Algebras for Parameterised Monads

Robert Atkey
School of Informatics, University of Edinburgh

7th September 2009

Outline

- A language with typed effects
- Parameterised monads
- Algebras for parameterised monads
- Relating parameterised monads to monads

Typed Effects

Type and Effect Systems classify programs according to the effects they may perform.

For example:

$$\Gamma \vdash M : \tau; \{\mathsf{r}(l_1), \mathsf{r}(l_2), \mathsf{w}(l_1)\}$$

M will, at most, read from l_1 and l_2 and write to l_1.

A Language with Typed Effects

Assume a set $\mathcal{A}$ of *permissions*. E.g. $\mathcal{A} = \mathcal{P}(\{\mathsf{r}(l), \mathsf{w}(l) \mid l \in \mathit{Loc}\})$

Two typing judgments:

$$\Gamma \vdash V : \tau$$

$$\Gamma; a \vdash M : \tau; b$$

$$\tau, \sigma ::= \texttt{int} \mid \texttt{bool} \mid \texttt{unit} \mid \tau \times \sigma \mid (\tau; a) \to (\sigma; b)$$

Example: Reading and Writing

$$\mathcal{A} = \mathcal{P}_{fin}(\{\mathsf{r}(l), \mathsf{w}(l) \mid l \in Loc\})$$

$$-;\{\mathsf{r}(l), \mathsf{w}(l')\} \vdash \texttt{let } x \Leftarrow \texttt{read}(l) \texttt{ in } \texttt{write}(l', x) : \texttt{unit};\{\mathsf{r}(l), \mathsf{w}(l')\}$$

Example: Linear Permissions

$$\mathcal{A} = \{p_1 \otimes \ldots \otimes p_n \mid p_i \in \{\mathsf{open}_d, \mathsf{closed}_d\}, d \in \mathit{Device}\}$$

$$\begin{array}{l} -; \mathsf{closed}_d \otimes \mathsf{open}_{d'} \vdash \\ \quad \mathtt{open}(d);\ \mathtt{let}\ x \Leftarrow \mathtt{input}(d')\ \mathtt{in}\ \mathtt{output}(d, x);\ \mathtt{close}(d) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad : \mathtt{unit}; \mathsf{closed}_d \otimes \mathsf{open}_{d'} \end{array}$$

Example: Session Types

$$\mathcal{A} \ni S \quad ::= \quad !\tau \mid ?\tau \mid S_1 + S_2 \mid S_1.S_2 \mid \circ$$

$$\begin{array}{l} -;?\texttt{int}.(!\texttt{bool} + \circ) \vdash \\ \qquad \texttt{let } x \Leftarrow \texttt{input in} \\ \qquad \texttt{if } x = 0 \texttt{ then output}(\texttt{true}) \texttt{ else val } () \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad : \texttt{unit}; \circ \end{array}$$

Consequence

Often, we have permissions a, a' such that every effect a' permits, a permits. Write this as $a \Rightarrow a'$.

In the type system:

$$\frac{a \Rightarrow a' \qquad \Gamma; a' \vdash M : \tau; b' \qquad b' \Rightarrow b}{\Gamma; a \vdash M : \tau; b}$$

Sequencing

The obvious way to sequence M and N is with the rule:

$$\frac{\Gamma; a \vdash M : \tau; b \qquad \Gamma, x : \tau; b \vdash N : \sigma; c}{\Gamma; a \vdash \texttt{let}\ x \Rightarrow M\ \texttt{in}\ N : \sigma; c}$$

But N may require extra permissions over those bequeathed to it by M:

$$\Gamma; b \bullet r \vdash N : c$$

So we sequence by:

$$\frac{\Gamma; a \vdash M : \tau; b \qquad \Gamma, x : \tau; b \bullet r \vdash N : \sigma; c}{\Gamma; a \bullet r \vdash \texttt{let}\ x \Leftarrow M\ \texttt{in}\ N : \sigma; c}$$

The operation $- \bullet r$ places permissions ``in a context".

For reading and writing, $\varepsilon \bullet \varepsilon' = \varepsilon \cup \varepsilon'$

Primitive Operations

$$\frac{\Gamma \vdash V : \tau \qquad \mathsf{op} : (\tau; a) \to (\sigma; b)}{\Gamma; a \vdash \mathsf{op}\ V : \sigma; b}$$

For a collection Ω of operations $\mathsf{op} : (\tau; a) \to (\sigma; b)$. Types τ, σ must be ground.

Example: Reading and writing

- $\varepsilon, \varepsilon' \in \mathcal{A} = \mathcal{P}(\{\mathsf{r}(l), \mathsf{w}(l) \mid l \in \mathit{Loc}\})$
- $\varepsilon \Rightarrow \varepsilon'$ iff $\varepsilon' \subseteq \varepsilon$
- $\varepsilon \bullet \varepsilon' = \varepsilon \cup \varepsilon'$
- Operations:
  - $\mathtt{read}_l : (\mathtt{unit}; \{\mathsf{r}(l)\}) \to (\mathtt{int}; \{\mathsf{r}(l)\})$
  - $\mathtt{write}_l : (\mathtt{int}; \{\mathsf{w}(l)\}) \to (\mathtt{unit}; \{\mathsf{w}(l)\})$

Example: Linear permissions

- $P, Q ::= X \mid P \otimes Q \mid I$
- $P \Rightarrow Q$ iff P and Q are the same multiset.
- $P \bullet Q = P \otimes Q$
- Operations (for example):
  - $\mathtt{open}_d : (\mathtt{unit}; \mathsf{closed}_d) \longrightarrow (\mathtt{unit}; \mathsf{open}_d)$
  - $\mathtt{input}_d : (\mathtt{unit}; \mathsf{open}_d) \longrightarrow (\mathtt{int}; \mathsf{open}_d)$
  - $\mathtt{output}_d : (\mathtt{int}; \mathsf{open}_d) \longrightarrow (\mathtt{unit}; \mathsf{open}_d)$
  - $\mathtt{close}_d : (\mathtt{unit}; \mathsf{open}_d) \longrightarrow (\mathtt{unit}; \mathsf{closed}_d)$

Example: Session Types

- $S ::= ?\tau \mid !\tau \mid S_1 + S_2 \mid \circ \mid S_1.S_2$
- $S_1 + S_2 \Rightarrow S_1$ and $S_1 + S_2 \Rightarrow S_2$ and $S.\circ \Leftrightarrow S \Leftrightarrow \circ.S$
- $S_1 \bullet S_2 = S_1.S_2$
- Operations (τ ground):
  - $\texttt{input}_\tau : (\texttt{unit}; ?\tau) \longrightarrow (\tau; \circ)$
  - $\texttt{output}_\tau : (\tau; !\tau) \longrightarrow (\texttt{unit}; \circ)$

Semantics: Parameterised Monads

We give the abstract semantics of the language using parameterised monads.

An $\mathcal{A}$-parameterised monad on *Set* consists of:

- A functor $T : \mathcal{A}^{\mathrm{op}} \times \mathcal{A} \times \mathit{Set} \to \mathit{Set}$
- A transformation $\eta_{aX} : X \to T(a, a, X)$
  - Natural in A, dinatural in a.
- A transformation
  $\mu_{abcXY} : T(a, b, X) \times (X \to T(b, c, Y)) \to T(a, c, Y)$
  - Natural in X, Y, a and c. Dinatural in b.
- Satisfying:

$$\mu_{aab}(\eta_a(x), f) = f(x) \qquad \mu_{abb}(t, \lambda x.\eta_b(x)) = t$$

$$\mu_{acd}(\mu_{abc}(t, f), g) = \mu_{abd}(t, \lambda x.\mu_{bcd}(f(x), g)).$$

Parameterised Monads

If $a \Rightarrow a'$ then we can weaken the pre-permission.

$$T(a', b, X) \to T(a, b, X)$$

If $b \Rightarrow b'$ then we can strengthen the post-permission.

$$T(a, b, X) \to T(a, b', X)$$

Dinaturality means that if we have:

- $T(a, b, X)$;
- $X \to T(b', c, Y)$;
- $b \Rightarrow b'$

then it does not matter it we strengthen the first's post-permission or weaken the second's pre-permission before sequencing.

Semantics

$$[\![\texttt{int}]\!] = \mathbb{Z} \qquad [\![\texttt{bool}]\!] = \mathbb{B} \qquad [\![\texttt{unit}]\!] = \{\star\}$$

$$[\![\tau \times \sigma]\!] = [\![\tau]\!] \times [\![\sigma]\!] \qquad [\![(\tau; a) \to (\sigma; b)]\!] = [\![\tau]\!] \to T(a, b, [\![\sigma]\!]).$$

Value judgements $\Gamma \vdash V : \tau$ are interpreted by $[\![\Gamma]\!] \to [\![\tau]\!]$.
A judgement $\Gamma; a \vdash M : \tau; b$ is interpreted by $[\![\Gamma]\!] \to T(a, b, [\![\tau]\!])$.

$$[\![\texttt{val}_a\ V]\!]\rho = \eta_a([\![V]\!]\rho)$$

$$[\![\texttt{op}\ V]\!]\rho = [\![\texttt{op}]\!]([\![V]\!]\rho)$$

$$\left[\!\!\left[\frac{a \Rightarrow a' \quad \Gamma; a' \vdash M : \tau; b' \quad b' \Rightarrow b}{\Gamma; a \vdash M : \tau; b} \right]\!\!\right] \rho = T(a \Rightarrow a', b' \Rightarrow b, [\![M]\!]\rho)$$

Semantics of Sequencing

To interpret sequencing, we need to lift the $\bullet r$ operation to interpretations of programs.

Require natural transformations

$$(\bullet r) : T(a,b,X) \to T(a \bullet r, b \bullet r, X)$$

that commute with η and μ:

$$\mu_{(a\bullet d)(b\bullet d)(c\bullet d)}(t\bullet d, \lambda x.f(x)\bullet d) = \mu_{abc}(t,f)\bullet d \qquad \eta_a(x)\bullet b = \eta_{a\bullet b}(x)$$

Define:

$$[\![\texttt{let } x \Leftarrow M \texttt{ in } N]\!]\rho = \mu_{(a\bullet r)(b\bullet r)d}(([\![M]\!]\rho)\bullet r, \lambda x.[\![N]\!](\rho,x))$$

Algebras and Monads

Plotkin and Power: derive monads for effects from algebraic theories.

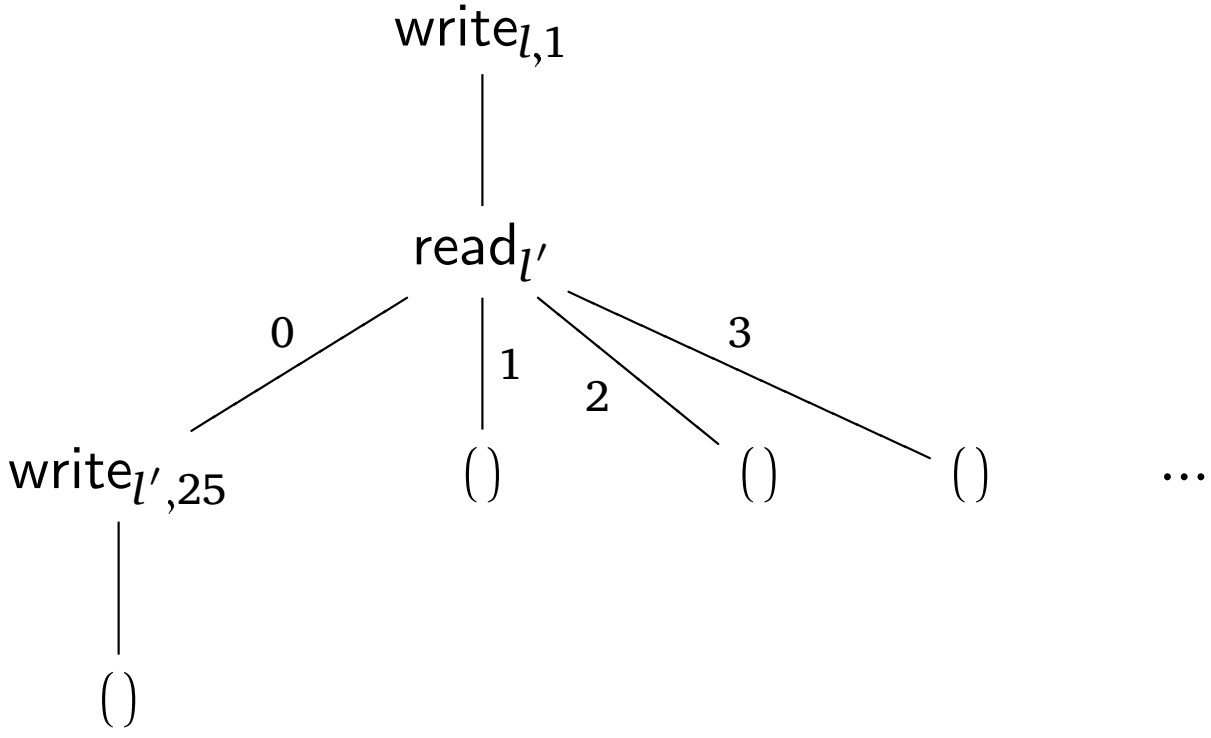
For reading and writing:

- Algebraic operations:
  - read_l of arity $\mathbb{N}$
  - write_{lv} of arity 1
- Equations:
  - $\mathsf{read}_l(\lambda v.\mathsf{read}_l(\lambda v'.t\ v\ v')) = \mathsf{read}_l(\lambda v.t\ v\ v)$
  - Six more...

An operation of arity X takes X-many continuations, one for each possible outcome of performing that effect.

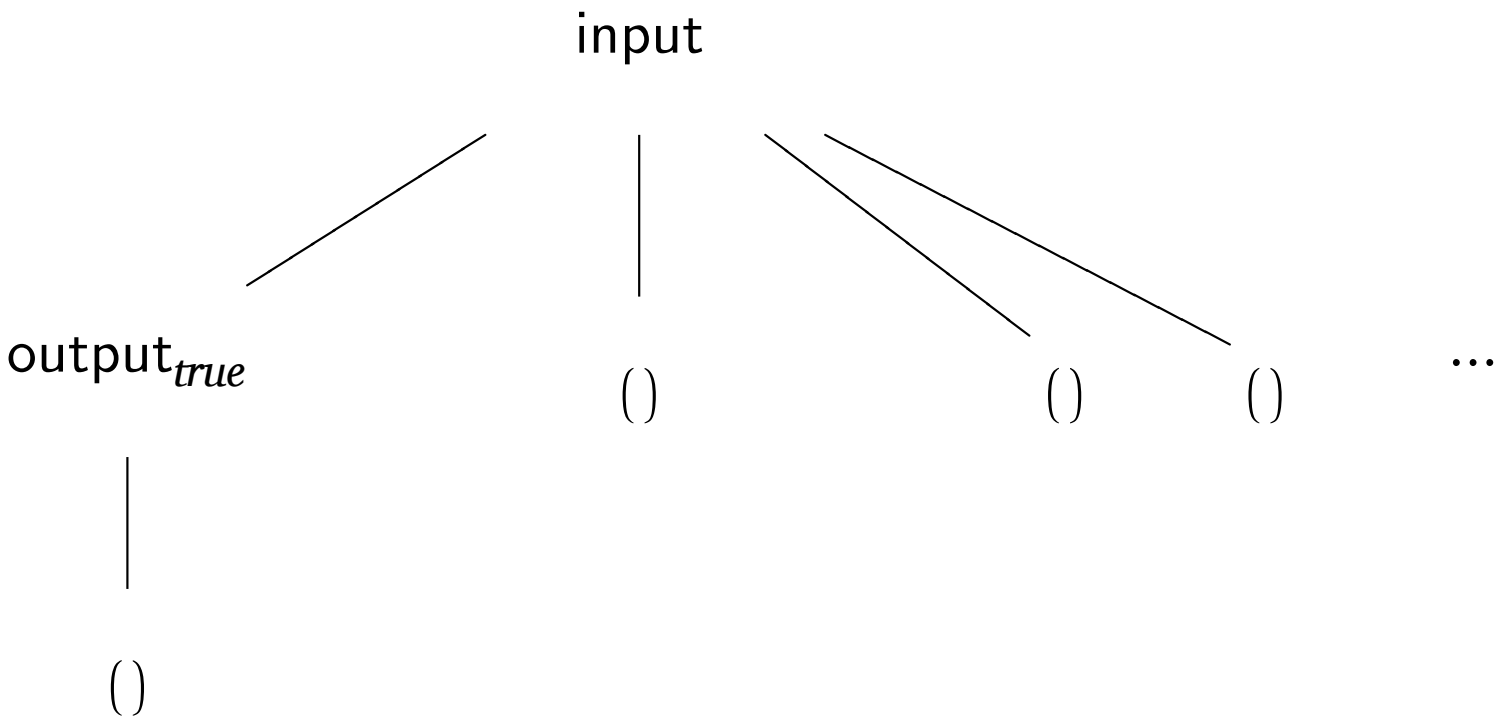
Programs to Terms

$$\texttt{write}(l,1);\ \texttt{let}\ x \Leftarrow \texttt{read}(l')\ \texttt{in}$$
$$\texttt{if}\ x = 0\ \texttt{then}\ \texttt{write}(l',25)\ \texttt{else}\ \texttt{val}\ ()$$



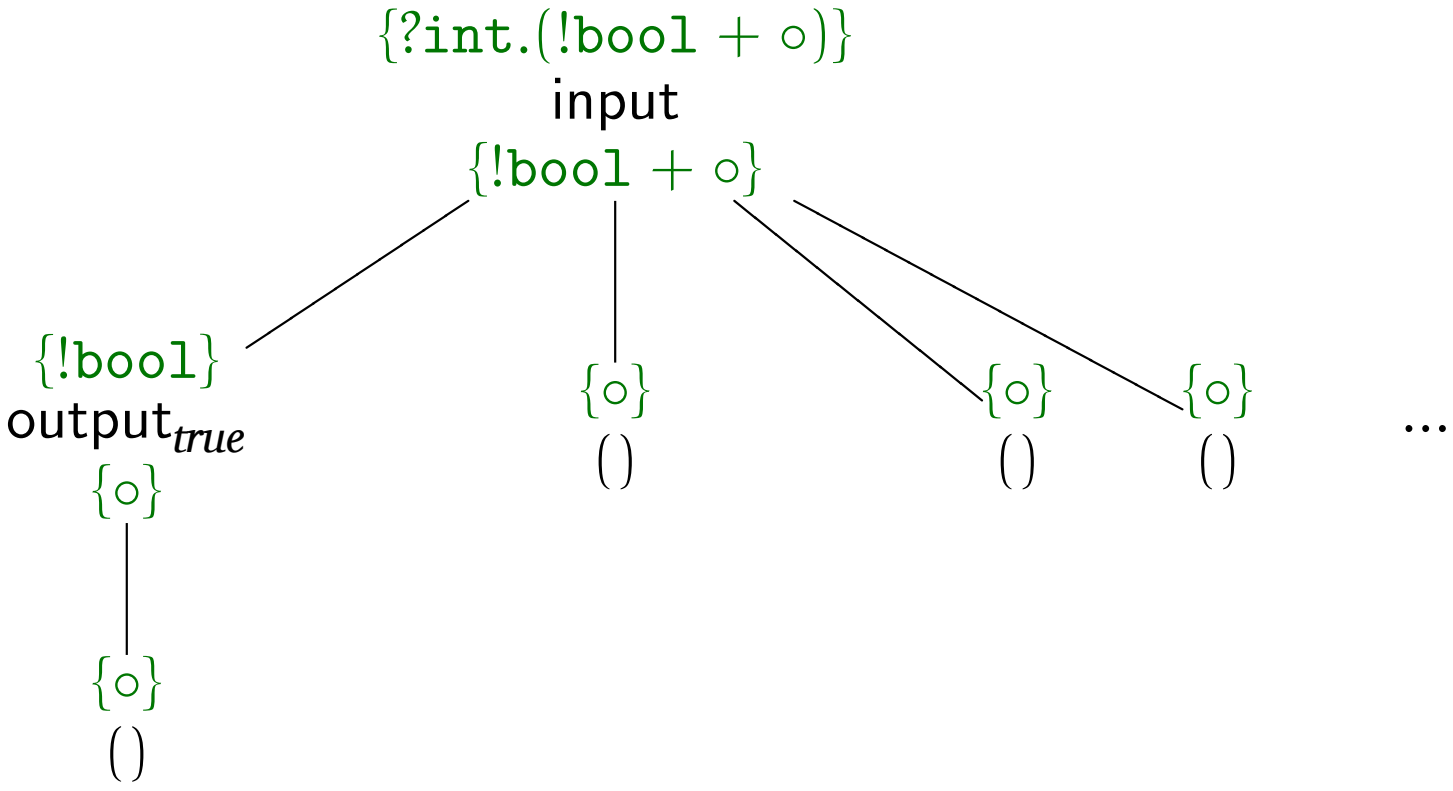
Adding Permissions to Terms

$$\texttt{let } x \Leftarrow \texttt{input in}$$
$$\texttt{if } x = 0 \texttt{ then output}(\texttt{true}) \texttt{ else val } ()$$



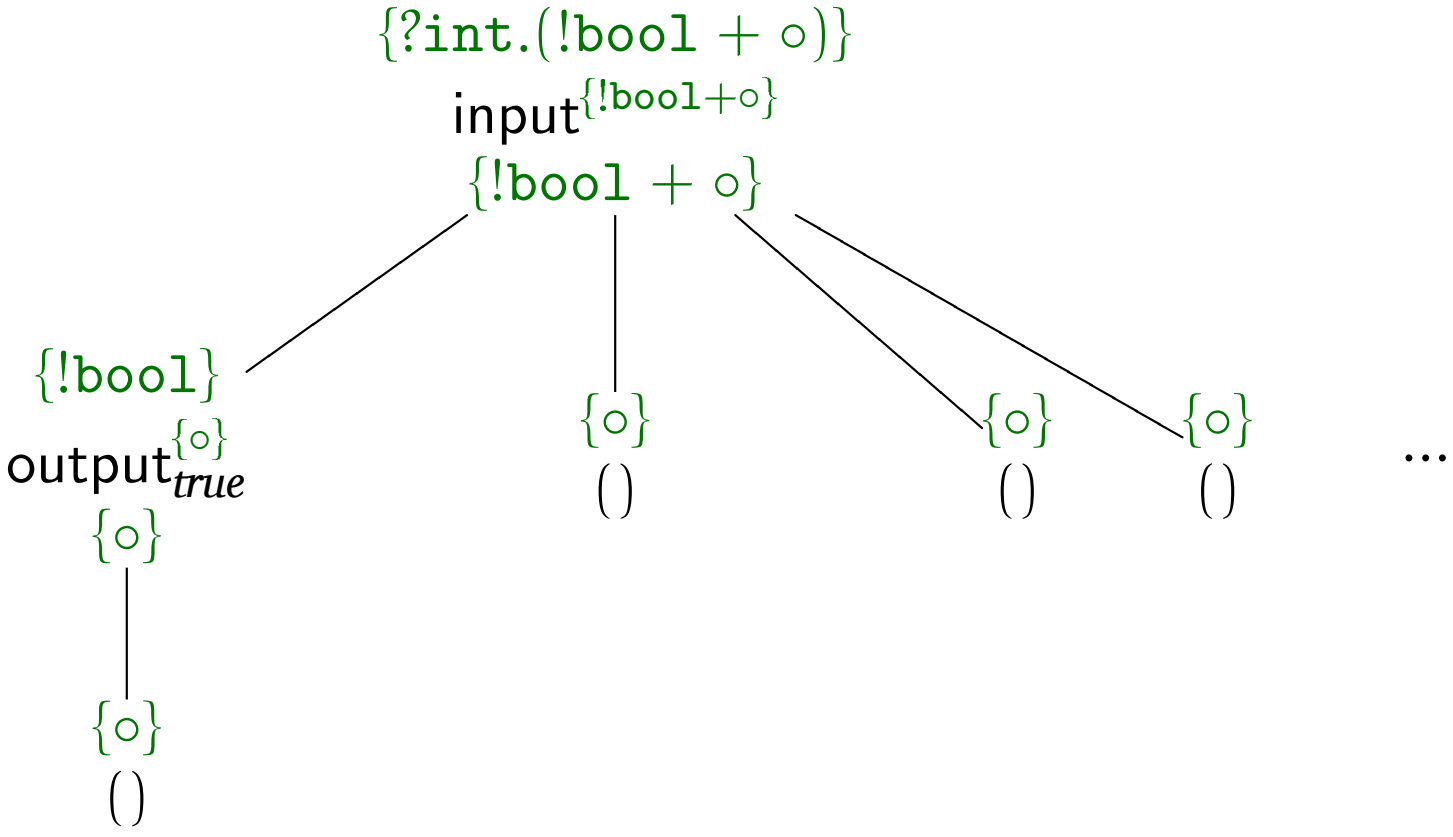
Adding Permissions to Terms

$$\texttt{let } x \Leftarrow \texttt{input in}$$
$$\texttt{if } x = 0 \texttt{ then output}(\texttt{true}) \texttt{ else val } ()$$



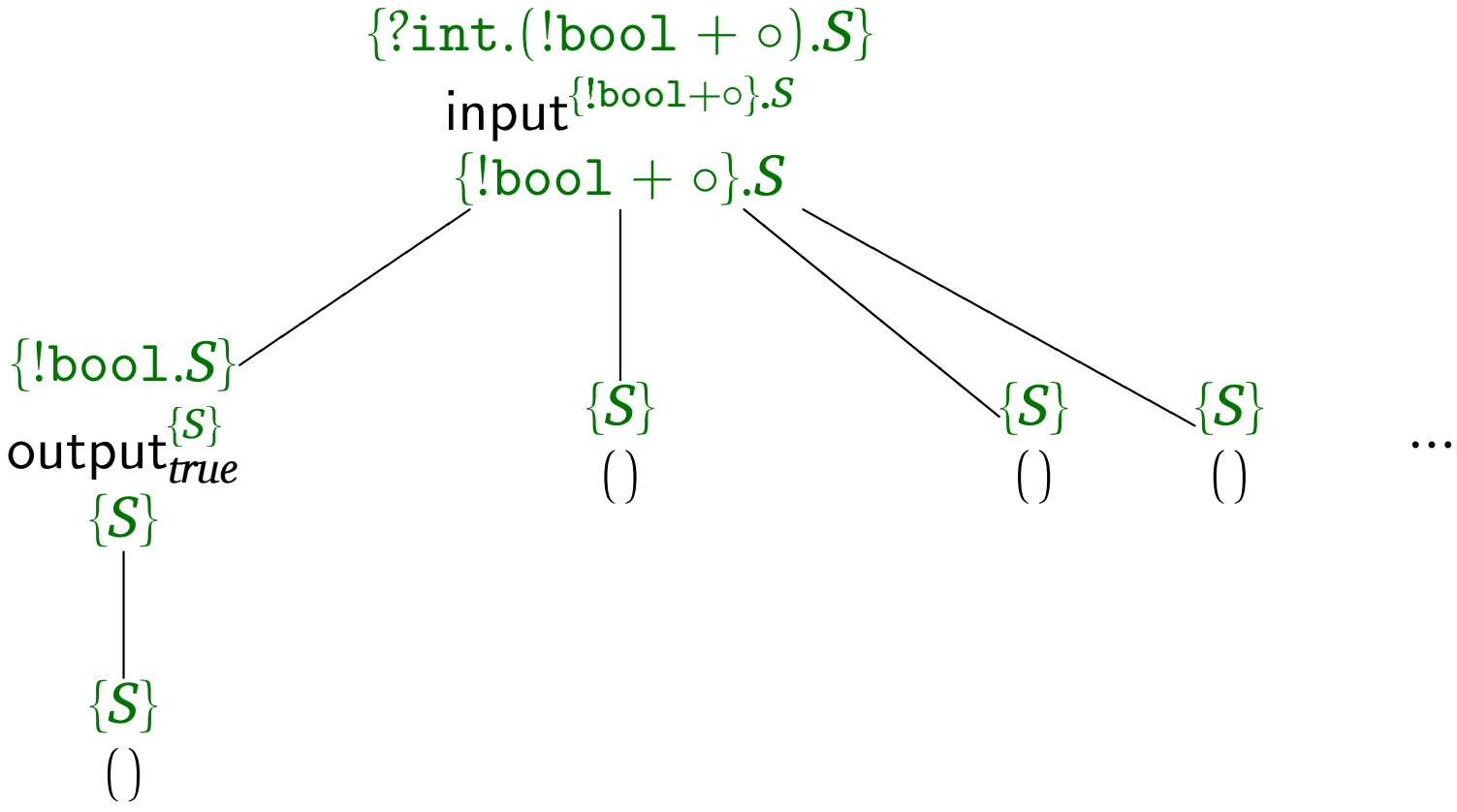
Adding Permissions to Terms

$$\texttt{let } x \Leftarrow \texttt{input in}$$
$$\texttt{if } x = 0 \texttt{ then output(true) else val } ()$$



Adding Permissions to Terms: Lifting

$$\texttt{let } x \Leftarrow \texttt{input in}$$
$$\texttt{if } x = 0 \texttt{ then output(true) else val } ()$$



Parameterised Algebras

An $\mathcal{A}$-signature Σ is a collection of operations

$$\sigma : a \leftarrow b^X$$

where $a, b \in \mathcal{A}$ and X is a countable set.

A Σ-algebra in *Set* consists of:

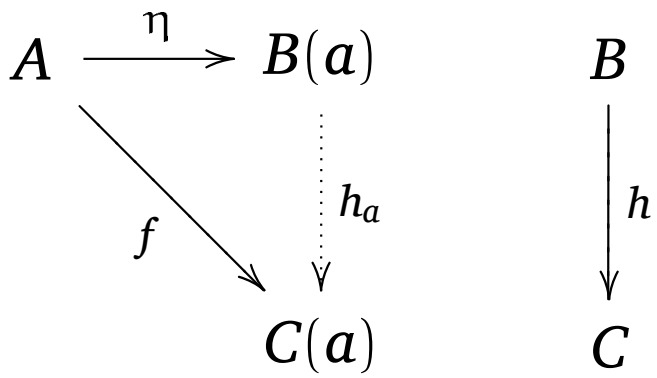
- A functor $A : \mathcal{A}^{\mathrm{op}} \to \mathit{Set}$
- For each $\sigma : a \leftarrow b^X$ a function $\sigma_A : A(b)^X \to A(a)$.

Homomorphisms are natural transformations that preserve the operations.

Idea: $A(a)$ represents computations starting with permissions a.

Free Parameterised Algebras

An algebra $\langle B, \{\sigma_B\}\rangle$ is free for a permission a and set A if there is an arrow $\eta : A \to B(a)$ such that for any algebra $\langle C, \{\sigma_C\}\rangle$ and function $f : A \to C(a)$, there is a unique homomorphism $h : B \to C$ such that $\eta; h_a = f$.



Gives a parameterised monad via adjunctions with parameters.

Constructing Free Parameterised Algebras

We can construct the free parameterised algebra for a signature Σ by the rules:

$$\frac{x \in A}{\mathsf{e}(a \Rightarrow b, x) \in T_\Sigma(a, b, A)}$$

$$\frac{\sigma : a \leftarrow b^X \in \Sigma \qquad \langle t_x \in T_\Sigma(b, b', A) \rangle_{x \in X}}{\mathsf{op}(a' \Rightarrow a, \sigma, \langle t_x \rangle) \in T_\Sigma(a', b', A)}$$

Proposition (Term Algebra is Free)

The sets $T_\Sigma(a, b, A)$ can be made into a functor $T_\Sigma : \mathcal{A}^{\mathsf{op}} \times \mathcal{A} \times \mathit{Set} \to \mathit{Set}$ such that for all a and A, the functor $T_\Sigma(-, a, A)$ is the carrier of an Σ-algebra. Moreover, it is free for a and A.

Equations

Let X be some countably infinite set of variables.

A set of equations $\mathcal{E}$ of equations for Σ consists of pairs:

$$(t, t') \in T_\Sigma(a, b, X) \times T_\Sigma(a, b, X)$$

for some a and b.

A Σ-algebra $\langle A, \{\sigma_A\}\rangle$ satisfies $\mathcal{E}$ if for all $(t, t') \in \mathcal{E}(a, b)$ and functions $f : X \to A(b)$, $\mathsf{eval}(t, f) = \mathsf{eval}(t', f)$ holds.

Given a Σ-algebra $\langle A, \{\sigma_A\}\rangle$, and the congruence $\equiv$ generated from $\mathcal{E}$, then $\langle A/\equiv, \{\sigma_A\}\rangle$ is a $(\Sigma, \mathcal{E})$-algebra.

Congruences respect the operations and functor action of A.

Lifting

To define the lifting operation $(\bullet r)$ on the resulting parameterised monad we need two conditions on theories $(\Sigma, \mathcal{E})$:

- if $\sigma : a \leftarrow b^X \in \Sigma$ then exists $\sigma \bullet r : (a \bullet r) \leftarrow (b \bullet r)^X \in \Sigma$
  - Define:

$$\begin{aligned} c \bullet \mathsf{e}(a \Rightarrow b, x) &= \mathsf{e}(c \bullet a \Rightarrow c \bullet b, x) \\ c \bullet \mathsf{op}(a \Rightarrow b, \sigma, \langle t_x \rangle) &= \mathsf{op}(c \bullet a \Rightarrow c \bullet b, c \bullet \sigma, \langle c \bullet t_x \rangle) \end{aligned}$$

- if $(t, t') \in \mathcal{E}$ then $(t \bullet r, t' \bullet r) \in \mathcal{E}$.

Can close any theory $(\Sigma, \mathcal{E})$ under the conditions above to get a theory $(\Sigma^*, \mathcal{E}^*)$ suitable for the language.

Eilenberg-Moore Category of Algebras

Every parameterised monad has an Eilenberg-Moore category of algebras:

- A functor $A : \mathcal{A}^{\mathsf{op}} \to \mathcal{C}$
- Morphisms $h_{s_1 s_2} : T(s_1, s_2, A(s_2)) \to A(s_1)$

Isomorphic to the category of algebras for a theory by (parameterised extension of) Beck's theorem.

Relating Typed and Untyped Semantics

Algebras constructed in this way contain information about effect types.

Need to relate this to the normal semantics of the program in some monad M that supports all the basic operations and equations.

We relate the typed semantics $[\![-]\!]^t$ to the untyped semantics $[\![-]\!]^u$ in some monad M:

- Define an erasure function $\mathsf{erase} : T(a, b, X) \to MX$
- Show that, for closed programs at type `bool`, then $\mathsf{erase}([\![M]\!]^t) = [\![M]\!]^u$.
- Conclude that, if $[\![M]\!]^t = [\![N]\!]^t$ then for all closed typing derivation contexts $C[-]$ of type `bool`, $[\![C[M]]\!]^u = [\![C[N]]\!]^u$, by compositionality.

Conclusions

- Parameterised Monads can be used to model some effect systems
- Algebraic technique extends to type and effect systems
- Can be used to reason about effect based equivalences

Problems:

- Recursion
- Effect deconstructors
- Higher-order store
- Dependency of values on effects