Linear Types
for Computational Effects

Alex Simpson

LFCS, School of Informatics
University of Edinburgh, UK

Joint work with: Jeff Egger (LFCS, Edinburgh)
Rasmus Møgelberg (ITU, Copenhagen)
Computational effects

Computational effects are the non-functional aspects of computation, such as:

- nontermination
- nondeterminism
- probabilistic choice
- exceptions
- side effects
- input/output
- continuations
- resumptions, ...

A call-by-value program from input type $X$ to output type $Y$ is modelled by a function

$$X \rightarrow TY$$

where $T$ is a computational monad capturing whatever computational effects are present in the scenario [Moggi 1989]
Computational metalanguage [Moggi 1991]

Extend typed λ-calculus with new type constructor $T$,

\[ A, B, \ldots ::= \alpha | 1 | A \times B | A \rightarrow B | TA \]

Typing rules for $T$:

\[
\frac{\Gamma \vdash t : A}{\Gamma \vdash \lfloor t \rfloor : TA} \quad \frac{\Gamma \vdash t : TA \quad \Gamma, x : A \vdash u : TB}{\Gamma \vdash \text{let } x \leftarrow t \text{ in } u : TB}
\]

Model of computational metalanguage:

cartesian-closed category + strong monad $T$
Effect calculus

The $T$ constructor enjoys a more general elimination rule, by introducing notion of computation type:

\[
A, B, \ldots ::= \alpha \mid \alpha \mid 1 \mid A \times B \mid A \rightarrow B \mid TA \quad \text{(value types)} \\
A, B, \ldots ::= \alpha \mid 1 \mid A \times B \mid A \rightarrow B \mid TA \quad \text{(computation types)}
\]

\[
\frac{\Gamma \vdash t : A}{\Gamma \vdash [t] : TA} \quad \frac{\Gamma \vdash t : TA \quad \Gamma, x : A \vdash u : B}{\Gamma \vdash \text{let } x \leftarrow t \text{ in } u : B}
\]

Cf.: “effect PCF” [Filinski 1996]

“call by push value” (CBPV) [Levy 1999, 2004]
Cbv and cbn translations

The call-by-value (cbv) translation [Moggi] translates a simple type $\sigma$ to a value type $\sigma^{cbv}$.

The call-by-name (cbn) translation (cf. [Filinski, Levy]) translates $\sigma$ to a computation type $\sigma^{cbn}$.

$$
\begin{align*}
\alpha^{cbv} &= \alpha \\
1^{cbv} &= 1 && \quad \alpha^{cbn} = \alpha \\
(\sigma \times \tau)^{cbv} &= \sigma^{cbv} \times \tau^{cbv} \\
(\sigma \times \tau)^{cbn} &= \sigma^{cbn} \times \tau^{cbn} \\
(\sigma \to \tau)^{cbv} &= \sigma^{cbv} \to T \tau^{cbv} \\
(\sigma \to \tau)^{cbn} &= \sigma^{cbn} \to \tau^{cbn}.
\end{align*}
$$

Typing judgements $\Gamma \vdash t : \sigma$ get translated to:

$$
\begin{align*}
\Gamma^{cbv} \vdash t^{cbv} : T \sigma^{cbv} \\
\Gamma^{cbn} \vdash t^{cbn} : \sigma^{cbn}
\end{align*}
$$
Linear types for computational effects

Two kinds of motivation:

1. Logical
   
   Effect calculus versus intuitionistic linear logic (ILL)

2. Computational
   
   — linearly-used continuations

   — other instances of the linear usage of effects

   — interactions between linearity and effects more generally
Effect calculus versus ILL, cf. [Benton & Wadler 1996]

There is an inclusion (modulo reconfiguration):

\[
\text{Models of effect calculus} \supseteq \text{Models of ILL}
\]

This gives an interpretation

\[
\text{Effect calculus} \rightarrow \text{ILL}
\]

The interpretation induces a correspondence between translations of typed \(\lambda\)-calculus into the effect calculus and translations into ILL.

\[
\text{cbn} \cong \text{Girard translation} \\
\text{cbv} \cong \text{variant Girard translation}
\]
Loss of generality

Every model of ILL determines a model of the effect calculus with a commutative monad. Equivalently, the interpretation of the effect calculus in ILL validates:

\[
\text{let } x \leftarrow s \text{ in let } y \leftarrow t \text{ in } u = \text{ let } y \leftarrow t \text{ in let } x \leftarrow s \text{ in } u
\]

i.e., order of execution of effects is immaterial.

Commutative effects: nontermination, nondeterminism, probabilistic choice.

Non-commutative effects: exceptions, side effects, input/output, continuations, resumptions, combined probabilistic+nondeterministic choice, ...
“We do not know if it is possible to define a non-commutative linear calculus which corresponds to a wider class of monad models.”

[Benton & Wadler 1996]
Linearly-used continuations

Programs from $X$ to $Y$ with control operators are modelled as continuation transformers:

$$(Y \to R) \to (X \to R) \cong X \to (Y \to R) \to R$$

Cbv for the continuations monad $TY = (Y \to R) \to R$.

For many common structured forms of control the continuation transformers are linear [Berdine et al. 2002]:

$$(Y \to R) \multimap (X \to R) \cong X \to (Y \to R) \multimap R$$

Cbv for the linearly-used continuations monad $TY = (Y \to R) \multimap R$. 

Simple types $\sigma$ translate to ILL types $\sigma^R_v$ (cbv) and $\sigma^R_n$ (cbn):

\[
\begin{align*}
\alpha^R_v &= \alpha \\
1^R_v &= 1 \\
(\sigma \times \tau)^R_v &= \sigma^R_v \times \tau^R_v \\
(\sigma \rightarrow \tau)^R_v &= (\tau^R_v \rightarrow \mathbb{R}) \rightarrow (\sigma^R_v \rightarrow \mathbb{R})
\end{align*}
\]

\[
\begin{align*}
\alpha^R_n &= \alpha \rightarrow \circ \mathbb{R} \\
1^R_n &= 1 \\
(\sigma \times \tau)^R_n &= \sigma^R_n \times \tau^R_n \\
(\sigma \rightarrow \tau)^R_n &= \sigma^R_n \rightarrow \tau^R_n
\end{align*}
\]

Typing judgements $\Gamma \vdash t : \sigma$ translate to:

\[
\begin{align*}
\Gamma^R_v \mid - \vdash t^R_v : (\sigma^R_v \rightarrow \mathbb{R}) \rightarrow \circ \mathbb{R} \\
\Gamma^R_n \mid - \vdash t^R_n : \sigma^R_n
\end{align*}
\]

where $|$ separates intuitionistic context on left from linear context (empty above) on right, cf. DILL [Barber 1997]
Linearly-used effects

Linearly-used continuations offer one example of a more general phenomenon: the linear usage of effects [Hasegawa 2002]

Is ILL the best framework for modelling linearly-used effects?

Possible drawbacks of ILL:

— Commutativity equation
— Models of ILL not so ubiquitous

Natural alternative, start with effect calculus and add linear primitives to it.

By doing so, we obtain something more general than ILL: the enriched effect calculus
Linearity in effect calculus

Basic dichotomy.

— A value is a pervasive static object, it just is. We shall have no notion of linear usage of values.

— A computation is a dynamic entity. There is a natural intuition of linearity: the computation is performed exactly once.

Our treatment of linearity will thus be intimately tied up with the separation between value types and computation types.
Judgement forms

Have two judgement forms.

(i) $\Gamma |- t : A$

(ii) $\Gamma \mid z : A \vdash t : B$

$\Gamma$ is a context associating (value) types to variables.

The reading of judgement (ii) is: in executing the computation $t$, the computation $z$ is used linearly.

Alternatively, think of (ii) as identifying $t[z]$ as an evaluation context.

Both judgement forms are conveniently subsumed by $\Gamma \mid \Delta \vdash t : A$, where $\Delta$ is a stoup [Girard 1991]
Typing rules (effect calculus)

\[
\begin{array}{c}
\Gamma, x : A \vdash x : A \\
\Gamma \vdash \Delta \vdash t : A \\
\Gamma \vdash \Delta \vdash u : B \\
\Gamma \vdash \Delta \vdash \langle t, u \rangle : A \times B \\
\Gamma, x : A \vdash t : B \\
\Gamma \vdash \Delta \vdash \lambda x : A. t : A \to B \\
\Gamma \vdash t : A \\
\Gamma \vdash [t] : TA \\
\Gamma \vdash s : A \to B \\
\Gamma \vdash \Delta \vdash s(t) : B \\
\Gamma \vdash \Delta \vdash t : A \times B \\
\Gamma \vdash \Delta \vdash \lambda x : A. t : A \to B \\
\Gamma \vdash \Delta \vdash \lambda x : A. u : B \\
\Gamma, x : A \vdash u : B \\
\end{array}
\]

Closely related to CBPV with (complex) stacks [Levy 2005].
\( T \) versus \( ! \) (effect calculus)

The rules for \( T \) are instances of the standard rules for \( ! \) from linear logic (as in, e.g., DILL [Barber 1997])

\[
\frac{
\Gamma |- t : A
}{
\Gamma |- !t : !A
}
\]

\[
\frac{
\Gamma |- \Delta |- t : !A \quad \Gamma, x : A |- u : B
}{
\Gamma |- \Delta |- \text{let} \, !x \, \text{be} \, t \, \text{in} \, u : B
}
\]

We thus replace \( T \) by \( ! \) :

\[
A, B, \ldots ::= \alpha | \alpha | 1 | A \times B | A \rightarrow B | !A
\]

\[
A, B, \ldots ::= \alpha | 1 | A \times B | A \rightarrow B | !A
\]

N.B., Levy uses \( F \) (and \( U \)) instead of \( T \) or \( ! \)
Enriching effect calculus with ⊸

We add type constructor \( A \rightarrow B \) for linear function space between computation types.

\[
\frac{\Gamma \vdash z : A \quad \Gamma \vdash t : B}{\Gamma \vdash \lambda z : A. t : A \rightarrow B}
\]

\[
\frac{\Gamma \vdash s : A \rightarrow B \quad \Gamma, t : A \vdash B}{\Gamma, \Delta \vdash s[t] : B}
\]

There is no reason to have \( A \rightarrow B \) as a computation type itself. So, for example:

\[
(A \rightarrow B) \rightarrow C \quad \text{value type}
\]

\[
(A \rightarrow B) \rightarrow C \quad \text{computation type}
\]

\[
(A \rightarrow B) \rightarrow C \quad \text{prohibited}
\]

\[
A \rightarrow B \rightarrow C \quad \text{prohibited}
\]
Enriched effect calculus

Full calculus obtained by adding type constructors:

\[ A ::= \ldots | A \to B | !A \otimes B | 0 | A \oplus B \]

\[ A ::= \ldots | !A \otimes B | 0 | A \oplus B . \]

Remaining typing rules:

\[
\begin{align*}
\Gamma |- t : A & \quad \Gamma | \Delta |- u : B \\
\quad & \quad \quad \quad \Gamma | \Delta |- !t \otimes u : !A \otimes B \\
\Gamma | \Delta |- s : !A \otimes B & \quad \Gamma, x : A | z : B |- t : C \\
\quad & \quad \quad \quad \Gamma | \Delta |- \text{let} !x \otimes z \text{ be } s \text{ in } t : C \\
\Gamma | \Delta |- t : 0 & \\
\quad & \quad \quad \quad \Gamma | \Delta |- \text{image}(t) : A \\
\Gamma | \Delta |- t : A & \\
\quad & \quad \quad \quad \Gamma | \Delta |- \text{inl}(t) : A \oplus B \\
\Gamma | \Delta |- t : B & \\
\quad & \quad \quad \quad \Gamma | \Delta |- \text{inr}(t) : A \oplus B \\
\Gamma | \Delta |- s : A \oplus B & \quad \Gamma | x : A |- t : C & \quad \Gamma | y : B |- u : C \\
\quad & \quad \quad \quad \Gamma | \Delta |- \text{case } s \text{ of } (\text{inl}(x).t; \text{inr}(y).u) : C
\end{align*}
\]
Semantics

A model of the enriched effect calculus is given by:

— a cartesian-closed category $\mathcal{V}$,

— together with a $\mathcal{V}$-enriched category $\mathcal{C}$,

— where $\mathcal{C}$ has (specified) $\mathcal{V}$-enriched powers and copowers, and finite $\mathcal{V}$-enriched products and coproducts,

— together with a $\mathcal{V}$-enriched adjunction $F \dashv U : \mathcal{C} \to \mathcal{V}$.

There is a syntactic model built from types and terms (modulo equations).

**Theorem**  The syntactic model is initial (up to coherent natural isomorphism) w.r.t. structure preserving functors.
EC versus EEC versus ILL

We have inclusions (modulo reconfiguration):

\[
\text{Models of EC} \supseteq \text{Models of EEC} \supseteq \text{Models of ILL}
\]

**Theorem** Every model of EC has a full and faithful structure-preserving embedding into a model of EEC.

Thus EEC answers Benton and Wadler’s (implicit) question positively.

Syntactically:

\[
\text{EC} \subseteq \text{EEC} \subseteq \text{ILL}
\]

**Theorem** EEC is a conservative extension of EC.

N.B., ILL is *not* a conservative extension of EEC. (The inclusion is neither full nor faithful.)
Dual models

Given an EEC model $\mathcal{M} = (F \dashv U : \mathcal{C} \to \mathcal{V})$ and an object $R \in \mathcal{C}$, define the $R$-dual model:

$$\mathcal{M}^R := (R(\_ \_ \to \mathcal{C}(\_, R) : \mathcal{C}^{\text{op}} \to \mathcal{V})$$

N.B., this construction is a decomposition of Lawvere’s dual monad into an adjunction.

**Theorem** For any EEC model $\mathcal{M}$, there is an equivalence of EEC models between $\mathcal{M}$ and $\mathcal{M}^{R_{F1}}$.

(In particular, there is an equivalence between $\mathcal{M}$ and $\mathcal{M}^{F1_{F1}}$.)

N.B., models of EC and ILL are not closed under the dual model construction.
Let $\mathcal{M}_{\text{syn}}$ be the syntactic model for EEC with a distinguished computation type constant $R$.

By initiality, there is a unique (up to coherent natural isomorphism) map of models:

$$L: \mathcal{M}_{\text{syn}} \rightarrow \mathcal{M}_{\text{syn}}^R$$

satisfying:

$$L(\alpha) = \alpha$$

$$L(\alpha) = \alpha$$

$$L(R) = !1$$

**Theorem**  $L$ is an equivalence of models.

Thus the free model $\mathcal{M}_{\text{syn}}$ is self dual.
Syntactic version

Syntactically, \( L : \mathcal{M}_{\text{syn}} \rightarrow \mathcal{M}_{\text{syn}}^R \) amounts to a translation of EEC into itself.

A value type \( A \) translates to a value type \( A^{\mathcal{V}_R} \).

A computation type \( A \) translates to a computation type \( A^{\mathcal{C}_R} \).

These translations satisfy \( A^{\mathcal{V}_R} \simeq A^{\mathcal{C}_R} \circ R \).

A typing judgement \( \Gamma \vdash t : A \) translates to

\[
\Gamma^{\mathcal{V}_R} \vdash t^{\mathcal{V}_R} : A^{\mathcal{V}_R}
\]

A typing judgement \( \Gamma \vdash z : A \vdash u : B \) translates to

\[
\Gamma^{\mathcal{V}_R} \vdash z : B^{\mathcal{C}_R} \vdash u^{\mathcal{C}_R} : A^{\mathcal{C}_R}
\]
\[ \begin{align*}
\alpha^\nu R &= \alpha \\
\alpha^{\nu R} &= \alpha^{\mathcal{CR}} \to R \\
1^{\nu R} &= 1 \\
(A \times B)^{\nu R} &= A^{\nu R} \times B^{\nu R} \\
(A \to B)^{\nu R} &= A^{\nu R} \to B^{\nu R} \\
(!A)^{\nu R} &= (!A)^{\mathcal{CR}} \to R \\
(A \to B)^{\nu R} &= B^{\mathcal{CR}} \to A^{\mathcal{CR}} \\
(!A \otimes B)^{\nu R} &= (!A \otimes B)^{\mathcal{CR}} \to R \\
(0)^{\nu R} &= (0)^{\mathcal{CR}} \to R \\
(A \oplus B)^{\nu R} &= (A \oplus B)^{\mathcal{CR}} \to R \\
R^{\nu R} &= R \\
\alpha^{\mathcal{CR}} &= \alpha \\
1^{\mathcal{CR}} &= 0 \\
(A \times B)^{\mathcal{CR}} &= A^{\mathcal{CR}} \oplus B^{\mathcal{CR}} \\
(A \to B)^{\mathcal{CR}} &= !A^{\nu R} \otimes B^{\mathcal{CR}} \\
(!A)^{\mathcal{CR}} &= A^{\nu R} \to R \\
(!A \otimes B)^{\mathcal{CR}} &= A^{\nu R} \to B^{\mathcal{CR}} \\
(0)^{\mathcal{CR}} &= 1 \\
(A \oplus B)^{\mathcal{CR}} &= A^{\mathcal{CR}} \times B^{\mathcal{CR}} \\
R^{\mathcal{CR}} &= !1
\end{align*} \]
The self duality of the syntactic model manifests itself syntactically as the involutivity of the above translation.

Theorem (Involution property) We have isomorphisms

\[ A \rightarrow A^{\nu R\nu R} \quad A \rightarrow A^{CR\CR} \]

modulo which \( t = t^{\nu R\nu R} \) and \( u = u^{CR\CR} \).

Our proof of this is semantic, using self duality. Obtaining a syntactic proof looks like a formidable exercise.

Corollary The translations \((\cdot)^{\nu R}\) and \((\cdot)^{CR}\) are full and faithful.
Recovering linearly-used CPS translations

Theorem (cbv) There is an isomorphism \( \sigma^Rv \cong \sigma^{cbv}v^R \), modulo which we have \( t^Rv = t^{cbv}v^R \)

Corollary cf. [Hasegawa 2002] The cbv linearly-used CPS translation \((\cdot)^Rv\) gives a full and faithful translation from Moggi’s \(\lambda_v\)-calculus into EEC.

Theorem (cbn) There is an isomorphism \( \sigma^Rn \cong \sigma^{cbn}v^R \), modulo which we have \( t^Rn = t^{cbn}v^R \)

Corollary cf. [Hasegawa 2004] The cbn linearly-used CPS translation \((\cdot)^Rn\) gives a full and faithful translation from simply-typed \(\lambda_\beta\eta\)-calculus into EEC.

Hasegawa has the same results for ILL rather than EEC.
Another example: linearly-used state

Programs from $X$ to $Y$ with side effects are modelled as maps (where $S$ is set of all states):

$$X \times S \rightarrow Y \times S$$

Cbv for the side-effects monad $TY = S \rightarrow (X \times S)$.

In practice, state is typically used linearly (e.g., there is no snapback [O’Hearn and Reynolds 2000]), i.e., programs are maps:

$$!X \otimes S \rightarrowtail !Y \otimes S$$

Cbv for the linear side-effects monad $TY = S \rightarrowtail (!Y \otimes S)$.

This formalizable in ECC ($S$ must be computation type)
ECC via polymorphism [Møgelberg and S. 2007]

\[ A, B, \ldots ::= X \mid X \mid A \to B \mid A \multimap B \mid \forall X. A \mid \forall X. A \]

(A, B, \ldots ::= X \mid A \to B \mid \forall X. A \mid \forall X. A)

Relational parametricity allows one to recover types of ECC; e.g.:

\[ !A = \forall X. (A \to X) \to X \]

\[ 1 = \forall X. 0 \to X \]

\[ A \times B = \forall X. ((A \multimap X) + (B \multimap X)) \to X \]

\[ !A \otimes B = \forall X. (A \to B \multimap X) \to X \]

\[ 0 = \forall X. X \]

\[ A \oplus B = \forall X. (A \multimap X) \to (B \multimap X) \to X \]

Also obtain (co)inductive types and existential types.
Further directions

The story remains (essentially) unchanged if we include sums of value types as in Levy’s CBPV. (There are complications to some proofs.)

EEC combines nicely with Plotkin and Power’s theory of algebraic effects, cf. [Møgelberg and S. 2007]

Is it possible to include a symmetric premonoidal tensor $A \otimes B$ of computation types in EEC, instead of having the asymmetric $!A \otimes B$ as a primitive? If so, is this useful?

Can a linear typing be used for modelling single-threaded nondeterminism by considering free linear semilattices, where the choice operation has type: $A \times A \rightarrow A$?

Programming language (or other) applications?