Effect systems revisited – control-flow algebra and semantics (slides)

Nielson-Nielson Festschrift 2016

Alan Mycroft[1] Dominic Orchard[1,2] Tomas Petricek[1]

[1]University of Cambridge

[2]Imperial College London

Simple set-based effects à la (Gifford, Lucassen 1986)

$$\boxed{\Gamma \vdash e : \tau, \Phi}$$

Simple set-based effects à la (Gifford, Lucassen 1986)

$$\boxed{\Gamma \vdash e : \tau, \Phi}$$

$$(\textsc{play})\ \frac{}{\Gamma \vdash \mathsf{play}(N, L) : \mathsf{void},\ \{N\}}$$

Simple set-based effects à la (Gifford, Lucassen 1986)

$$\boxed{\Gamma \vdash e : \tau, \Phi}$$

$$(\text{PLAY})\ \frac{}{\Gamma \vdash \mathsf{play}(N, L) : \mathsf{void},\ \{N\}}$$

$$(\text{IF})\ \frac{\Gamma \vdash e_0 : \mathsf{bool}, \Phi_0 \qquad \Gamma \vdash e_1 : \tau, \Phi_1 \qquad \Gamma \vdash e_2 : \tau, \Phi_2}{\Gamma \vdash \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau,\ \Phi_0 \cup \Phi_1 \cup \Phi_2}$$

Simple set-based effects à la (Gifford, Lucassen 1986)

$$\boxed{\Gamma \vdash e : \tau, \Phi}$$

$$(\text{PLAY}) \quad \frac{}{\Gamma \vdash \mathsf{play}(N, L) : \mathsf{void}, \ \{N\}}$$

$$(\text{IF}) \quad \frac{\Gamma \vdash e_0 : \mathsf{bool}, \Phi_0 \qquad \Gamma \vdash e_1 : \tau, \Phi_1 \qquad \Gamma \vdash e_2 : \tau, \Phi_2}{\Gamma \vdash \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau, \ \Phi_0 \cup \Phi_1 \cup \Phi_2}$$

$$(\text{SEQ}) \quad \frac{\Gamma \vdash e_1 : \tau_1, \ \Phi_1 \qquad \Gamma \vdash e_2 : \tau_2, \ \Phi_2}{\Gamma \vdash e_1; e_2 : \tau_2, \ \Phi_1 \cup \Phi_2}$$

Simple set-based effects à la (Gifford, Lucassen 1986)

$$\boxed{\Gamma \vdash e : \tau, \Phi}$$

$$(\text{PLAY}) \quad \frac{}{\Gamma \vdash \mathsf{play}(N, L) : \mathsf{void}, \ \{N\}}$$

$$(\text{IF}) \quad \frac{\Gamma \vdash e_0 : \mathsf{bool}, \Phi_0 \qquad \Gamma \vdash e_1 : \tau, \Phi_1 \qquad \Gamma \vdash e_2 : \tau, \Phi_2}{\Gamma \vdash \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau, \ \Phi_0 \cup \Phi_1 \cup \Phi_2}$$

$$(\text{SEQ}) \quad \frac{\Gamma \vdash e_1 : \tau_1, \ \Phi_1 \qquad \Gamma \vdash e_2 : \tau_2, \ \Phi_2}{\Gamma \vdash e_1 ; e_2 : \tau_2, \ \Phi_1 \cup \Phi_2}$$

$$(\text{FOR}) \quad \frac{\Gamma \vdash e : \mathsf{void}, \ \Phi}{\Gamma \vdash \mathsf{for}\ i = n_1\ \mathsf{to}\ n_2\ \mathsf{do}\ e : \mathsf{void}, \ \Phi}$$

Simple set-based effects à la (Gifford, Lucassen 1986)

$$\boxed{\Gamma \vdash e : \tau, \Phi}$$

$$(\text{PLAY}) \quad \frac{}{\Gamma \vdash \mathsf{play}(N, L) : \mathsf{void}, \ \{N\}}$$

$$(\text{IF}) \quad \frac{\Gamma \vdash e_0 : \mathsf{bool}, \Phi_0 \qquad \Gamma \vdash e_1 : \tau, \Phi_1 \qquad \Gamma \vdash e_2 : \tau, \Phi_2}{\Gamma \vdash \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau, \ \Phi_0 \cup \Phi_1 \cup \Phi_2}$$

$$(\text{SEQ}) \quad \frac{\Gamma \vdash e_1 : \tau_1, \ \Phi_1 \qquad \Gamma \vdash e_2 : \tau_2, \ \Phi_2}{\Gamma \vdash e_1 ; e_2 : \tau_2, \ \Phi_1 \cup \Phi_2}$$

$$(\text{FOR}) \quad \frac{\Gamma \vdash e : \mathsf{void}, \ \Phi}{\Gamma \vdash \mathsf{for}\ i = n_1\ \mathsf{to}\ n_2\ \mathsf{do}\ e : \mathsf{void}, \ \Phi}$$

for our example we get $\{G3, F3, A3, B3, C4, D4, E4, G4\}$

...in POPL'94, richer effect systems for concurrency

Higher-Order Concurrent Programs with Finite Communication Topology

(Extended Abstract)

Hanne Riis Nielson Flemming Nielson

Computer Science Department
Aarhus University, Denmark.

e-mail: {hrnielson,fnielson}@daimi.aau.dk

Abstract

Concurrent ML (CML) is an extension of the functional language Standard ML (SML) with primitives for the dynamic creation of processes and channels and for the communication of values over channels. Because of the powerful abstraction mechanisms the communication topology of a given program may be very complex and therefore an efficient implementation may be facilitated by knowledge of the topology.

This paper presents an analysis for determining when a bounded number of processes and channels will be generated. The analysis proceeds in two stages. First we extend a polymorphic type system for SML to deduce not only the type of CML programs but also their communication behaviour expressed as terms in a new process algebra. Next we develop an analysis that given

1 Introduction

Higher-order concurrent languages as CML [8] and Facile [2] offer mechanisms for the *dynamic* creation of channels and processes in addition to the possibility of sending and receiving values over channels. To obtain an *efficient implementation* of programs in such languages we would need information about their communication topology:

- Does the program only spawn a finite number of processes?
- Does the program only create a finite number of channels?

If the answer to the first question is yes we may load the processes on the available processors and dispense with

...in POPL'94, richer effect systems for concurrency

...in POPL'94, richer effect systems for concurrency

- *Causality* of operations (from sets to lists of effects)

...in POPL'94, richer effect systems for concurrency

- *Causality* of operations (from sets to lists of effects)

$$\text{(SEQ)}\ \frac{\Gamma \vdash e_1 : \tau_1, \quad \Phi_1 \qquad \Gamma \vdash e_2 : \tau_2, \quad \Phi_2}{\Gamma \vdash e_1; e_2 : \tau_2, \quad \Phi_1; \Phi_2}$$

...in POPL'94, richer effect systems for concurrency

- *Causality* of operations (from sets to lists of effects)

$$(\textsc{seq})\ \frac{\Gamma \vdash e_1 : \tau_1, \quad \Phi_1 \qquad \Gamma \vdash e_2 : \tau_2, \quad \Phi_2}{\Gamma \vdash e_1; e_2 : \tau_2, \quad \Phi_1; \Phi_2}$$

- *Branching control-flow* (from lists to trees)

...in POPL'94, richer effect systems for concurrency

- *Causality* of operations (from sets to lists of effects)

$$(\text{SEQ})\ \frac{\Gamma \vdash e_1 : \tau_1,\quad \Phi_1 \qquad \Gamma \vdash e_2 : \tau_2,\quad \Phi_2}{\Gamma \vdash e_1; e_2 : \tau_2,\quad \Phi_1; \Phi_2}$$

- *Branching control-flow* (from lists to trees)

$$(\text{IF})\ \frac{\Gamma \vdash e_0 : \mathsf{bool}, \Phi_0 \qquad \Gamma \vdash e_1 : \tau, \Phi_1 \qquad \Gamma \vdash e_2 : \tau, \Phi_2}{\Gamma \vdash \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau,\ \Phi_0; (\Phi_1 + \Phi_2)}$$

...in POPL'94, richer effect systems for concurrency

- *Causality* of operations (from sets to lists of effects)

$$(\textsc{seq})\ \frac{\Gamma \vdash e_1 : \tau_1,\quad \Phi_1 \qquad \Gamma \vdash e_2 : \tau_2,\quad \Phi_2}{\Gamma \vdash e_1; e_2 : \tau_2,\quad \Phi_1; \Phi_2}$$

- *Branching control-flow* (from lists to trees)

$$(\textsc{if})\ \frac{\Gamma \vdash e_0 : \mathsf{bool}, \Phi_0 \qquad \Gamma \vdash e_1 : \tau, \Phi_1 \qquad \Gamma \vdash e_2 : \tau, \Phi_2}{\Gamma \vdash \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau,\ \Phi_0; (\Phi_1 + \Phi_2)}$$

- *Parallel control-flow*

...in POPL'94, richer effect systems for concurrency

- *Causality* of operations (from sets to lists of effects)

$$(\textsc{seq})\ \frac{\Gamma \vdash e_1 : \tau_1,\quad \Phi_1 \qquad \Gamma \vdash e_2 : \tau_2,\quad \Phi_2}{\Gamma \vdash e_1; e_2 : \tau_2,\quad \Phi_1; \Phi_2}$$

- *Branching control-flow* (from lists to trees)

$$(\textsc{if})\ \frac{\Gamma \vdash e_0 : \mathsf{bool}, \Phi_0 \qquad \Gamma \vdash e_1 : \tau, \Phi_1 \qquad \Gamma \vdash e_2 : \tau, \Phi_2}{\Gamma \vdash \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau,\ \Phi_0; (\Phi_1 + \Phi_2)}$$

- *Parallel control-flow*

$$(\textsc{fork})\ \frac{\Gamma \vdash e : \mathsf{unit} \xrightarrow{\Phi_0} \tau, \epsilon}{\Gamma \vdash \mathit{fork}\, e : \tau,\ \mathit{FORK}\, \Phi_0}$$

…in 1999, rich general effect-system structure

Type and Effect Systems

Flemming Nielson and Hanne Riis Nielson

Department of Computer Science, Aarhus University, Denmark.

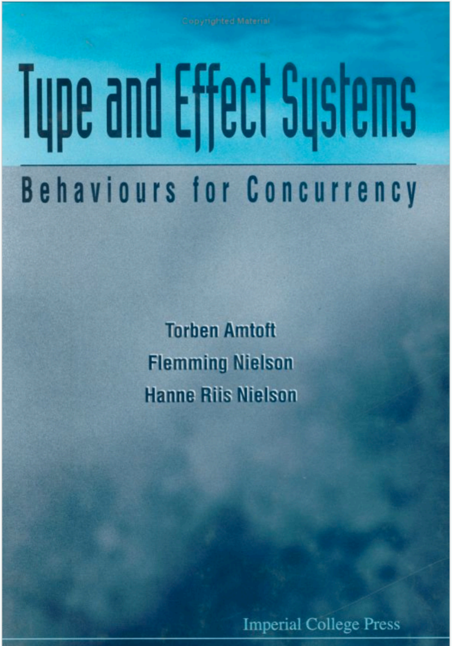
Abstract. The design and implementation of a correct system can benefit from employing static techniques for ensuring that the dynamic behaviour satisfies the specification. Many programming languages incorporate types for ensuring that certain operations are only applied to data of the appropriate form. A natural extension of type checking techniques is to enrich the types with annotations and effects that further describe intensional aspects of the dynamic behaviour.

Keywords. Polymorphic type systems, effect annotations, subeffecting and subtyping, semantic correctness, type inference algorithms, syntactic soundness and completeness. Analyses for control flow, binding times, side effects, region structure, and communication structure.

1 Introduction

Static analysis of programs comprises a broad collection of techniques for predicting safe and computable approximations to the set of values or behaviours arising dynamically during computation; this may be used to validate program transformations, to generate more efficient code or to increase the understanding

...in 1999, rich general effect-system structure



...in 1999, rich general effect-system structure

...in 1999, rich general effect-system structure

(Ordered) Semiring of effects $(\mathcal{F}, \bullet, 1, +, \sqsubseteq)$

...in 1999, rich general effect-system structure

(Ordered) Semiring of effects $(\mathcal{F}, \bullet, 1, +, \sqsubseteq)$

- $(\mathcal{F}, \bullet, 1)$ for sequential composition (monoid)

...in 1999, rich general effect-system structure

(Ordered) Semiring of effects $(\mathcal{F}, \bullet, 1, +, \sqsubseteq)$

- $(\mathcal{F}, \bullet, 1)$ for sequential composition (monoid)
- $(\mathcal{F}, +)$ for alternation (semigroup)

...in 1999, rich general effect-system structure

(Ordered) Semiring of effects $(\mathcal{F}, \bullet, 1, +, \sqsubseteq)$

- $(\mathcal{F}, \bullet, 1)$ for sequential composition (monoid)
- $(\mathcal{F}, +)$ for alternation (semigroup)
- $\sqsubseteq$ for subeffecting

...in 1999, rich general effect-system structure

(Ordered) Semiring of effects $(\mathcal{F}, \bullet, 1, +, \sqsubseteq)$

- $(\mathcal{F}, \bullet, 1)$ for sequential composition (monoid)
- $(\mathcal{F}, +)$ for alternation (semigroup)
- $\sqsubseteq$ for subeffecting
- ... with fixed-point $\text{rec}\beta.\Phi$

...in 1999, rich general effect-system structure

(Ordered) Semiring of effects $(\mathcal{F}, \bullet, 1, +, \sqsubseteq)$

- $(\mathcal{F}, \bullet, 1)$ for sequential composition (monoid)
- $(\mathcal{F}, +)$ for alternation (semigroup)
- $\sqsubseteq$ for subeffecting
- ... with fixed-point $\mathsf{rec}\beta.\Phi$
- ... subeffecting axioms provide semiring equations, *e.g.*

$$(\Phi_1 + \Phi_2); \Phi_3 \sqsubseteq (\Phi_1; \Phi_3) + (\Phi_1; \Phi_2)$$
$$(\Phi_1; \Phi_3) + (\Phi_1; \Phi_2) \sqsubseteq (\Phi_1 + \Phi_2); \Phi_3$$

For our musical example...

Two example instances:

For our musical example...

Two example instances:

- **Song structure** via term-algebra:

$$\Phi = \mathsf{C}, \mathsf{D}, \mathsf{E}, \ldots, \mathsf{rest} \mid \Phi_1 + \Phi_2 \mid \Phi_1 \bullet \Phi_2 \mid \Phi^*$$

For our musical example...

Two example instances:

- **Song structure** via term-algebra:

$$\Phi = \mathsf{C}, \mathsf{D}, \mathsf{E}, \ldots, \mathsf{rest} \mid \Phi_1 + \Phi_2 \mid \Phi_1 \bullet \Phi_2 \mid \Phi^*$$

with $\Gamma \vdash \mathsf{play}(N, L) : \mathsf{void}, N$

For our musical example...

Two example instances:

- **Song structure** via term-algebra:

$$\Phi = \mathsf{C}, \mathsf{D}, \mathsf{E}, \ldots, \mathsf{rest} \mid \Phi_1 + \Phi_2 \mid \Phi_1 \bullet \Phi_2 \mid \Phi^*$$

  with $\Gamma \vdash \mathsf{play}(N, L) : \mathsf{void}, N$

  gives
  $(\mathsf{G3}\bullet\mathsf{G3}\bullet(\mathsf{G4}\bullet\mathsf{E4}\bullet\mathsf{C4}\bullet\mathsf{B3}\bullet\mathsf{A3}+\mathsf{A3}\bullet\mathsf{G3}\bullet(\mathsf{D4}\bullet\mathsf{C4}+\mathsf{C4}\bullet\mathsf{B3})))^*$

For our musical example...

Two example instances:

- **Song structure** via term-algebra:

$$\Phi = \mathsf{C}, \mathsf{D}, \mathsf{E}, \ldots, \mathsf{rest} \mid \Phi_1 + \Phi_2 \mid \Phi_1 \bullet \Phi_2 \mid \Phi^*$$

  with $\Gamma \vdash \mathsf{play}(N, L) : \mathsf{void}, N$

  gives
  $(\mathsf{G3} \bullet \mathsf{G3} \bullet (\mathsf{G4} \bullet \mathsf{E4} \bullet \mathsf{C4} \bullet \mathsf{B3} \bullet \mathsf{A3} + \mathsf{A3} \bullet \mathsf{G3} \bullet (\mathsf{D4} \bullet \mathsf{C4} + \mathsf{C4} \bullet \mathsf{B3})))^*$

- or, **timing** via $(\mathbb{R}^+, +, 0, \max, \leq)$

For our musical example...

Two example instances:

- **Song structure** via term-algebra:

  $$\Phi = \mathsf{C}, \mathsf{D}, \mathsf{E}, \ldots, \mathsf{rest} \mid \Phi_1 + \Phi_2 \mid \Phi_1 \bullet \Phi_2 \mid \Phi^*$$

  with $\Gamma \vdash \mathsf{play}(N, L) : \mathsf{void}, N$

  gives
  $(\mathsf{G3} \bullet \mathsf{G3} \bullet (\mathsf{G4} \bullet \mathsf{E4} \bullet \mathsf{C4} \bullet \mathsf{B3} \bullet \mathsf{A3} + \mathsf{A3} \bullet \mathsf{G3} \bullet (\mathsf{D4} \bullet \mathsf{C4} + \mathsf{C4} \bullet \mathsf{B3})))^*$

- or, **timing** via $(\mathbb{R}^+, +, 0, \max, \leq)$
  with $\Gamma \vdash \mathsf{play}(N, L) : \mathsf{void}, L$

For our musical example...

Two example instances:

- **Song structure** via term-algebra:

  $$\Phi = \mathsf{C}, \mathsf{D}, \mathsf{E}, \ldots, \mathsf{rest} \mid \Phi_1 + \Phi_2 \mid \Phi_1 \bullet \Phi_2 \mid \Phi^*$$

  with $\Gamma \vdash \mathsf{play}(N, L) : \mathsf{void}, N$

  gives
  $(\mathsf{G3} \bullet \mathsf{G3} \bullet (\mathsf{G4} \bullet \mathsf{E4} \bullet \mathsf{C4} \bullet \mathsf{B3} \bullet \mathsf{A3} + \mathsf{A3} \bullet \mathsf{G3} \bullet (\mathsf{D4} \bullet \mathsf{C4} + \mathsf{C4} \bullet \mathsf{B3})))^*$

- or, **timing** via $(\mathbb{R}^+, +, 0, \max, \leq)$
  with $\Gamma \vdash \mathsf{play}(N, L) : \mathsf{void}, L$
  gives 13 seconds

from "Type and effect systems" (Nielson, Nielson, 1999)

"Now we turn to explaining the individual steps in the overall methodology of designing and using type and effect systems:

- *devise a semantics for the programming language,*
- *develop a program analysis in the form of a type and effect system ...*
- *prove the semantic correctness of the analysis,*
- *develop an efficient inference algorithm, ..."*

from "Type and effect systems" (Nielson, Nielson, 1999)

"Now we turn to explaining the individual steps in the overall methodology of designing and using type and effect systems:

- *devise a semantics for the programming language,*
- *develop a program analysis in the form of a type and effect system ...*
- *prove the semantic correctness of the analysis,*
- *develop an efficient inference algorithm, ..."*

from "Type and effect systems" (Nielson, Nielson, 1999)

"Now we turn to explaining the individual steps in the overall methodology of designing and using type and effect systems:

- *devise a semantics for the programming language,*
- *develop a program analysis in the form of a type and effect system ...*
- *prove the semantic correctness of the analysis,*
- *develop an efficient inference algorithm, ..."*

from “Type and effect systems” (Nielson, Nielson, 1999)

“Now we turn to explaining the individual steps in the overall methodology of designing and using type and effect systems:

- *devise a semantics for the programming language,*
- *develop a program analysis in the form of a type and effect system ...*
- *prove the semantic correctness of the analysis,*
- *develop an efficient inference algorithm, ...”*

from "Type and effect systems" (Nielson, Nielson, 1999)

"Now we turn to explaining the individual steps in the overall methodology of designing and using type and effect systems:

- *devise a semantics for the programming language,*
- *develop a program analysis in the form of a type and effect system ...*
- *prove the semantic correctness of the analysis,*
- *develop an efficient inference algorithm, ..."*

- An effect-directed semantics unifies the first three steps

from “Type and effect systems” (Nielson, Nielson, 1999)

“Now we turn to explaining the individual steps in the overall methodology of designing and using type and effect systems:

- *devise a semantics for the programming language,*
- *develop a program analysis in the form of a type and effect system ...*
- *prove the semantic correctness of the analysis,*
- *develop an efficient inference algorithm, ...”*

- An effect-directed semantics unifies the first three steps
- We develop an effect-directed semantics for rich Nielson-Nielson-style effects

Type-directed semantics

Type-directed semantics

untyped model: $[\![e]\!] : D$

$$D \;\cong\; \mathbb{Z} + (D \to D) + \{\mathsf{wrong}\}$$

Type-directed semantics

untyped model: $[\![e]\!] : D$

$$D \cong \mathbb{Z} + (D \to D) + \{\mathsf{wrong}\}$$

simply typed model: $[\![\Gamma \vdash e : \tau]\!] : D_\tau$

$$D_{\mathsf{int}} = \mathbb{Z}$$
$$D_{\sigma\to\tau} = D_\sigma \to D_\tau$$

Type-directed semantics

untyped model: $[\![e]\!] : D$

$$D \cong \mathbb{Z} + (D \to D) + \{\mathsf{wrong}\}$$

simply typed model: $[\![\Gamma \vdash e : \tau]\!] : D_\tau$

$$D_{\mathsf{int}} = \mathbb{Z}$$
$$D_{\sigma\to\tau} = D_\sigma \to D_\tau$$

effect-directed model: $[\![\Gamma \vdash e : \tau, F]\!] : D_\tau^F$

Type-directed semantics

untyped model: $[\![e]\!] : D$

$$D \cong \mathbb{Z} + (D \to D) + \{\mathsf{wrong}\}$$

simply typed model: $[\![\Gamma \vdash e : \tau]\!] : D_\tau$

$$D_{\mathsf{int}} = \mathbb{Z}$$
$$D_{\sigma\to\tau} = D_\sigma \to D_\tau$$

effect-directed model: $[\![\Gamma \vdash e : \tau, F]\!] : D_\tau^F$

Core idea: algebra-semantics homomorphism

Algebraic structure of $\mathcal{F}$ determines structure on family $D_\tau^{\mathcal{F}}$.

Effect analysis and semantics

Lattice effects
(Gifford, Lucassen '86)
$\Gamma \vdash e : \tau, \mathcal{F}$

⟹

Richer structure on $\mathcal{F}$
(Nielson, Nielson '94, '99)

⇓

"Marriage of effects and monads"
(Wadler, Thiemann '03)
$[\![\Gamma \vdash e : M^{\mathcal{F}}\tau]\!] : [\![\Gamma]\!] \to \mathsf{T}[\![\tau]\!]$

⇓

Effect-directed semantics
(Katsumata '14)
$[\![\Gamma \vdash e : M^{\mathcal{F}}\tau]\!] : [\![\Gamma]\!] \to \mathsf{T}_{\mathcal{F}}[\![\tau]\!]$
(graded monads)

⟹

This paper
Richer effect-directed semantics
(graded joinads)

Effect analysis and semantics

Lattice effects
(Gifford, Lucassen '86)
$\Gamma \vdash e : \tau, \mathcal{F}$

$\Longrightarrow$

Richer structure on $\mathcal{F}$
(Nielson, Nielson '94, '99)

$\Downarrow$

"Marriage of effects and monads"
(Wadler, Thiemann '03)
$[\![\Gamma \vdash e : M^{\mathcal{F}}\tau]\!] : [\![\Gamma]\!] \to \mathsf{T}[\![\tau]\!]$

$\Downarrow$

Effect-directed semantics
(Katsumata '14)
$[\![\Gamma \vdash e : M^{\mathcal{F}}\tau]\!] : [\![\Gamma]\!] \to \mathsf{T}_{\mathcal{F}}[\![\tau]\!]$
(graded monads)

$\Longrightarrow$

This paper
Richer effect-directed semantics
(graded joinads)

Operations on $\mathsf{T}_{\mathcal{F}}$ homomorphic to operations on $\mathcal{F}$

Modelling effects with graded monads (Katsumata 2014)

Modelling effects with graded monads (Katsumata 2014)

$$[\![\Gamma \vdash e : \tau, F]\!] : [\![\Gamma]\!] \to \mathsf{T}_F[\![\tau]\!]$$

Modelling effects with graded monads (Katsumata 2014)

$$[\![\Gamma \vdash e : \tau, F]\!] : [\![\Gamma]\!] \to \mathsf{T}_F [\![\tau]\!]$$

e.g. for state $\mathsf{T}A = S \to (A \times S)$.

Modelling effects with graded monads (Katsumata 2014)

$$[\![\Gamma \vdash e : \tau, F]\!] : [\![\Gamma]\!] \to \mathsf{T}_F[\![\tau]\!]$$

e.g. for state $\mathsf{T}_F A = (\mathit{reads}(F)) \to (A \times \mathit{writes}(F))$.

Modelling effects with graded monads (Katsumata 2014)

$$[\![\Gamma \vdash e : \tau, F]\!] : [\![\Gamma]\!] \to \mathsf{T}_F[\![\tau]\!]$$

e.g. for state $\mathsf{T}_F A = (\mathit{reads}(F)) \to (A \times \mathit{writes}(F))$.
e.g. for partiality $\mathsf{T}A = \bot + A$

Modelling effects with graded monads (Katsumata 2014)

$$[\![\Gamma \vdash e : \tau, F]\!] : [\![\Gamma]\!] \to \mathsf{T}_F[\![\tau]\!]$$

e.g. for state $\mathsf{T}_F A = (\mathit{reads}(F)) \to (A \times \mathit{writes}(F))$.
e.g. for partiality $\mathsf{T}_\bot A = \bot$, $\mathsf{T}_\top A = A$,

Modelling effects with graded monads (Katsumata 2014)

$$[\![\Gamma \vdash e : \tau, F]\!] : [\![\Gamma]\!] \to \mathsf{T}_F[\![\tau]\!]$$

e.g. for state $\mathsf{T}_F A = (\mathit{reads}(F)) \to (A \times \mathit{writes}(F))$.
e.g. for partiality $\mathsf{T}_\bot A = \bot$, $\mathsf{T}_\top A = A$, $\mathsf{T}_? A = A + 1$

Modelling effects with graded monads (Katsumata 2014)

$$[\![\Gamma \vdash e : \tau, F]\!] : [\![\Gamma]\!] \to \mathsf{T}_F [\![\tau]\!]$$

e.g. for state $\mathsf{T}_F A = (reads(F)) \to (A \times writes(F))$.
e.g. for partiality $\mathsf{T}_\bot A = \bot$, $\mathsf{T}_\top A = A$, $\mathsf{T}_? A = A + 1$

Graded monads provide sequential composition

Modelling effects with graded monads (Katsumata 2014)

$$[\![\Gamma \vdash e : \tau, F]\!] : [\![\Gamma]\!] \to \mathsf{T}_F [\![\tau]\!]$$

e.g. for state $\mathsf{T}_F A = (reads(F)) \to (A \times writes(F))$.
e.g. for partiality $\mathsf{T}_\bot A = \bot$, $\mathsf{T}_\top A = A$, $\mathsf{T}_? A = A + 1$

Graded monads provide sequential composition

Let $(\mathcal{F}, \bullet, 0)$ be an effect monoid.

Modelling effects with graded monads (Katsumata 2014)

$$[\![\Gamma \vdash e : \tau, F]\!] : [\![\Gamma]\!] \to \mathsf{T}_F[\![\tau]\!]$$

e.g. for state $\mathsf{T}_F A = (\mathit{reads}(F)) \to (A \times \mathit{writes}(F))$.
e.g. for partiality $\mathsf{T}_\bot A = \bot$, $\mathsf{T}_\top A = A$, $\mathsf{T}_? A = A + 1$

Graded monads provide sequential composition

Let $(\mathcal{F}, \bullet, 0)$ be an effect monoid.
Given $d_1 : A \to \mathsf{T}_F B$ and $d_2 : B \to \mathsf{T}_G C$

Modelling effects with graded monads (Katsumata 2014)

$$[\![\Gamma \vdash e : \tau, F]\!] : [\![\Gamma]\!] \to \mathsf{T}_F [\![\tau]\!]$$

e.g. for state $\mathsf{T}_F A = (\mathit{reads}(F)) \to (A \times \mathit{writes}(F))$.
e.g. for partiality $\mathsf{T}_\bot A = \bot$, $\mathsf{T}_\top A = A$, $\mathsf{T}_? A = A + 1$

Graded monads provide sequential composition

Let $(\mathcal{F}, \bullet, 0)$ be an effect monoid.
Given $d_1 : A \to \mathsf{T}_F B$ and $d_2 : B \to \mathsf{T}_G C$
then $d_2 \,\hat{\circ}\, d_1 : A \to \mathsf{T}_{F \bullet G} C$

Modelling effects with graded monads (Katsumata 2014)

$$[\![\Gamma \vdash e : \tau, F]\!] : [\![\Gamma]\!] \to \mathsf{T}_F[\![\tau]\!]$$

e.g. for state $\mathsf{T}_F A = (\mathit{reads}(F)) \to (A \times \mathit{writes}(F))$.
e.g. for partiality $\mathsf{T}_\bot A = \bot$, $\mathsf{T}_\top A = A$, $\mathsf{T}_? A = A + 1$

Graded monads provide sequential composition

Let $(\mathcal{F}, \bullet, 0)$ be an effect monoid.
Given $d_1 : A \to \mathsf{T}_F B$ and $d_2 : B \to \mathsf{T}_G C$
then $d_2 \,\hat{\circ}\, d_1 : A \to \mathsf{T}_{F \bullet G} C$ with $\hat{id}_A : A \to \mathsf{T}_0 A$.

Modelling effects with graded monads (Katsumata 2014)

$$[\![\Gamma \vdash e : \tau, F]\!] : [\![\Gamma]\!] \rightarrow \mathsf{T}_F [\![\tau]\!]$$

e.g. for state $\mathsf{T}_F A = (\mathit{reads}(F)) \rightarrow (A \times \mathit{writes}(F))$.
e.g. for partiality $\mathsf{T}_\bot A = \bot$, $\mathsf{T}_\top A = A$, $\mathsf{T}_? A = A + 1$

Graded monads provide sequential composition

Let $(\mathcal{F}, \bullet, 0)$ be an effect monoid.
Given $d_1 : A \rightarrow \mathsf{T}_F B$ and $d_2 : B \rightarrow \mathsf{T}_G C$
then $d_2 \,\hat{\circ}\, d_1 : A \rightarrow \mathsf{T}_{F \bullet G} C$ with $\hat{id}_A : A \rightarrow \mathsf{T}_0 A$.

With ordering

Given partially ordered $(\mathcal{F}, \bullet, 0, \sqsubseteq)$ then for all $F \sqsubseteq G$ then coercion: $\iota_{F,G,A} : \mathsf{T}_F A \rightarrow \mathsf{T}_G A$

Semantics of branching: derived vs. parameterised

Semantics of branching: derived vs. parameterised

$$[\![\text{if } e_0 \text{ then } e_1 \text{ else } e_2]\!] = \text{COND}([\![e_0]\!], [\![e_1]\!], [\![e_2]\!])$$

Semantics of branching: derived vs. parameterised

$$[\![\textsf{if } e_0 \textsf{ then } e_1 \textsf{ else } e_2]\!] = \mathsf{COND}([\![e_0]\!], [\![e_1]\!], [\![e_2]\!])$$

- COND definable via cond : $\mathbb{B} \times A \times A \to A$

Semantics of branching: derived vs. parameterised

$$[\![\textsf{if } e_0 \textsf{ then } e_1 \textsf{ else } e_2]\!] = \mathrm{COND}([\![e_0]\!], [\![e_1]\!], [\![e_2]\!])$$

- COND definable via $\mathrm{cond} : \mathbb{B} \times A \times A \to A$

$$\mathrm{cond}(\mathrm{true}, x, y) = x \qquad \mathrm{cond}(\mathrm{false}, x, y) = y$$

Semantics of branching: derived vs. parameterised

$$[\![\textsf{if } e_0 \textsf{ then } e_1 \textsf{ else } e_2]\!] = \mathsf{COND}([\![e_0]\!], [\![e_1]\!], [\![e_2]\!])$$

- COND definable via cond : $\mathbb{B} \times A \times A \to A$

$$\mathsf{cond}(\mathsf{true}, x, y) = x \qquad \mathsf{cond}(\mathsf{false}, x, y) = y$$

- Restricts effect branching behaviour:

Semantics of branching: derived vs. parameterised

$$[\![\text{if } e_0 \text{ then } e_1 \text{ else } e_2]\!] = \text{COND}([\![e_0]\!], [\![e_1]\!], [\![e_2]\!])$$

- COND definable via $\text{cond} : \mathbb{B} \times A \times A \to A$

$$\text{cond}(\text{true}, x, y) = x \qquad \text{cond}(\text{false}, x, y) = y$$

- Restricts effect branching behaviour:

$$(\text{IF})\ \frac{\Gamma \vdash e_0 : \text{bool}, F \qquad \Gamma \vdash e_1 : \tau, G \qquad \Gamma \vdash e_2 : \tau, H}{\Gamma \vdash \text{if } e_0 \text{ then } e_1 \text{ else } e_2 : \tau,\ F \bullet (G \sqcup H)}$$

Semantics of branching: derived vs. parameterised

$$[\![\text{if } e_0 \text{ then } e_1 \text{ else } e_2]\!] = \mathrm{COND}([\![e_0]\!], [\![e_1]\!], [\![e_2]\!])$$

- COND definable via $\mathrm{cond} : \mathbb{B} \times A \times A \to A$

$$\mathrm{cond}(\mathrm{true}, x, y) = x \qquad \mathrm{cond}(\mathrm{false}, x, y) = y$$

- Restricts effect branching behaviour:

$$(\textsc{if})\ \frac{\Gamma \vdash e_0 : \mathrm{bool}, F \qquad \Gamma \vdash e_1 : \tau, G \qquad \Gamma \vdash e_2 : \tau, H}{\Gamma \vdash \text{if } e_0 \text{ then } e_1 \text{ else } e_2 : \tau,\ F \bullet (G \sqcup H)}$$

$$\mathrm{cond} : \mathbb{B} \times \mathsf{T}_{G \sqcup H} A \times \mathsf{T}_{G \sqcup H} A \to \mathsf{T}_{G \sqcup H} A$$

Semantics of branching: derived vs. parameterised

$$[\![\text{if } e_0 \text{ then } e_1 \text{ else } e_2]\!] = \text{COND}([\![e_0]\!], [\![e_1]\!], [\![e_2]\!])$$

- COND definable via $\text{cond} : \mathbb{B} \times A \times A \to A$

$$\text{cond}(\text{true}, x, y) = x \qquad \text{cond}(\text{false}, x, y) = y$$

- Restricts effect branching behaviour:

$$(\text{IF})\ \frac{\Gamma \vdash e_0 : \text{bool}, F \qquad \Gamma \vdash e_1 : \tau, G \qquad \Gamma \vdash e_2 : \tau, H}{\Gamma \vdash \text{if } e_0 \text{ then } e_1 \text{ else } e_2 : \tau,\ F \bullet (G \sqcup H)}$$

$$\text{cond} : \mathbb{B} \times \mathsf{T}_{G \sqcup H} A \times \mathsf{T}_{G \sqcup H} A \to \mathsf{T}_{G \sqcup H} A$$

- Towards Nielson-Nielson richer effects, but $+$ may not be $\sqcup$.

Semantics of branching: derived vs. parameterised

$$[\![\text{if } e_0 \text{ then } e_1 \text{ else } e_2]\!] = \mathrm{COND}([\![e_0]\!], [\![e_1]\!], [\![e_2]\!])$$

- COND definable via $\mathrm{cond} : \mathbb{B} \times A \times A \to A$

$$\mathrm{cond}(\mathrm{true}, x, y) = x \qquad \mathrm{cond}(\mathrm{false}, x, y) = y$$

- Restricts effect branching behaviour:

$$(\text{IF})\ \frac{\Gamma \vdash e_0 : \text{bool}, F \qquad \Gamma \vdash e_1 : \tau, G \qquad \Gamma \vdash e_2 : \tau, H}{\Gamma \vdash \text{if } e_0 \text{ then } e_1 \text{ else } e_2 : \tau,\ F \bullet (G \sqcup H)}$$

$$\mathrm{cond} : \mathbb{B} \times \mathsf{T}_{G \sqcup H} A \times \mathsf{T}_{G \sqcup H} A \to \mathsf{T}_{G \sqcup H} A$$

- Towards Nielson-Nielson richer effects, but $+$ may not be $\sqcup$.
- Instead: parameterise semantics on
  $\mathrm{COND} : \mathsf{T}_F \mathbb{B} \times \mathsf{T}_G A \times \mathsf{T}_H A \to \mathsf{T}_{?+(F,G,H)} A$

Definition: jonoid

For a set of effects $\mathcal{F}$ then $(\mathcal{F}, \bullet, I, \&, ?+, \sqsubseteq)$ is a *joinoid control-flow algebra*:

Definition: jonoid

For a set of effects $\mathcal{F}$ then $(\mathcal{F}, \bullet, I, \&, ?\!+, \sqsubseteq)$ is a *joinoid control-flow algebra*:

- $(\mathcal{F}, \bullet, I)$ is a monoid, representing sequential composition and purity;

Definition: jonoid

For a set of effects $\mathcal{F}$ then $(\mathcal{F}, \bullet, I, \&, ?\!+, \sqsubseteq)$ is a *joinoid control-flow algebra*:

- $(\mathcal{F}, \bullet, I)$ is a monoid, representing sequential composition and purity;
- $(\mathcal{F}, \&, I)$ is a commutative monoid, representing parallel composition;

Definition: jonoid

For a set of effects $\mathcal{F}$ then $(\mathcal{F}, \bullet, I, \&, ?{+}, \sqsubseteq)$ is a *joinoid control-flow algebra*:

- $(\mathcal{F}, \bullet, I)$ is a monoid, representing sequential composition and purity;
- $(\mathcal{F}, \&, I)$ is a commutative monoid, representing parallel composition;
- letting $F + G = ?{+}(I, F, G)$ (pure guard)

Definition: jonoid

For a set of effects $\mathcal{F}$ then $(\mathcal{F}, \bullet, I, \&, ?{+}, \sqsubseteq)$ is a *joinoid control-flow algebra*:

- $(\mathcal{F}, \bullet, I)$ is a monoid, representing sequential composition and purity;
- $(\mathcal{F}, \&, I)$ is a commutative monoid, representing parallel composition;
- letting $F + G = ?{+}(I, F, G)$ (pure guard) $(\mathcal{F}, +)$ is a semigroup, representing choice between branches

Definition: jonoid

For a set of effects $\mathcal{F}$ then $(\mathcal{F}, \bullet, I, \&, ?{+}, \sqsubseteq)$ is a *joinoid control-flow algebra*:

- $(\mathcal{F}, \bullet, I)$ is a monoid, representing sequential composition and purity;
- $(\mathcal{F}, \&, I)$ is a commutative monoid, representing parallel composition;
- letting $F + G = ?{+}(I, F, G)$ (pure guard) $(\mathcal{F}, +)$ is a semigroup, representing choice between branches
- with right-distributivity axioms:

Definition: jonoid

For a set of effects $\mathcal{F}$ then $(\mathcal{F}, \bullet, I, \&, ?\!+, \sqsubseteq)$ is a *joinoid control-flow algebra*:

- $(\mathcal{F}, \bullet, I)$ is a monoid, representing sequential composition and purity;
- $(\mathcal{F}, \&, I)$ is a commutative monoid, representing parallel composition;
- letting $F + G = ?\!+(I, F, G)$ (pure guard) $(\mathcal{F}, +)$ is a semigroup, representing choice between branches
- with right-distributivity axioms:

$$(F + G) \bullet H = (F \bullet H) + (G \bullet H)$$
$$(F + G) \,\&\, H = (F \,\&\, H) + (G \,\&\, H)$$

Definition: jonoid

For a set of effects $\mathcal{F}$ then $(\mathcal{F}, \bullet, I, \&, ?{+}, \sqsubseteq)$ is a *joinoid control-flow algebra*:

- $(\mathcal{F}, \bullet, I)$ is a monoid, representing sequential composition and purity;
- $(\mathcal{F}, \&, I)$ is a commutative monoid, representing parallel composition;
- letting $F + G = ?{+}(I, F, G)$ (pure guard) $(\mathcal{F}, +)$ is a semigroup, representing choice between branches
- with right-distributivity axioms:

$$(F + G) \bullet H = (F \bullet H) + (G \bullet H)$$
$$(F + G) \,\&\, H = (F \,\&\, H) + (G \,\&\, H)$$

- all operations are monotonic with respect to $\sqsubseteq$.

Graded generalised joinad

Definition: graded conditional joinad

Given a jonoid $(\mathcal{F}, \bullet, I, \&, ?\!+, \sqsubseteq)$:

Graded generalised joinad

Definition: graded conditional joinad

Given a jonoid $(\mathcal{F}, \bullet, I, \&, ?\!+, \sqsubseteq)$:

- graded monad for the pre-ordered monoid $(\mathcal{F}, \bullet, I, \sqsubseteq)$

Graded generalised joinad

Definition: graded conditional joinad

Given a jonoid $(\mathcal{F}, \bullet, I, \&, ?\!+, \sqsubseteq)$:

- graded monad for the pre-ordered monoid $(\mathcal{F}, \bullet, I, \sqsubseteq)$
- additional operations:

Graded generalised joinad

Definition: graded conditional joinad

Given a jonoid $(\mathcal{F}, \bullet, I, \&, ?{+}, \sqsubseteq)$:

- graded monad for the pre-ordered monoid $(\mathcal{F}, \bullet, I, \sqsubseteq)$
- additional operations:

$$\begin{aligned} \mathsf{cond}_{F,G,H,A} &: \mathsf{T}_F \mathbb{B} \times \mathsf{T}_G A \times \mathsf{T}_H A \to \mathsf{T}_{?{+}(F,G,H)} A \\ \mathsf{par}_{F,G,A} &: \mathsf{T}_F A \times \mathsf{T}_G A \to \mathsf{T}_{F \& G} A \end{aligned}$$

Graded generalised joinad

Definition: graded conditional joinad

Given a jonoid $(\mathcal{F}, \bullet, I, \&, ?+, \sqsubseteq)$:

- graded monad for the pre-ordered monoid $(\mathcal{F}, \bullet, I, \sqsubseteq)$
- additional operations:

$$\begin{aligned} \mathsf{cond}_{F,G,H,A} &: \mathsf{T}_F\mathbb{B} \times \mathsf{T}_G A \times \mathsf{T}_H A \to \mathsf{T}_{?+(F,G,H)} A \\ \mathsf{par}_{F,G,A} &: \mathsf{T}_F A \times \mathsf{T}_G A \to \mathsf{T}_{F\&G} A \end{aligned}$$

- Satisfy jonoid axioms (modulo lifting to functors)

Graded generalised joinad

Definition: graded conditional joinad

Given a jonoid $(\mathcal{F}, \bullet, I, \&, ?\!+, \sqsubseteq)$:

- graded monad for the pre-ordered monoid $(\mathcal{F}, \bullet, I, \sqsubseteq)$
- additional operations:

$$\begin{aligned} \mathsf{cond}_{F,G,H,A} &: \mathsf{T}_F \mathbb{B} \times \mathsf{T}_G A \times \mathsf{T}_H A \to \mathsf{T}_{?+(F,G,H)} A \\ \mathsf{par}_{F,G,A} &: \mathsf{T}_F A \times \mathsf{T}_G A \to \mathsf{T}_{F \& G} A \end{aligned}$$

- Satisfy jonoid axioms (modulo lifting to functors) where $?\!+ = \mathsf{cond}$ and $\& = \mathsf{par}$

Monoids are to (graded) monads as jonoids are to (graded) joinads

Theorem: soundness

Given a graded joinadic semantics for the simply-effect-and-typed λ-calculus with if and par then, for all e, e', Γ, τ, F:

Theorem: soundness

Given a graded joinadic semantics for the simply-effect-and-typed λ-calculus with if and par then, for all e, e', Γ, τ, F:

$$\Gamma \vdash e \equiv e' : \tau, F \quad \Rightarrow \quad [\![\Gamma \vdash e : \tau, F]\!] = [\![\Gamma \vdash e' : \tau, F]\!]$$

Theorem: soundness

Given a graded joinadic semantics for the simply-effect-and-typed λ-calculus with if and par then, for all e, e', Γ, τ, F:

$$\Gamma \vdash e \equiv e' : \tau, F \quad \Rightarrow \quad [\![\Gamma \vdash e : \tau, F]\!] = [\![\Gamma \vdash e' : \tau, F]\!]$$

wrt. CBV β-$\equiv$ with additional equations:

Theorem: soundness

Given a graded joinadic semantics for the simply-effect-and-typed λ-calculus with if and par then, for all e, e', Γ, τ, F:

$$\Gamma \vdash e \equiv e' : \tau, F \quad \Rightarrow \quad [\![\Gamma \vdash e : \tau, F]\!] = [\![\Gamma \vdash e' : \tau, F]\!]$$

wrt. CBV β-$\equiv$ with additional equations:

$$
\begin{array}{rl}
(\text{IF}\beta 1') & \mathsf{if\,true\,then}\; e \;\mathsf{else}\; x \equiv e \\
(\text{IF}\beta 2') & \mathsf{if\,false\,then}\; x \;\mathsf{else}\; e' \equiv e' \\
(\text{IF-DIST-PAR}) & (\mathsf{if}\; b \;\mathsf{then}\; e \;\mathsf{else}\; e')\;\mathsf{par}\; e'' \\
& \quad \equiv \mathsf{if}\; b \;\mathsf{then}\; (e \;\mathsf{par}\; e'') \;\mathsf{else}\; (e' \;\mathsf{par}\; e'') \\
(\text{IF-DIST-SEQ}) & \mathsf{let}\; x = (\mathsf{if}\; e \;\mathsf{then}\; e' \;\mathsf{else}\; e'') \;\mathsf{in}\; e''' \\
& \quad \equiv \mathsf{if}\; e \;\mathsf{then}\; (\mathsf{let}\; x = e' \;\mathsf{in}\; e''') \;\mathsf{else}\; (\mathsf{let}\; x = e'' \;\mathsf{in}\; e''') \\
(\text{PAR-PURE}) & x \;\mathsf{par}\; e \equiv (x, e) \\
(\text{PAR-SYM}) & e \;\mathsf{par}\; e' \equiv \mathsf{swap}\; (e' \;\mathsf{par}\; e) \\
(\text{PAR-ASSOC}) & e \;\mathsf{par}\; (e' \;\mathsf{par}\; e'') \equiv \mathsf{assoc}\; ((e \;\mathsf{par}\; e') \;\mathsf{par}\; e'')
\end{array}
$$

Conclusions

Conclusions

- unify Hanne and Flemming's rich effects with semantics

Conclusions

- unify Hanne and Flemming's rich effects with semantics

$$\text{monads} \xrightarrow{\text{indexing}} \text{graded monads} \xrightarrow{\text{non-seq. control}} \text{graded joinads}$$

Conclusions

- unify Hanne and Flemming's rich effects with semantics

$$\text{monads} \xrightarrow{\text{indexing}} \text{graded monads} \xrightarrow{\text{non-seq. control}} \text{graded joinads}$$

- Semantics for various kinds of parallel, concurrent (e.g. music) and speculative behaviour (e.g. prefetching)

Conclusions

- unify Hanne and Flemming's rich effects with semantics

  $$\text{monads} \xrightarrow{\text{indexing}} \text{graded monads} \xrightarrow{\text{non-seq. control}} \text{graded joinads}$$

- Semantics for various kinds of parallel, concurrent (e.g. music) and speculative behaviour (e.g. prefetching)

- Considerable simplification to proofs

Conclusions

- unify Hanne and Flemming's rich effects with semantics

$$\text{monads} \xrightarrow{\text{indexing}} \text{graded monads} \xrightarrow{\text{non-seq. control}} \text{graded joinads}$$

- Semantics for various kinds of parallel, concurrent (e.g. music) and speculative behaviour (e.g. prefetching)
- Considerable simplification to proofs
- Represent effect-dependent optimisations more easily

Conclusions

- unify Hanne and Flemming's rich effects with semantics

$$\text{monads} \xrightarrow{\text{indexing}} \text{graded monads} \xrightarrow{\text{non-seq. control}} \text{graded joinads}$$

- Semantics for various kinds of parallel, concurrent (e.g. music) and speculative behaviour (e.g. prefetching)
- Considerable simplification to proofs
- Represent effect-dependent optimisations more easily
- Effect-directed semantics provides *co-design* approach

Conclusions

- unify Hanne and Flemming's rich effects with semantics

$$\text{monads} \xrightarrow{\text{indexing}} \text{graded monads} \xrightarrow{\text{non-seq. control}} \text{graded joinads}$$

- Semantics for various kinds of parallel, concurrent (e.g. music) and speculative behaviour (e.g. prefetching)
- Considerable simplification to proofs
- Represent effect-dependent optimisations more easily
- Effect-directed semantics provides *co-design* approach
  - Equations of analysis carry over to semantics, and vice versa

Conclusions

- unify Hanne and Flemming's rich effects with semantics

  $$\text{monads} \xrightarrow{\text{indexing}} \text{graded monads} \xrightarrow{\text{non-seq. control}} \text{graded joinads}$$

- Semantics for various kinds of parallel, concurrent (e.g. music) and speculative behaviour (e.g. prefetching)
- Considerable simplification to proofs
- Represent effect-dependent optimisations more easily
- Effect-directed semantics provides *co-design* approach
  - Equations of analysis carry over to semantics, and vice versa
  - Exposes which structure is needed in each direction

Thanks Hanne and Flemming for the inspiration.
Happy Birthday!

Thanks to Sam Aaron (Cambridge) for the Sonic Pi language used for the intro program

Backup slides

Modeling effects with *monads*

Model effectful computations via some data type T

Modeling effects with *monads*

Model effectful computations via some data type T

$$[\![\Gamma \vdash e : \tau]\!] : [\![\Gamma]\!] \to \mathsf{T}[\![\tau]\!]$$

Modeling effects with *monads*

Model effectful computations via some data type T

$$[\![\Gamma \vdash e : \tau]\!] : [\![\Gamma]\!] \to \mathsf{T}[\![\tau]\!]$$

e.g. for state $\mathsf{T}A = S \to (A \times S)$.

Modeling effects with *monads*

Model effectful computations via some data type T

$$\llbracket \Gamma \vdash e : \tau \rrbracket : \llbracket \Gamma \rrbracket \to \mathsf{T}\llbracket \tau \rrbracket$$

e.g. for state $\mathsf{T}A = S \to (A \times S)$.
e.g. for partiality $\mathsf{T}A = \bot + A$

Modeling effects with *monads*

Model effectful computations via some data type T

$$[\![\Gamma \vdash e : \tau]\!] : [\![\Gamma]\!] \to \mathsf{T}[\![\tau]\!]$$

e.g. for state $\mathsf{T}A = S \to (A \times S)$.
e.g. for partiality $\mathsf{T}A = \bot + A$

Monads provide sequential composition

Given $f : A \to \mathsf{T}B$ and $g : B \to \mathsf{T}C$ then

Modeling effects with *monads*

Model effectful computations via some data type T

$$[\![\Gamma \vdash e : \tau]\!] : [\![\Gamma]\!] \to \mathsf{T}[\![\tau]\!]$$

e.g. for state $\mathsf{T}A = S \to (A \times S)$.
e.g. for partiality $\mathsf{T}A = \bot + A$

Monads provide sequential composition

Given $f : A \to \mathsf{T}B$ and $g : B \to \mathsf{T}C$ then

$$g \mathbin{\hat{\circ}} f : A \to \mathsf{T}C$$

Modeling effects with *monads*

Model effectful computations via some data type T

$$[\![\Gamma \vdash e : \tau]\!] : [\![\Gamma]\!] \to \mathsf{T}[\![\tau]\!]$$

e.g. for state $\mathsf{T}A = S \to (A \times S)$.
e.g. for partiality $\mathsf{T}A = \bot + A$

Monads provide sequential composition

Given $f : A \to \mathsf{T}B$ and $g : B \to \mathsf{T}C$ then

$$g \,\hat{\circ}\, f : A \to \mathsf{T}C$$

with $\hat{id}_A : A \to \mathsf{T}A$.