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

UNIVERSITY OF CAMBRIDGE Isaac Newton Trust Computer Laboratory
sicsa*
THE UNIVERSITY of EDINBURGH informatics
THE ROYAL SOCIETY
lfcs Laboratory for Foundations of Computer Science

Pure program transformations

Swap: $\boxed{\begin{array}{l} M; \\ K \end{array}} = \boxed{\begin{array}{l} K; \\ M \end{array}}$

Cache: $\boxed{\begin{array}{l} \mathrm{let}\ x = M\ \mathrm{in} \\ \mathrm{let}\ y = M\ \mathrm{in} \\ K \end{array}} = \boxed{\begin{array}{l} \mathrm{let}\ x = M\ \mathrm{in} \\ \mathrm{let}\ y = x\ \mathrm{in} \\ K \end{array}}$

Pure program transformations

Swap: $\boxed{\begin{array}{l} M; \\ K \end{array}} = \boxed{\begin{array}{l} K; \\ M \end{array}}$

Cache: $\boxed{\begin{array}{lll} \text{let } x & = & M \text{ in} \\ \text{let } y & = & M \text{ in} \\ K & & \end{array}} = \boxed{\begin{array}{lll} \text{let } x & = & M \text{ in} \\ \text{let } y & = & x \text{ in} \\ K & & \end{array}}$

No effects

M must not:

- Modify memory.
- Read memory.
- Raise exceptions.
- Be non-deterministic or random.

Effect-dependent optimisations

Swap: $\boxed{\begin{array}{l} M; \\ K \end{array}} = \boxed{\begin{array}{l} K; \\ M \end{array}}$

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: $\boxed{\begin{array}{l} \text{let } x = M \text{ in} \\ \text{let } y = M \text{ in} \\ K \end{array}} = \boxed{\begin{array}{l} \text{let } x = M \text{ in} \\ \text{let } y = x \text{ in} \\ K \end{array}}$

If either:

- M only reads.
- M only writes.

(but not *both!*)

- M raises exceptions.

Type and effect systems

$$M : \mathrm{int} \quad \longmapsto \quad M^\sharp : \mathrm{int} \,!\, \{\mathrm{read}, \mathrm{raise}\}$$

$$\longrightarrow \quad \boxed{\text{typed source}} \quad \xrightarrow{\text{effect analysis}} \quad \boxed{\text{annotated code}} \quad \longrightarrow$$

Formalizing transformations

$$\frac{\Gamma \vdash M : A \,!\, \{\mathsf{read}\} \qquad \Gamma, x : A, y : A \vdash K : B \,!\, \varepsilon}{\Gamma \vdash \boxed{\begin{array}{l} \mathsf{let}\ x = M\ \mathsf{in} \\ \mathsf{let}\ y = M\ \mathsf{in} \\ K \end{array}} = \boxed{\begin{array}{l} \mathsf{let}\ x = M\ \mathsf{in} \\ \mathsf{let}\ y = x\ \mathsf{in} \\ K \end{array}} : B \,!\, \varepsilon}$$

Problem

- Validate optimisations.

Problem

- Validate optimisations.
  Rigour is essential:

$$n \text{ effects} \qquad \Longrightarrow \qquad 2^n \text{ effect sets}$$

Problem

- Validate optimisations.
  Rigour is essential:

$$n \text{ effects} \qquad \Longrightarrow \qquad 2^n \text{ effect sets}$$

- Reuse the theory.

Contribution

Craft

case by case treatment

$\Downarrow$

Science

general semantic account of Gifford-style effect type systems

$\Downarrow$

Engineering

- results
- tools
- methods

Structure

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

A language a paper

- N. Benton and A. Kennedy. *Monads, effects and transformations*, 1999.
- N. Benton, A. Kennedy, L. Beringer, M. Hofmann. *Reading, writing and relations*, 2006.
- N. Benton and P. Buchlovsky. *Semantics of an effect analysis for exceptions*, 2007.
- N. Benton, A. Kennedy, L. Beringer, M. Hofmann. *Relational semantics for effect-based program transformations with dynamic allocation*, 2007.
- N. Benton, A. Kennedy, L. Beringer, M. Hofmann. *Relational semantics for effect-based program transformations: higher-order store*, 2009.
- J. Thamsborg, L. Birkedal. *A kripke logical relation for effect-based program transformations*, 2011.

Denotational semantics

- Types A *denote* sets $[\![A]\!]$, e.g.

$$[\![\mathrm{bit}]\!] := \{0, 1\}$$

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

$$[\![M]\!] \in [\![\mathrm{bit}]\!] \to [\![\mathrm{bit}]\!] \times [\![A]\!]$$

Validity

$$\text{An optimisation } M = K \text{ is valid} \qquad \iff \qquad [\![M]\!] = [\![K]\!]$$

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*.

Marriage of effects and monads [Wadler and Thiemann'03]

Observation [Wadler'98]

Change notation:

$$\Gamma \vdash M : A \,!\, \varepsilon \implies \Gamma \vdash M : T_\varepsilon A$$

$T_\varepsilon A$ is an indexed family of monadic types.

$$\begin{tikzcd}[column sep=small, row sep=small]
 & \varepsilon_4 & \\
\varepsilon_2 \arrow[ur, phantom, sloped, "\subseteq"] & & \varepsilon_3 \arrow[ul, phantom, sloped, "\supseteq"] \\
 & \varepsilon_1 \arrow[ul, phantom, sloped, "\subseteq"] \arrow[ur, phantom, sloped, "\supseteq"] &
\end{tikzcd}
\implies
\begin{tikzcd}[column sep=small, row sep=small]
 & T_{\varepsilon_4} & \\
T_{\varepsilon_2} \arrow[ur] & & T_{\varepsilon_3} \arrow[ul] \\
 & T_{\varepsilon_1} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

effect inclusions — monad morphisms

Suggested monads for global state

$$
\begin{tikzcd}
 & T_{\{\mathtt{read},\mathtt{write}\}}(A) = [\![\mathrm{bit}]\!] \to ([\![\mathrm{bit}]\!] \times A) & \\
T_{\{\mathtt{read}\}}(A) = [\![\mathrm{bit}]\!] \to A \arrow[ur, hook] & & T_{\{\mathtt{write}\}}(A) = (\{\star\} + [\![\mathrm{bit}]\!]) \times A \arrow[ul, hook] \\
 & T_{\emptyset}(A) = A \arrow[ul, hook] \arrow[ur, hook] &
\end{tikzcd}
$$

Universal algebra

Monoids

Signature σ: $$e : 0$$ $$* : 2$$

Equations E: $$e * x = x$$ $$x * e = x$$ $$x * (y * z) = (x * y) * z$$

Derived equations

$$E \vdash t = s : \quad x * (e * y) = x * (y * e)$$

Universal algebra

	Monoids	Groups
Signature σ:	$e : 0$	$(-)^{-1} : 1$
	$* : 2$	
Equations E:	$e * x = x$	$x^{-1} * x = e$
	$x * e = x$	$x * x^{-1} = e$
	$x * (y * z) = (x * y) * z$	

Derived equations

$$E \vdash t = s : \quad x * (e * y) = x * (y * e) \quad \left(x^{-1}\right)^{-1} = x$$

Define:

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

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

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

Then T is a monad, and, roughly, all monads arise thus.

Algebraic theory of effects [Plotkin and Power]

A theory for state

Memoids [Melliès]

Signature σ: $\quad \mathtt{read} : 2$

$\mathtt{write}_0, \ \mathtt{write}_1 : 1$

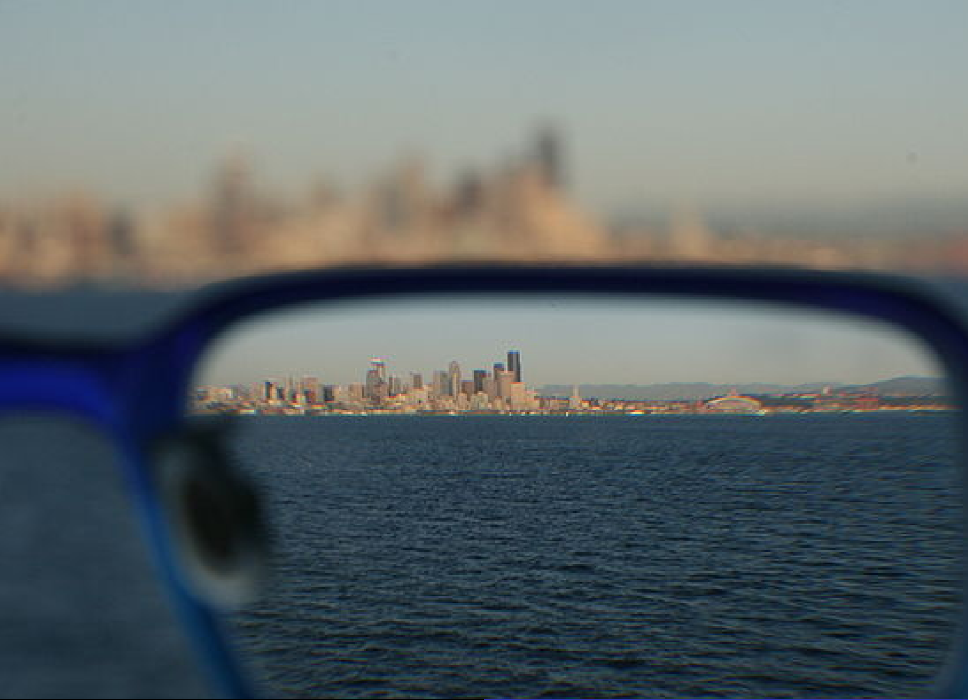
Equations E:

$$\mathtt{write}_b(\mathtt{write}_{b'} x) = \mathtt{write}_{b'} x$$

$$\mathtt{read}(\mathtt{write}_0 x, \mathtt{write}_1 x) = x$$

$$\mathtt{write}_b(\mathtt{read}(x_0, x_1)) = \mathtt{write}_b x_b$$

The resulting monad satisfies $TA \cong [\![\mathrm{bit}]\!] \to [\![\mathrm{bit}]\!] \times A$.



$$\boxed{\boxed{\text{effects in annotations} \leftrightarrow \text{algebraic operations}}}$$

subsets ε of σ $\leftrightarrow$ subsignatures ε of σ

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

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

e.g., for global state, E_ε contains:

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

We call E_ε the *conservative restriction* of E to ε.
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:

$$T_{\{\mathtt{read}\}}A \cong [\![\mathrm{bit}]\!] \to A$$

Theorem

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

$$T_{\{\mathtt{write}_0,\mathtt{write}_1\}}A \cong (\{\star\} + [\![\mathrm{bit}]\!]) \times A$$

General type-and-effect systems

Plotkin and Power:

$\langle \sigma, E \rangle \longmapsto$ a source language Src and denotational semantics for it

Our extension:

$\langle \sigma, E \rangle \longmapsto$ an annotated language IL and denotational semantics for it

Define:

$$\mathrm{Erase} : \mathsf{IL} \to \mathsf{Src}$$

Theorem

For all closed terms of ground type $M : T_\varepsilon \mathrm{bit}$, $K : T_\varepsilon \mathrm{bit}$,

$$[\![\mathrm{Erase}(M)]\!] = [\![\mathrm{Erase}(K)]\!] \qquad \Longleftrightarrow \qquad [\![M]\!] = [\![K]\!]$$

Optimisation taxonomy

Structural optimisation

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

- β, η laws
- sequencing laws: $(M; N); K = M; (N; K)$

also known as:

- constant propagation
- inlining
- common subexpression elimination

in the compiler literature.

Optimisation taxonomy

Local algebraic optimisations

Single equations from E, e.g.

$$\mathtt{write}_b(\mathtt{read}(x_0, x_1)) = \mathtt{write}_b x_b$$

become optimisations

$$\boxed{\begin{array}{l} a := x; \\ \text{let } y = {!a} \text{ in} \\ K \end{array}} = \boxed{\begin{array}{l} a := x; \\ \text{let } y = x \text{ in} \\ K \end{array}}$$

Optimisation taxonomy

Global algebraic optimisations

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

$$\frac{\Gamma \vdash M : T_\varepsilon A \qquad \Gamma \vdash K : B}{\Gamma \vdash \boxed{\begin{array}{l}\text{let } x = M \text{ in} \\ K\end{array}} = \boxed{K} : B}$$

originates from an *absorption law*:
for all n and ε-terms $t(x_1, \ldots, x_n)$,

$$t(x, \ldots, x) = x$$

Optimisation taxonomy

Global algebraic optimisations

Similarly,

Cache:
$$\boxed{\begin{array}{lcl} \text{let } x & = & M \text{ in} \\ \text{let } y & = & M \text{ in} \\ K & & \end{array}} = \boxed{\begin{array}{lcl} \text{let } x & = & M \text{ in} \\ \text{let } y & = & x \text{ in} \\ K & & \end{array}}$$

originates from an *idempotency law*:
for all n and ε-terms $t(x_1, \ldots, x_n)$,

$$t(t(x_1^1, \ldots, x_n^1), \ldots, t(x_1^n, \ldots, x_n^n)) = t(x_1^1, \ldots, x_n^n)$$

Optimisation taxonomy

New optimisations

The algebraic view is *lightweight*.
E.g., slight variation on idempotency:
for all n and ε-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

$$\boxed{\begin{array}{l}\text{let } x \;=\; M \text{ in} \\ M; \\ K\end{array}} \quad = \quad \boxed{\begin{array}{l}\text{let } x \;=\; M \text{ in} \\ K\end{array}}$$

Modularity

Theorem

A theory $\langle \varepsilon, E \rangle$ validates the Discard optimisation if and only if for every $op : n$ in ε

$$op(x, ..., x) = x$$

Similarly for Swap, but *not* for Cache.

Towards engineering

A decision procedure for each optimisation: given ε, is the optimisation valid?
optimisation tables for operation-wise valid optimisations.

Discard

	toss	read	write	throw	get	put
ζ	1	1	0	0	0	0

Swap

	toss	read	write	throw	get	put
toss	1	1	1	1	0	0
read	1	1	0	1	1	1
write	1	0	0	0	1	1
throw	1	1	0	0	0	0
get	0	1	1	0	0	0
put	0	1	1	0	0	0

Enrichment (optional)

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

$\Downarrow$

Science

general semantic account of Gifford-style effect type systems

$\Downarrow$

Engineering

- results
- tools
- methods