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
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 $\lbrack \alpha \rbrack, \lbrack \beta \rbrack, \ldots$
  - interpret **product type** as finite product structure on $\mathcal{C}$
  - interpret **(pure) function type** $\sigma \rightarrow \tau$
    as the exponential $\lbrack \sigma \rbrack \Rightarrow \lbrack \tau \rbrack$
  - interpret **value terms** $\Gamma \vdash t : \sigma$ as morphisms $\lbrack \Gamma \rbrack \rightarrow \lbrack \sigma \rbrack$

- Moggi’s insight for impure languages:
  - use a strong monad $T : \mathcal{C} \rightarrow \mathcal{C}$ to model computational effects
  - $T \lbrack \sigma \rbrack$ stands for computations returning values from $\lbrack \sigma \rbrack$
  - interpret **impure function type** $\sigma \hookrightarrow \tau$
    as the Kleisli exponential $\lbrack \sigma \rbrack \Rightarrow T \lbrack \tau \rbrack$
  - interpret computations as Kleisli maps $\lbrack \Gamma \rbrack \rightarrow T \lbrack \sigma \rbrack$
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 \( \sigma \rightarrow \tau \)
    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 \( \sigma \mapsto \tau \)
    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 $S \times X \rightarrow S \times Y$)
  - local state - $(TX)_n = \left( \int_{m \in (n/I)} (S_m \times X_m) \right)^{S_n}$
  - finite nondeterminism - $TX = F^+ X$
  - 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 $J : \aleph_1^{\text{op}} \rightarrow \mathcal{L}$
  - (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 $op : O \rightarrow I$
    - (equivalently a family of operations $opi_{i \in I} : O \rightarrow 1$)
  - 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 \(M : \mathcal{L} \to \mathcal{C}\)

- The models of \(\mathcal{L}\) together with nat. transfs.
  - form a category \(\text{Mod}(\mathcal{L}, \mathcal{C})\) with \(U : \text{Mod}(\mathcal{L}, \mathcal{C}) \to \mathcal{C}\)
  - having a left adjoint \(F : \mathcal{C} \to \text{Mod}(\mathcal{L}, \mathcal{C})\)
  - the adjoint functors induce a monad \(T = UF\)

- For the purposes of this talk, we let \(\mathcal{C} = \text{Set}\)
  - To give a model \(M\) of \(\mathcal{L}\) is equivalent to
    - giving a set \(X = M1\)
    - for every operation \(op : O \to I\) a morphism \(X^O \to X^I\)
  - 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 \(M : \mathcal{L} \to \mathcal{C}\)

- The models of \(\mathcal{L}\) together with nat. transfs. :
  - form a category \(\text{Mod}(\mathcal{L}, \mathcal{C})\) with \(U : \text{Mod}(\mathcal{L}, \mathcal{C}) \to \mathcal{C}\)
  - having a left adjoint \(F : \mathcal{C} \to \text{Mod}(\mathcal{L}, \mathcal{C})\)
  - the adjoint functors induce a monad \(T = UF\)

- For the purposes of this talk, we let \(\mathcal{C} = \text{Set}\)

- To give a model \(M\) of \(\mathcal{L}\) is equivalent to
  - giving a set \(X = M1\)
  - for every operation \(op : O \to I\) a morphism \(X^O \to X^I\)

- Because
  - \(M1\) determines \(MO\) up to coherent isomorphism
  - \(MO \cong M\left(\prod_{o \in O} 1\right) \cong \prod_{o \in O} (M1) \cong (M1)^O\)
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** $\text{Loc}$
- Take the **set of states** to be $S = V^{\text{Loc}}$

- The theory is freely generated by operations
  - $\text{lookup} : V \rightarrow \text{Loc}$
  - $\text{update} : 1 \rightarrow \text{Loc} \times V$

subject to commuting diagrams expressed set-theoretically

1. $\text{lookup}_{\text{loc}}(\text{update}_{\text{loc}, v}(x))_v = x$
2. $\text{lookup}_{\text{loc}}(\text{lookup}_{\text{loc}}(x_{vv'})_v)_v' = \text{lookup}_{\text{loc}}(x_{vv})_v$
3. $\text{update}_{\text{loc}, v}(\text{update}_{\text{loc}, v'}(x)) = \text{update}_{\text{loc}, v'}(x)$
4. $\text{update}_{\text{loc}, v}(\text{read}_{\text{loc}}(x'_v)_v) = \text{update}_{\text{loc}, v}(x_v)$
5. $\text{update}_{\text{loc}, v}(\text{update}_{\text{loc}'}, v'(x)) = \text{update}_{\text{loc}', v'}(\text{update}_{\text{loc}, v}(x))$ \quad (\text{loc} \neq \text{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
  - \( lookup : V \rightarrow Loc \)
  - \( update : 1 \rightarrow Loc \times V \)

subject to **commuting diagrams** expressed set-theoretically

1. \( lookup_{loc}(update_{loc,v}(x))_v = x \)
2. \( lookup_{loc}(lookup_{loc}(x_{vv'}))_{v'} = lookup_{loc}(x_{vv})_v \)
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. \( update_{loc,v}(update_{loc',v'}(x)) = update_{loc',v'}(update_{loc,v}(x)) \) \( (loc \neq loc') \)
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**
  - $lookup : V \rightarrow Loc$
  - $update : 1 \rightarrow Loc \times V$

subject to **commuting diagrams** expressed set-theoretically

1. $lookup_{loc}(update_{loc,v}(x))_v = x$
2. $lookup_{loc}(lookup_{loc}(x_{vv'}))_{v'} = lookup_{loc}(x_{vv})_v$
3. $update_{loc,v}(update_{loc,v'}(x)) = update_{loc,v'}(x)$
4. $update_{loc,v}(read_{loc}(x'_{v'})) = update_{loc,v}(x_v)$
5. $update_{loc,v}(update_{loc',v'}(x)) = update_{loc',v'}(update_{loc,v}(x))$ (for $loc \neq loc'$)
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
  - $lookup : V \rightarrow Loc$
  - $update : 1 \rightarrow Loc \times V$

subject to **commuting diagrams** expressed set-theoretically

1. $lookup_{loc}(update_{loc,v}(x))_v = x$
2. $lookup_{loc}(lookup_{loc_{v'}}(x_{vv'}))_{v'} = lookup_{loc_{v'}}(x_{vv'})_v$
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. $update_{loc,v}(update_{loc_{v'},v'}(x)) = update_{loc_{v'},v'}(update_{loc,v}(x))$ \quad (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
  - $\text{lookup} : V \rightarrow Loc$
  - $\text{update} : 1 \rightarrow Loc \times V$

subject to commuting diagrams expressed set-theoretically

1. $\text{lookup}_{loc}(\text{update}_{loc,v}(x))_v = x$
2. $\text{lookup}_{loc}(\text{lookup}_{loc}(x_{vv'})_v)_v = \text{lookup}_{loc}(x_{vv})_v$
3. $\text{update}_{loc,v}(\text{update}_{loc,v'}(x)) = \text{update}_{loc,v'}(x)$
4. $\text{update}_{loc,v}(\text{read}_{loc}(x'_v)_v) = \text{update}_{loc,v}(x_v)$
5. $\text{update}_{loc,v}(\text{update}_{loc',v'}(x)) = \text{update}_{loc',v'}(\text{update}_{loc,v}(x))$ (loc $\neq$ loc')
6. ...
Small detour into local state

- \((TX)_n = \left( \int_{m \in \{n/\text{Inj}\}} (S_m \times X_m) \right)^{S_n}\)

- Monad and algebra are given in category \(\text{Set}^{\text{Inj}}\)
  - \((\text{Inj} \text{ is the category of finite sets and injections})\)

- \(L_n = \text{Inj}(1, n), \quad V_n = V, \quad S_n = V^n\)

- The algebra is given by
  - \text{lookup} : X^V \rightarrow X^\text{Loc}
  - \text{update} : X \rightarrow X^{\text{Loc} \times V}
  - \text{block} : [L, X] \rightarrow X^V
  - subject to appropriate diagrams commuting

- \((Y^X)_n = [\text{Inj, Set}](X - \times \text{Inj}(n, -), Y -)\)
- \([X, Y]_n = [\text{Inj, Set}](X-, Y(n + -))\)

- See also work by Power (cotensoring models with comodels) and Staton (completeness via nominal sets)
Small detour into local state

• \((TX)_n = \left( \int_{m \in (n/\text{Inj})} (S_m \times X_m) \right)^{S_n}\)

• Monad and algebra are given in category \(\text{Set}^{\text{Inj}}\)
  • \((\text{Inj} \text{ is the category of finite sets and injections})\)

• \(L_n = \text{Inj}(1, n), \quad V_n = V, \quad S_n = V^n\)

• The algebra is given by
  • \(\text{lookup} : X^V \to X^{Loc}\)
  • \(\text{update} : X \to X^{Loc \times V}\)
  • \(\text{block} : [L, X] \to X^V\)
  • subject to appropriate diagrams commuting

• \((Y^X)_n = [\text{Inj}, \text{Set}] (X - \times \text{Inj}(n, -), Y -)\)

• \([X, Y]_n = [\text{Inj}, \text{Set}] (X - , Y(n + -))\)

• See also work by Power (cotensoring models with comodels) and Staton (completeness via nominal sets)
Small detour into local state

\[
(TX)_n = \left( \bigint_{m \in (n/Inj)} (S_m \times X_m) \right)^{S_n}
\]

- Monad and algebra are given in category \( \text{Set}^{\text{Inj}} \)
  - \((\text{Inj} \text{ is the category of finite sets and injections})\)
- \(L_n = Inj(1, n), \quad V_n = V, \quad S_n = V^n\)
- The algebra is given by
  - \(\text{lookup} : X^V \to X^{\text{Loc}}\)
  - \(\text{update} : X \to X^{\text{Loc} \times V}\)
  - \(\text{block} : [L, X] \to X^V\)
  - subject to appropriate diagrams commuting

\[
(Y^X)_n = [\text{Inj}, \text{Set}](X \to \times \text{Inj}(n, -), Y -)
\]
\[
[X, Y]_n = [\text{Inj}, \text{Set}](X -, Y(n + -))
\]

- 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 \(\text{Set}^{Inj}\)
  - \((\text{Inj} \text{ is the category of finite sets and injections})\)

- \(L_n = Inj(1, n), \quad V_n = V, \quad S_n = V^n\)

- The algebra is given by
  - \(\text{lookup} : X^V \to X^{Loc}\)
  - \(\text{update} : X \to X^{Loc \times V}\)
  - \(\text{block} : [L, X] \to X^V\)
  - subject to appropriate diagrams commuting

- \((Y^X)_n = [\text{Inj}, \text{Set}](X \times \text{Inj}(n, -), Y -)\)
- \([X, Y]_n = [\text{Inj}, \text{Set}](X-, Y(n + -))\)

- 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, ...

- 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

\[
\begin{align*}
\Gamma \vdash \sigma : \text{Ref}(\sigma) & \quad \Gamma \vdash \phi : \text{Ref}(\sigma) \quad \Gamma, x : \phi \vdash P : \text{wf} \\
\Gamma \vdash \Sigma_{x:\phi} \psi : \text{Ref}(\sigma_1 \times \sigma_2) & \quad \Gamma \vdash \phi : \text{Ref}(\sigma_1) \quad \Gamma, x : \phi \vdash \psi : \text{Ref}(\sigma_2) \\
\Gamma \vdash (x:\phi)P : \text{Ref}(\sigma) & \quad \Gamma \vdash \Pi_{x:\phi} \psi : \text{Ref}(\sigma \rightarrow \tau) \\
\end{align*}
\]

- Examples of typing rules

\[
\begin{align*}
\Gamma \vdash t : \phi \quad \Gamma \vdash P[t/x] & \quad \Gamma \vdash t : (x:\phi)P \\
\Gamma \vdash \lambda x : \phi. t : \Pi_{x:\phi} \psi & \quad \Gamma \vdash t_1 : \Pi_{x:\phi} \psi \quad \Gamma \vdash t_2 : \phi \\
& \quad \Gamma \vdash t_1 t_2 : \psi[t_2/x]
\end{align*}
\]
Refinement types

- Well-formedness of refinement types

\[
\Gamma \vdash \sigma : \text{Ref}(\sigma) \\
\Gamma, x : \phi \vdash P : \text{wf} \\
\Gamma \vdash (x : \phi)P : \text{Ref}(\sigma)
\]

\[
\Gamma \vdash \phi : \text{Ref}(\sigma_1) \quad \Gamma, x : \phi \vdash \psi : \text{Ref}(\sigma_2) \\
\Gamma \vdash \Sigma_{x : \phi} \psi : \text{Ref}(\sigma_1 \times \sigma_2)
\]

\[
\Gamma \vdash \phi : \text{Ref}(\sigma) \quad \Gamma, x : \phi \vdash \psi : \text{Ref}(\tau) \\
\Gamma \vdash \Pi_{x : \phi} \psi : \text{Ref}(\sigma \to \tau)
\]

- Examples of typing rules

\[
\Gamma \vdash t : \phi \quad \Gamma \vdash P[t/x] \\
\Gamma \vdash t : (x : \phi)P
\]

\[
\Gamma, x : \phi \vdash t : \psi \\
\Gamma \vdash \lambda x : \phi . t : \Pi_{x : \phi} \psi
\]

\[
\Gamma \vdash t_1 : \Pi_{x : \phi} \psi \quad \Gamma \vdash t_2 : \phi \\
\Gamma \vdash t_1 t_2 : \psi[t_2/x]
\]
Refinement types

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

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

\[ \begin{array}{c}
\mathcal{P} \\
\downarrow
\end{array} \quad \begin{array}{c}
\mathcal{T} \\
\downarrow
\end{array} \quad \begin{array}{c}
\mathcal{C} \\
\text{cod}
\end{array} \]
Refinement types

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

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

\[
\begin{array}{ccc}
\mathcal{P} & \rightarrow & \mathbb{T} \\
\downarrow & & \downarrow \\
\mathcal{C} & \rightarrow & \mathcal{C}
\end{array}
\]

\text{cod}
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
  - $lookup : X^V \rightarrow X^{Open(Loc)}$
  - $update : X \rightarrow X^{Open(Loc) \times V}$

- 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
  
  - $lookup : X^V \rightarrow X^{Open(Loc)}$
  - $update : X \rightarrow X^{Open(Loc) \times V}$

• However, notice that this requires an a priori given collection of open locations
Refining global state

- We had the finite set of locations \( \text{Loc} \)

- Assume that we now have predicates \( \text{Open}(\text{Loc}) \) and \( \text{Closed}(\text{Loc}) = \neg \text{Open}(\text{loc}) \) on the locations \( \text{Loc} \)

- Conceptually they denote subsets of \( \text{Loc} \)

- We should only be able to read from and write to locations that are open
  - \( \text{lookup} : \mathcal{X}^\mathcal{V} \rightarrow \mathcal{X}^{\text{Open}(\text{Loc})} \)
  - \( \text{update} : \mathcal{X} \rightarrow \mathcal{X}^{\text{Open}(\text{Loc}) \times \mathcal{V}} \)

- 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
  - $\text{lookup} : X^V \rightarrow X^{\text{Open}(\text{Loc})}$
  - $\text{update} : X \rightarrow X^{\text{Open}(\text{Loc}) \times V}$
  - $\text{open} : X \rightarrow X^{\text{Closed}(\text{Loc})}$
  - $\text{close} : X \rightarrow X^{\text{Open}(\text{Loc})}$

- 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 $\text{Set}^W$

- 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
  - \( \text{lookup} : X^V \rightarrow X^{\text{Open}(\text{Loc})} \)
  - \( \text{update} : X \rightarrow X^{\text{Open}(\text{Loc}) \times V} \)
  - \( \text{open} : X \rightarrow X^{\text{Closed}(\text{Loc})} \)
  - \( \text{close} : X \rightarrow X^{\text{Open}(\text{Loc})} \)

- 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 \( \text{Set}^W \)
  - 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
  - $\text{lookup} : X^V \rightarrow X^{\text{Open}(\text{Loc})}$
  - $\text{update} : X \rightarrow X^{\text{Open}(\text{Loc}) \times V}$
  - $\text{open} : X \rightarrow X^{\text{Closed}(\text{Loc})}$
  - $\text{close} : X \rightarrow X^{\text{Open}(\text{Loc})}$

- 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 $\text{Set}^W$

- 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
  - \( \text{lookup} : X^V \rightarrow X^{\text{Open}(\text{Loc})} \)
  - \( \text{update} : X \rightarrow X^{\text{Open}(\text{Loc}) \times V} \)
  - \( \text{open} : X \rightarrow X^{\text{Closed}(\text{Loc})} \)
  - \( \text{close} : X \rightarrow X^{\text{Open}(\text{Loc})} \)

- 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 \( \text{Set}^W \)

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

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

• The forgetful functor $U : \text{Mod}(\mathcal{L}, \text{Set}) \to \text{Set}^W$ 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 $J : W^\ast \rightarrow \mathcal{L}$
  - *(where $W^\ast$ has as objects words $w_0, ..., w_n$ over $W$)*

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

- The forgetful functor $U : \text{Mod}(\mathcal{L}, \text{Set}) \rightarrow \text{Set}^W$ 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 \( L \) with countable products
  • an id. on objects countable-product preserving functor
    \( J : W^* \longrightarrow L \)
  • (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 \( M : L \longrightarrow \text{Set} \)

• The forgetful functor \( U : \text{Mod}(L, \text{Set}) \longrightarrow \text{Set}^W \) 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 $J : W^* \to \mathcal{L}$
  • (where $W^*$ has as objects words $w_0, ..., w_n$ over $W$)

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

• The forgetful functor $U : \text{Mod}(\mathcal{L}, \text{Set}) \to \text{Set}^W$ again has a left adjoint $F$ inducing a monad $T = UF$
Refining global state (W-sorted theories)

- Let the worlds be $W = \text{Bool}^W$

- We have families of operations in the theory
  - $\text{lookup}_{w \in W, \text{loc} \in \text{Open}_w(\text{Loc})} : w, ..., w \rightarrow w$
  - $\text{update}_{w \in W, \text{loc} \in \text{Open}_w(\text{Loc}), v \in V} : w \rightarrow w$
  - $\text{open}_{w \in W, \text{loc} \in \text{Open}_w(\text{Loc})} : w \rightarrow w[\text{loc} \mapsto \bot]$
  - $\text{close}_{w \in W, \text{loc} \in \text{Closed}_w(\text{Loc})} : w \rightarrow w[\text{loc} \mapsto \top]$
  - satisfying appropriate commuting diagrams

- Giving us the algebra
  - $\text{lookup}_{w \in W, \text{loc} \in \text{Open}_w(\text{Loc})} : (X^V)_w \rightarrow X_w$
  - $\text{update}_{w \in W, \text{loc} \in \text{Open}_w(\text{Loc}), v \in V} : X_w \rightarrow X_w$
  - $\text{open}_{w \in W, \text{loc} \in \text{Open}_w(\text{Loc})} : X_w \rightarrow X_w[\text{loc} \mapsto \bot]$
  - $\text{close}_{w \in W, \text{loc} \in \text{Closed}_w(\text{Loc})} : X_w \rightarrow X_w[\text{loc} \mapsto \top]$
Refining global state (W-sorted theories)

- Let the worlds be $W = \text{Bool}^W$

- We have families of operations in the theory
  - $\text{lookup}_{w \in W, loc \in \text{Open}_w(Loc)} : w, \ldots, w \rightarrow w$
  - $\text{update}_{w \in W, loc \in \text{Open}_w(Loc), v \in V} : w \rightarrow w$
  - $\text{open}_{w \in W, loc \in \text{Open}_w(Loc)} : w \rightarrow w[loc \mapsto \bot]$
  - $\text{close}_{w \in W, loc \in \text{Closed}_w(Loc)} : w \rightarrow w[loc \mapsto \top]$
  - satisfying appropriate commuting diagrams

- Giving us the algebra
  - $\text{lookup}_{w \in W, loc \in \text{Open}_w(Loc)} : (X^V)_w \rightarrow X_w$
  - $\text{update}_{w \in W, loc \in \text{Open}_w(Loc), v \in V} : X_w \rightarrow X_w$
  - $\text{open}_{w \in W, loc \in \text{Open}_w(Loc)} : X_w \rightarrow X_w[loc \mapsto \bot]$
  - $\text{close}_{w \in W, loc \in \text{Closed}_w(Loc)} : X_w \rightarrow X_w[loc \mapsto \top]$
Refining global state (*W*-sorted theories)

- So we have the algebra
  - $\text{lookup}_{w \in W, loc \in \text{Open}_w(\text{Loc})} : (X^V)_w \rightarrow X_w$
  - $\text{update}_{w \in W, loc \in \text{Open}_w(\text{Loc}), v \in V} : X_w \rightarrow X_w$
  - $\text{open}_{w \in W, loc \in \text{Open}_w(\text{Loc})} : X_w \rightarrow X_w[\text{loc} \mapsto \bot]$
  - $\text{close}_{w \in W, loc \in \text{Close}_w(\text{Loc})} : X_w \rightarrow X_w[\text{loc} \mapsto \top]$

- Inducing monad $TX_w = UFX_w = (\sum_{w' \in W} (S_{w'} \times X_{w'}))^{S_w}$

- With the unit $\eta_x : X \rightarrow UFX$ of the adjunction given by:
  $\eta_{x,w} \gamma = \lambda s . \text{inj}_w (s, \gamma)$

- And the counit $\varepsilon_A : FUA \rightarrow A$ of the adjunction:
  $\varepsilon_{A,w} = (\prod (S \times A_{w'}))^{S} \rightarrow (\prod (S \times \text{close}))^{S} \rightarrow (\prod (S \times A_{w \top}))^{S} \rightarrow (S \times A_{w \top})^{S} \rightarrow (A_{w \top})^{S} \rightarrow A_{w \top} \rightarrow \text{open} \rightarrow A_w$

- And the Kleisli extension is given by $(\_)^* = U\varepsilon F$
Refining global state ($W$-sorted theories)

- So we have the algebra
  - $\text{lookup}_{w \in W, \text{loc} \in \text{Open}_w(\text{Loc})} : (X^V)_w \to X_w$
  - $\text{update}_{w \in W, \text{loc} \in \text{Open}_w(\text{Loc}), v \in V} : X_w \to X_w$
  - $\text{open}_{w \in W, \text{loc} \in \text{Open}_w(\text{Loc})} : X_w \to X_w[\text{loc} \mapsto \bot]$
  - $\text{close}_{w \in W, \text{loc} \in \text{Closed}_w(\text{Loc})} : X_w \to X_w[\text{loc} \mapsto T]$

- Inducing monad $TX_w = UFX_w = (\sum_{w' \in W} (S_{w'} \times X_{w'}))^{S_w}$

- With the unit $\eta_x : X \to UFX$ of the adjunction given by:
  $\eta_{x,w} \gamma = \lambda s . \text{inj}_w (s, \gamma)$

- And the counit $\varepsilon_A : FUA \to A$ of the adjunction:
  $\varepsilon_{A,w} = (\prod (S \times A_w))^{S} \xrightarrow{(\prod (S \times \text{close}))^{S}} (\prod (S \times A_{wT}))^{S} \xrightarrow{=} (S \times A_{wT})^{S} \xrightarrow{\text{write}} (A_{wT})^{S} \xrightarrow{\text{read}} A_{wT} \xrightarrow{\text{open}} A_w$

- And the Kleisli extension is given by $(\_)^* = U\varepsilon F$
Refining global state (W-sorted theories)

- So we have the algebra
  - \( \text{lookup}_{w \in W, loc \in \text{Open}_w(Loc)} : (X^V)_w \rightarrow X_w \)
  - \( \text{update}_{w \in W, loc \in \text{Open}_w(Loc), v \in V} : X_w \rightarrow X_w \)
  - \( \text{open}_{w \in W, loc \in \text{Open}_w(Loc)} : X_w \rightarrow X_w[\text{loc} \mapsto \bot] \)
  - \( \text{close}_{w \in W, loc \in \text{Closed}_w(Loc)} : X_w \rightarrow X_w[\text{loc} \mapsto \top] \)

- Inducing monad \( TX_w = UFX_w = (\sum_{w' \in W} (S_{w'} \times X_{w'}))^{S_w} \)

- With the unit \( \eta_x : X \rightarrow UFX \) of the adjunction given by:
  \[
  \eta_{x, w} \gamma = \lambda s . \text{inj}_w (s, \gamma)
  \]

- And the counit \( \varepsilon_A : FUA \rightarrow A \) of the adjunction:
  \[
  \varepsilon_{A, w} = \left( \prod (S \times A_{w'}) \right)^{S} \xrightarrow{\text{(write)}} \left( \prod (S \times \text{close}) \right)^{S} \xrightarrow{\text{(write)}} (S \times A_{w'})^{S} \xrightarrow{\text{read}} A_{w'} \xrightarrow{\text{open}} A_w
  \]

- 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 $W = \text{Bool}$

• We are interested in axiomatizing logging in to and logging off from some system

• We have the theory
  • $\text{LogIn}_{p \in \text{Password}} : \text{true, false} \rightarrow \text{false}$
  • $\text{DoSomething} : \text{true} \rightarrow \text{true}$
  • $\text{LogOut} : \text{false} \rightarrow \text{true}$

• And the algebra
  • $\text{LogIn}_{p \in \text{Password}} : X_{\text{true}} \times X_{\text{false}} \rightarrow X_{\text{false}}$
  • $\text{DoSomething} : X_{\text{true}} \rightarrow X_{\text{true}}$
  • $\text{LogOut} : X_{\text{false}} \rightarrow X_{\text{true}}$

• However, $\text{LogIn}$ 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 $W = \text{Bool}$
- We are interested in axiomatizing logging in to and logging off from some system
- We have the theory
  - $\text{LogIn}_{p \in \text{Password}} : \text{true, false} \rightarrow \text{false}$
  - $\text{DoSomething} : \text{true} \rightarrow \text{true}$
  - $\text{LogOut} : \text{false} \rightarrow \text{true}$
- And the algebra
  - $\text{LogIn}_{p \in \text{Password}} : X_{\text{true}} \times X_{\text{false}} \rightarrow X_{\text{false}}$
  - $\text{DoSomething} : X_{\text{true}} \rightarrow X_{\text{true}}$
  - $\text{LogOut} : X_{\text{false}} \rightarrow X_{\text{true}}$
- However, $\text{LogIn}$ 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
  
  $$\text{op}_w : \prod_{o \in O_w} X_{\delta_o(w, o)} \rightarrow \prod_{i \in I_w} X_{\delta_i(w, i)}$$

  and in the theory
  
  $$\text{op}_w : \bigsqcup_{o \in O_w} \{\delta_o(w, o)\} \rightarrow \bigsqcup_{i \in I_w} \{\delta_i(w, i)\}$$

- But can’t always define them uniformly in $w$, e.g.:
  
  $$\text{lookup}_{[l_i \mapsto \bot]} : \bigsqcup_{v \in V} \{[l_i \mapsto \bot]\} \rightarrow 0$$

- 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
  
  
  \[
  \text{op}_w : \prod_{o \in O_w} X_{\delta_o(w,o)} \longrightarrow \prod_{i \in I_w} X_{\delta_i(w,i)}
  \]

  and in the theory
  
  \[
  \text{op}_w : \coprod_{o \in O_w} \{\delta_o(w,o)\} \longrightarrow \coprod_{i \in I_w} \{\delta_i(w,i)\}
  \]

- But can’t always define them uniformly in $w$, e.g.:

  \[
  \text{lookup}_{[i \mapsto \bot]} : \coprod_{v \in V} \{[[i \mapsto \bot]]\} \longrightarrow 0
  \]

- 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
  - $\text{op}_w : \prod_{o \in O_w} X_{\delta_o(w, o)} \to \prod_{i \in I_w} X_{\delta_i(w, i)}$
  and in the theory
  - $\text{op}_w : \coprod_{o \in O_w} \{\delta_o(w, o)\} \to \coprod_{i \in I_w} \{\delta_i(w, i)\}$
- But can’t always define them uniformly in $w$, e.g.:
  - $\text{lookup}_{[l_i \mapsto \bot]} : \coprod_{v \in V} \{[l_i \mapsto \bot]\} \to 0$
- Seems to be kind of inherent to the idea that not all operations should be definable in all worlds
Questions?