A general theory of type-and-effect systems via universal algebra

Ohad Kammar     Gordon Plotkin

Journées d’Informatique Fondamentale
de Paris Diderot
April 25, 2013
Pure program transformations

Swap:
\[ M; K = K; M \]

Cache:
\[
\begin{align*}
\text{let } x &= M \text{ in } \\
\text{let } y &= M \text{ in } \\
K
\end{align*}
\]
\[
\begin{align*}
\text{let } x &= M \text{ in } \\
\text{let } y &= x \text{ in } \\
K
\end{align*}
\]

No effects
\[ M \] must not:
- Modify memory.
- Read memory.
- Raise exceptions.
- Be non-deterministic or random.

Effect-dependent optimisations
\[
\begin{align*}
\text{let } x &= M \text{ in } \\
\text{let } y &= M \text{ in } \\
K
\end{align*}
\]
\[
\begin{align*}
\text{let } x &= M \text{ in } \\
\text{let } y &= x \text{ in } \\
K
\end{align*}
\]

If either:
- \( M \) only reads.
- \( M \) only writes.
  (but not both!)
- \( M \) raises exceptions.
Pure program transformations

Swap:

\[
\begin{align*}
M; K &= K; M \\
\text{let } x &= M \text{ in} \\
\text{let } y &= M \text{ in} \\
K &= \text{let } x &= M \text{ in} \\
\text{let } y &= x \text{ in} \\
K
\end{align*}
\]

No effects

\(M\) must not:

\begin{itemize}
\item Modify memory.
\item Read memory.
\item Raise exceptions.
\item Be non-deterministic or random.
\end{itemize}
Effect-dependent optimisations

Swap:

\[
\begin{array}{c}
M; \\
K
\end{array}
= 
\begin{array}{c}
K; \\
M
\end{array}
\]

If either:

- \( M, K \) only read from memory.
- \( M, K \) are probabilistic or non-deterministic.
- \( M, K \) write to different physical memory addresses.
Effect-dependent optimisations

Cache:

\[
\begin{align*}
\text{let } x &= M \text{ in} \\
\text{let } y &= M \text{ in} \\
K
\end{align*}
\]

= 

\[
\begin{align*}
\text{let } x &= M \text{ in} \\
\text{let } y &= x \text{ in} \\
K
\end{align*}
\]

If either:

- \(M\) only reads.
- \(M\) only writes.

(but not both!)

- \(M\) raises exceptions.
Type and effect systems

\[ M : \text{int} \quad \rightarrow \quad M^\# : \text{int} ! \{\text{read, raise}\} \]

\[ \rightarrow \quad \text{typed source} \quad \rightarrow \quad \text{effect analysis} \quad \rightarrow \quad \text{annotated code} \]

Formalizing transformations

\[ \Gamma \vdash M : A ! \{\text{read}\} \quad \Gamma, x : A, y : A \vdash K : B ! \varepsilon \]

\[ \Gamma \vdash \begin{array}{l}
\text{let } x = M \text{ in} \\
\text{let } y = M \text{ in} \\
K
\end{array} \quad = \quad \begin{array}{l}
\text{let } x = M \text{ in} \\
\text{let } y = x \text{ in} \\
K
\end{array} \quad : B ! \varepsilon \]
Problem

- Validate optimisations.
Problem

▶ Validate optimisations.

Rigour is essential:

\[ n \text{ effects} \implies 2^n \text{ effect sets} \]
Problem

- Validate optimisations.
  Rigour is essential:

  $n$ effects $\implies 2^n$ effect sets

- Reuse the theory.
Contribution

Craft
case by case treatment

Science
general semantic account of Gifford-style effect type systems

Engineering
- results
- tools
- methods
Structure

- Previous work
- Algebraic theory of effects
- Type-and-effect systems
- Optimisations
- Engineering
- Enrichment (optional)
- Conclusion and further work


Denotational semantics

▶ Types $A$ denote sets $\llbracket A \rrbracket$, e.g.

$$\llbracket \text{bit} \rrbracket := \{0, 1\}$$

▶ Programs $M : A$ denote elements, e.g., for global state:

$$\llbracket M \rrbracket \in \llbracket \text{bit} \rrbracket \rightarrow \llbracket \text{bit} \rrbracket \times \llbracket A \rrbracket$$

Validity

An optimisation $M = K$ is valid $\iff \llbracket M \rrbracket = \llbracket K \rrbracket$

Benton et al.
Denotational semantics to source and annotated languages
Monads [Moggi’89]

Programs $M : A$ of a sequential, effectful language denote elements of $[M] \in T[A]$ where $T$ is a monad.
Observation [Wadler’98]

Change notation:

\[ \Gamma \vdash M : A ! \varepsilon \quad \Longrightarrow \quad \Gamma \vdash M : T_\varepsilon A \]

\( T_\varepsilon A \) is an indexed family of monadic types.
Suggested monads for global state

\[ T_{\{\text{read}, \text{write}\}}(A) = \text{[bit]} \rightarrow ([\text{bit}] \times A) \]

\[ T_{\text{read}}(A) = \text{[bit]} \rightarrow A \]

\[ T_{\text{write}}(A) = (\{\star\} + \text{[bit]}) \times A \]

\[ T_{\emptyset}(A) = A \]
Universal algebra

Monoids

Signature $\sigma$:

$e : 0$

$\ast : 2$

Equations $E$:

$e \ast x = x$

$x \ast e = x$

$x \ast (y \ast z) = (x \ast y) \ast z$

Derived equations

$E \vdash t = s : x \ast (e \ast y) = x \ast (y \ast e)$
Universal algebra

Signature $\sigma$: $e : 0$
$\ast : 2$

Equations $E$: $e \ast x = x$
$x \ast e = x$
$x \ast (y \ast z) = (x \ast y) \ast z$

Derived equations
$E \vdash t = s : x \ast (e \ast y) = x \ast (y \ast e)$
$(x^{-1})^{-1} = x$
Define:

$$\text{Terms}_\sigma A := \{ t \text{ is a } \sigma\text{-term} \}$$

$$t \approx s \iff E \vdash t = s$$

$$TA := \text{Terms}_\sigma A / \approx$$

Then $T$ is a monad, and, roughly, all monads arise thus.
A theory for state

Signature $\sigma$:

Memoids [Melliès]

read : 2

write_0,

write_1 : 1

Equations $E$:

write_b(write_b'x) = write_b'x

read(write_0x, write_1x) = x

write_b(read(x_0, x_1)) = write_bx_b

The resulting monad satisfies $TA \cong [\text{bit}] \rightarrow [\text{bit}] \times A$. 
effects in annotations $\leftrightarrow$ algebraic operations

subsets $\varepsilon$ of $\sigma$ $\leftrightarrow$ subsignatures $\varepsilon$ of $\sigma$

monads $T_\varepsilon$ $\leftrightarrow$ theories $\langle \varepsilon, E_\varepsilon \rangle$ where:

$$E_\varepsilon := \{ E \vdash t = s \mid t, s \text{ are } \varepsilon\text{-terms} \}$$

e.g., for global state, $E_\varepsilon$ contains:

$$\text{read}(\text{read}(x_0^0, x_1^0), \text{read}(x_0^1, x_1^1)) = \text{read}(x_0^0, x_1^1)$$

We call $E_\varepsilon$ the conservative restriction of $E$ to $\varepsilon$.
The conservative restriction is always defined, but may be hard to calculate.
Theorem
The monad for the conservative restriction of global state to read-only memory is:

\[ T_{\{\text{read}\}} A \cong [\text{bit}] \to A \]

Theorem
The monad for the conservative restriction of global state to write-only memory is:

\[ T_{\{\text{write}_0, \text{write}_1\}} A \cong (\{\star\} + [\text{bit}]) \times A \]
General type-and-effect systems
Plotkin and Power:

\( \langle \sigma, E \rangle \mapsto \) a source language \( \text{Src} \) and denotational semantics for it

Our extension:

\( \langle \sigma, E \rangle \mapsto \) an annotated language \( \text{IL} \) and denotational semantics for it

Define:

\[ \text{Erase} : \text{IL} \rightarrow \text{Src} \]

Theorem
For all closed terms of ground type \( M : T_{\text{\$bit}}, K : T_{\text{\$bit}}, \)

\[ [\text{Erase}(M)] = [\text{Erase}(K)] \quad \iff \quad [M] = [K] \]
Optimisation taxonomy

Structural optimisation

True for every $\langle \sigma, E \rangle$:

- $\beta, \eta$ laws
- sequencing laws: $(M; N); K = M; (N; K)$

also known as:

- constant propagation
- inlining
- common subexpression elimination

in the compiler literature.
Local algebraic optimisations

Single equations from \( E \), e.g.

\[
\text{write}_b(\text{read}(x_0, x_1)) = \text{write}_b x_b
\]

become optimisations

\[
\begin{array}{ll}
a := x; \\
\text{let } y = !a \text{ in } \\
K
\end{array}
= 
\begin{array}{ll}
a := x; \\
\text{let } y = x \text{ in } \\
K
\end{array}
\]
Global algebraic optimisations

Overall interaction of effects. E.g., Discard:

\[ \Gamma \vdash M : T_{\varepsilon}A \quad \Gamma \vdash K : B \]

\[ \Gamma \vdash \begin{array}{l}
\text{let } x = M \text{ in } \\
K
\end{array} = K : B \]

originates from an \textit{absorption law}:

for all \( n \) and \( \varepsilon \)-terms \( t(x_1, \ldots, x_n) \),

\[ t(x, \ldots, x) = x \]
Optimisation taxonomy

Global algebraic optimisations

Similarly,

Cache:

\[
\begin{align*}
\text{let } x & = M \text{ in } \\
\text{let } y & = M \text{ in } \\
K
\end{align*}
\]

\[
\begin{align*}
\text{let } x & = M \text{ in } \\
\text{let } y & = x \text{ in } \\
K
\end{align*}
\]

originates from an \textit{idempotency law}:

for all \( n \) and \( \varepsilon \)-terms \( t(x_1, \ldots, x_n) \),

\[
t(t(x_1^1, \ldots, x_n^1), \ldots, t(x_n^1, \ldots, x_n^n)) = t(x_1^1, \ldots, x_n^n)
\]
New optimisations

The algebraic view is *lightweight*. E.g., slight variation on idempotency:
for all $n$ and $\varepsilon$-terms $t(x_1, \ldots, x_n)$,

$$t(t(x_1, \ldots, x_n), \ldots, t(x_1, \ldots, x_n)) = t(x_1, \ldots, x_n)$$

gives

\[
\begin{array}{l}
\text{let } x = M \text{ in } M; \\
K
\end{array}
\quad = 
\begin{array}{l}
\text{let } x = M \text{ in } K
\end{array}
\]
Theorem
A theory $\langle \varepsilon, E \rangle$ validates the Discard optimisation if and only if for every $op : n$ in $\varepsilon$

$$\text{op}(x, \ldots, x) = x$$

Similarly for Swap, but not for Cache.
Towards engineering

A decision procedure for each optimisation: given $\varepsilon$, is the optimisation valid? 

*optimisation tables* for operation-wise valid optimisations.

**Discard**

<table>
<thead>
<tr>
<th></th>
<th>toss</th>
<th>read</th>
<th>write</th>
<th>throw</th>
<th>get</th>
<th>put</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\zeta$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Swap**

<table>
<thead>
<tr>
<th></th>
<th>toss</th>
<th>read</th>
<th>write</th>
<th>throw</th>
<th>get</th>
<th>put</th>
</tr>
</thead>
<tbody>
<tr>
<td>toss</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>read</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>write</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>throw</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>get</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>put</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Enrichment (optional)
Further work

- More sophisticated setting: domains, locality, concurrency.
  - Extend the algebraic theory of effects.
  - Extend equational logic.
- Foundations of global optimisations.
- Syntactic facets:
  - Effect inference.
  - Sub-effecting and effect polymorphism.
- Richer effect systems.
Further work

- More sophisticated setting: domains, locality, concurrency.
  - Extend the algebraic theory of effects.
  - Extend equational logic.
- Foundations of global optimisations.
- Syntactic facets:
  - Effect inference.
  - Sub-effecting and effect polymorphism.
- Richer effect systems.
Contribution

Craft

case by case treatment

⇓

Science

general semantic account of Gifford-style effect type systems

⇓

Engineering

▷ results

▷ tools

▷ methods