X_{true}$
- However, *LogIn* not captured by Atkey's parametrized monads as the arguments live in different worlds!

What next?

- The W-sorted approach gave us the monad we were after
- Can we make it work naturally in the singlesorted case?
- Idea, try to give more general form to the operations in the algebra
  - $op_w : \prod_{o \in O_w} X_{\delta_o(w,o)} \longrightarrow \prod_{i \in I_w} X_{\delta_i(w,i)}$

  and in the theory
  - $op_w : \coprod_{o \in O_w} \{\delta_o(w,o)\} \longrightarrow \coprod_{i \in I_w} \{\delta_i(w,i)\}$
- But can't always define them uniformly in w, e.g.:

$$lookup_{[l_i \mapsto \bot]} : \coprod_{v \in V} \{[l_i \mapsto \bot]\} \longrightarrow 0$$

- Seems to be kind of inherent to the idea that not all operations should be definable in all worlds

What next?

- The W-sorted approach gave us the monad we were after
- Can we make it work naturally in the singlesorted case?
- Idea, try to give more general form to the operations in the algebra
  - $op_w : \prod_{o \in O_w} X_{\delta_o(w,o)} \longrightarrow \prod_{i \in I_w} X_{\delta_i(w,i)}$

  and in the theory
  - $op_w : \coprod_{o \in O_w} \{\delta_o(w,o)\} \longrightarrow \coprod_{i \in I_w} \{\delta_i(w,i)\}$
- But can't always define them uniformly in w, e.g.:

  $$lookup_{[l_i \mapsto \perp]} : \coprod_{v \in V} \{[l_i \mapsto \perp]\} \longrightarrow 0$$
- Seems to be kind of inherent to the idea that not all operations should be definable in all worlds

What next?

- The W-sorted approach gave us the monad we were after
- Can we make it work naturally in the singlesorted case?
- Idea, try to give more general form to the operations in the algebra
  - $op_w : \prod_{o \in O_w} X_{\delta_o(w,o)} \longrightarrow \prod_{i \in I_w} X_{\delta_i(w,i)}$

  and in the theory
  - $op_w : \coprod_{o \in O_w} \{\delta_o(w,o)\} \longrightarrow \coprod_{i \in I_w} \{\delta_i(w,i)\}$
- But can't always define them uniformly in w, e.g.:

$$lookup_{[l_i \mapsto \perp]} : \coprod_{v \in V} \{[l_i \mapsto \perp]\} \longrightarrow 0$$

- Seems to be kind of inherent to the idea that not all operations should be definable in all worlds

Questions?