Reasoning about effectful programs and evaluation order

Dylan McDermott

University of Cambridge

Joint work with Alan Mycroft
General framework for proving statements of the form

If <restriction on side-effects> then <evaluation order 1>

is equivalent to <evaluation order 2>

Examples:

- If there are no effects, then call-by-value is equivalent to call-by-name
- If the only effect is nontermination, then call-by-name is equivalent to call-by-need
- If the only effect is nondeterminism, then call-by-value is equivalent to call-by-need
Method

Use an intermediate language that supports various evaluation orders:

1. Translate from source language to intermediate language

\[ e \rightarrow \langle e \rangle^n \rightarrow \langle e \rangle^v \]

- \( e \): source expression
- \( \langle e \rangle^n \): call-by-name intermediate term
- \( \langle e \rangle^v \): call-by-value intermediate term

2. Prove contextual equivalence

\[ \langle e \rangle^n \cong_{ctx} \langle e \rangle^v \]
Method

Use an intermediate language that supports various evaluation orders:

1. Translate from source language to intermediate language

\[ e \rightarrow \langle e \rangle^n \rightarrow \langle e \rangle^v \]

Source expression \( e \) \( \xrightarrow{\text{call-by-name}} \) \( \langle e \rangle^n \) \( \xrightarrow{\text{call-by-value}} \) \( \langle e \rangle^v \)

2. Prove contextual equivalence

\[ \phi(\langle e \rangle^n) \approx_{\text{ctx}} \langle e \rangle^v \]

Subtlety: two translations have different types

\[ \langle e \rangle^n \rightarrow \phi(\langle e \rangle^n) \]

Another intermediate term
Outline

How do we prove evaluation order equivalences (assuming **global** restriction on side-effects)?
- When are call-by-value and call-by-name equivalent?

How do we do call-by-need?
- New intermediate language: extension of Levy’s call-by-push-value to capture **call-by-need**
- Example: name and need are equivalent if only effect is nontermination

How do we do **local** (per expression) restrictions?
Call-by-push-value [Levy ’99]

Split syntax into **values** and **computations**

- Values don’t have side-effects, computations might
Call-by-push-value [Levy ’99]

Split syntax into **values** and **computations**

- Values don’t have side-effects, computations might

Not:

- Values don’t reduce, computations might (complex values)
- Value types are call-by-value, computations types are call-by-name
Call-by-push-value [Levy ’99]

- Can put two computations together: if $M_1, M_2$ are computations then
  
  $M_1$ to $x$. $M_2$

  is a computation

- Can thunk computations: if $M$ is a computation then
  
  thunk $M$

  is a value

$\Rightarrow$ can do call-by-value and call-by-name (but not call-by-need)
Call-by-push-value syntax

Value types:

\[ A, B ::= \ldots \]

\[ \mid U \mid C \]

Computation types:

\[ C, D ::= \ldots \]

\[ \mid A \to C \]

\[ \mid FA \]

Value terms:

\[ V, W ::= c \mid \ldots \]

\[ \mid \text{thunk } M \]

\[ \mid x \]

Computation terms:

\[ M, N ::= \ldots \]

\[ \mid \lambda x. M \mid V \downarrow M \]

\[ \mid \text{return } V \mid M_1 \to x. M_2 \]

\[ \mid \text{force } V \]

constants, products, etc.

thunks

products, etc.

functions

returners
Call-by-push-value syntax

Value types:

\[ A, B ::= \ldots \]
\[ | \text{UC} \]

Value terms:

\[ V, W ::= c \mid \ldots \]
\[ \text{constants, products, etc.} \]
\[ | \text{thunk } M \]
\[ | x \]

Computation types:

\[ C, D ::= \ldots \]
\[ | A \rightarrow C \]
\[ | FA \]

Computation terms:

\[ M, N ::= \ldots \]
\[ \text{products, etc.} \]
\[ | \lambda x. M \mid V \cdot M \]
\[ | \text{return } V \mid M_1 \text{ to } x. M_2 \]
\[ | \text{force } V \]

\[ \Gamma \vdash M : C \]
\[ \frac{\Gamma \vdash \text{thunk } M : \text{UC}}{\Gamma \vdash \text{thunk } M : \text{UC}} \]
\[ \frac{\Gamma \vdash V : \text{UC}}{\Gamma \vdash \text{force } V : C} \]
Call-by-push-value syntax

Value types:

\[ A, B ::= \ldots \]

| \text{U} \mathcal{C} |

Computation types:

\[ \mathcal{C}, \mathcal{D} ::= \ldots \]

| \mathcal{A} \to \mathcal{C} |

| \text{F} \mathcal{A} |

Value terms:

\[ V, W ::= c \mid \ldots \] constants, products, etc.

| \text{thunk} \ M |

| \ x |

Computation terms:

\[ M, N ::= \ldots \] products, etc.

| \lambda x. \ M | \ V \ \ M |

| \ \text{return} \ V | \ M_1 \ \text{to} \ x. \ M_2 |

| \ \text{force} \ V |

Typing contexts:

\[ \Gamma ::= \Diamond \mid x : \ A \]
Call-by-push-value syntax

Value types:

\[ A, B ::= \ldots \]

| \text{U} \text{C} |

Computation types:

\[ C, D ::= \ldots \]

| \text{A} \rightarrow C |

| FA |

Value terms:

\[ V, W ::= c \mid \ldots \text{constants, products, etc.} \]

| \text{thunk} M |

| x |

Computation terms:

\[ M, N ::= \ldots \text{products, etc.} \]

| \lambda x. M \mid V \cdot M |

| \text{return} V \mid M_1 \text{ to } x. M_2 |

| \text{force} V |

\[ \Gamma \vdash V : A \]

\[ \Gamma \vdash \text{return} V : FA \]

\[ \Gamma, x : A \vdash M_2 : C \]

\[ \Gamma \vdash M_1 \text{ to } x. M_2 : C \]
We also have an equational theory

\[ V \equiv V' \quad M \equiv M' \]

Use this to define contextual equivalence

\[ M \cong_{\text{ctx}} M' \]

iff

\[ C[M] \equiv C[M'] \]

for all closed \( C \) of type \( FG \), where \( G \) doesn’t contain thunks
Call-by-value and call-by-name

\[(e)^v \cong_{\text{ctx}} (e)^n\]
Call-by-value and call-by-name

Source language types:

\[ \tau ::= \text{unit} \mid \text{bool} \mid \tau \to \tau' \]

Translations from value and name into CBPV:

\[ \tau \mapsto \text{value type } (\langle \tau \rangle^v) \quad \tau \mapsto \text{computation type } (\langle \tau \rangle^n) \]

\[
\begin{align*}
\text{unit} & \mapsto \text{unit} & \text{unit} & \mapsto F\text{ unit} \\
\text{bool} & \mapsto \text{bool} & \text{bool} & \mapsto F\text{ bool} \\
(\tau \to \tau') & \mapsto U(\langle \tau \rangle^v \to F\langle \tau' \rangle^v) & (\tau \to \tau') & \mapsto ((U\langle \tau \rangle^n) \to \langle \tau' \rangle^n) \\
\end{align*}
\]

\[
\begin{align*}
\Gamma, x : \tau & \mapsto \langle \Gamma \rangle^v, x : \langle \tau \rangle^v & \Gamma, x : \tau & \mapsto \langle \Gamma \rangle^n, x : U\langle \tau \rangle^n \\
\Gamma \vdash e : \tau & \mapsto \langle \Gamma \rangle^v \vdash \langle e \rangle^v : F\langle \tau \rangle^v & \Gamma \vdash e : \tau & \mapsto \langle \Gamma \rangle^n \vdash \langle e \rangle^n : \langle \tau \rangle^n \\
\end{align*}
\]
Call-by-value and call-by-name

\[
\begin{align*}
\langle \Gamma \rangle^v & \xrightarrow{(e)^v} F(\langle \tau \rangle)^v \\
\langle \Gamma \rangle^n & \xrightarrow{(e)^n} (\langle \tau \rangle)^n \\
\end{align*}
\]
Call-by-value and call-by-name

Isomorphism between call-by-value and call-by-name computations?

\[ \Gamma \vdash M : F(\tau)^v \quad \leftrightarrow \quad \Gamma \vdash \Phi_\tau M : (\tau)^n \]

\[ \Gamma \vdash N : (\tau)^n \quad \leftrightarrow \quad \Gamma \vdash \Psi_\tau N : F(\tau)^v \]
Isomorphism between call-by-value and call-by-name computations?

\[
\begin{align*}
\Gamma \vdash M : F(\tau)v & \iff \Gamma \vdash \Phi_\tau M : (\tau)^n \\
\Gamma \vdash N : (\tau)^n & \iff \Gamma \vdash \Psi_\tau N : F(\tau)v
\end{align*}
\]

Value to Name to Value:

\[
\Psi_\tau(\Phi_\tau(\text{return } V)) \equiv \text{return } V
\]

The other way depends on the effects
Logical relations for CBPV

value types $A$ $\leftrightarrow$ relations $\mathcal{R}[[A]]$ on closed terms $V : A$

computation types $C$ $\leftrightarrow$ relations $\mathcal{R}[[C]]$ on closed terms $M : C$

We’ll want

$$(M, M') \in \mathcal{R}[[FG]] \Rightarrow M \equiv M'$$

for ground types $G$ (to prove contextual equivalence)
Logical relations for CBPV

Assume:

- Defined in usual way on type formers excluding \( F \)

\[
\begin{align*}
\mathcal{R}[\text{UC}] &= \{(\text{thunk } M, \text{thunk } M') \mid (M, M') \in \mathcal{R}[\text{C}]\} \\
\mathcal{R}[A \to \text{C}] &= \{(M, M') \mid \forall (V, V') \in \mathcal{R}[A]. (V'M, V''M') \in \mathcal{R}[\text{C}]\}
\end{align*}
\]

- Closed under \text{return}: 

\[(V, V') \in \mathcal{R}[A] \implies (\text{return } V, \text{return } V') \in \mathcal{R}[F A]\]

- Closed under \text{to}: if \( x : A \vdash N, N' : \text{C} \) and

\[
(M, M') \in \mathcal{R}[F A] \quad \forall (V, V') \in \mathcal{R}[A]. (N[x \mapsto V], N'[x \mapsto V'])
\]

then

\[
(M \text{ to } x. N, M' \text{ to } x. N') \in \mathcal{R}[\text{C}]
\]

- Constants related to themselves: if \( c : A \) then \((c, c) \in \mathcal{R}[A] \)

- Transitivity
Lemma (Fundamental)

If $x_1 : A_1, \ldots, x_n : A_n \vdash M : C$ and $(V_i, V'_i) \in \mathcal{R}[\llbracket A_i \rrbracket]$ for each $i$ then

$$(M[x_1 \mapsto V_1, \ldots, x_n \mapsto V_n], M[x_1 \mapsto V'_1, \ldots, x_n \mapsto V'_n]) \in \mathcal{R}[\llbracket C \rrbracket]$$
From Name to Value and back

Definition (Thunkable [Führmann '99])
A computation $\Gamma \vdash M : FA$ is thunkable if

$$M \text{ to } x. \text{return (thunk (return } x)) \quad \text{and} \quad \text{return (thunk } M)$$

are related by $R[[F(U(FA))]]$.

This implies:

$$M \text{ to } x. \text{thunk (return } x) \sim N \quad \text{related to} \quad \text{thunk } M \sim N$$

Lemma

*If everything is thunkable and $M : (|\tau|^n$ then

$$(\Phi_\tau(\Psi_\tau M)) \sim_R [(|\tau|^n)] \sim M$$
The equivalence

Want to show that

$$\text{(|}\Gamma\text{)}^v \xrightarrow{(e)^v} \text{F(|}\tau\text{)}^v \xleftarrow{\approx_{\text{ctx}}} \text{(|}\Gamma\text{)}^n \xrightarrow{(e)^n} \text{(|}\tau\text{)}^n$$

Meaning:

$$\text{(|}e\text{)}^v \approx_{\text{ctx}} \Psi_B \left( (|e\rangle^n\left[ \begin{array}{c} x_1 \mapsto \text{thunk} (\Phi_{A_1} (\text{return } x_1)) \\ \ldots \\ x_n \mapsto \text{thunk} (\Phi_{A_n} (\text{return } x_n)) \end{array} \right] \right)$$

In particular, for closed $e$ of ground type ($\text{unit}$ or $\text{bool}$):

$$\text{(|}e\text{)}^v \equiv (|e\rangle^n$$
The equivalence

Lemma

Suppose everything is thunkable. If \(x_1 : A_1, \ldots, x_n : A_n \vdash e : A\) and \(V_i\) related to \(V_i'\) for each \(i\) then

\[
\langle e \rangle^V [x_1 \mapsto V_1, \ldots, x_n \mapsto V_n]
\]

is related to

\[
\Psi_B \left( \langle e \rangle^n \left[ \begin{array}{c}
x_1 \mapsto \text{thunk} (\Phi_{A_1}(\text{return } V_1')) \\
\vdots \\
x_n \mapsto \text{thunk} (\Phi_{A_n}(\text{return } V_n'))
\end{array} \right] \right)
\]
A trivial example

For no side-effects:

$$\mathcal{R}[FA] = \{(\text{return } V, \text{return } V') \mid (V, V') \in \mathcal{R}[A]\}$$
A non-example

Read-only state

code

get : F bool

get to x. return () \equiv return ()

get to x. get to y. return (x, y) \equiv get to z. return (z, z)

Logical relation:

\[ R[FA] = \left\{ (\text{get to } x. \text{ if } x \text{ then return } V_1 \text{ else return } V_2, \text{get to } x. \text{ if } x \text{ then return } V_1' \text{ else return } V_2') \mid (V_1, V_1'), (V_2, V_2') \in R[A] \right\} \]

Not all computations are thunkable!

- All thunkable computations have the form

  \[ \text{return } V \]
Goal

General framework for proving statements of the form

\[
\text{If } <\text{restriction on side-effects}> \text{ then } <\text{evaluation order 1}> \text{ is equivalent to } <\text{evaluation order 2}>
\]

Examples:

▶ If there are no effects, then call-by-value is equivalent to call-by-name
▶ If the only effect is nontermination, then call-by-name is equivalent to call-by-need
▶ If the only effect is nondeterminism, then call-by-value is equivalent to call-by-need
Extended call-by-push-value (ECBPV)

New computation forms:

\[ M, N ::= \ldots \]

| \_x \_ | \_ computation variables
| \_M_1 need \_x. \_M_2 \_ | \_call-by-need sequencing

Typing:

\[ \Gamma ::= \ldots | \_x : FA \]

\[
\frac{(\_x : FA) \in \Gamma}{\Gamma \vdash \_x : FA}
\frac{\Gamma \vdash M_1 : FA}{\Gamma, \_x : FA \vdash M_2 : C}
\frac{\Gamma \vdash \_x : FA}{\Gamma \vdash M_1 \_need \_x. \_M_2 : C}
\]
Extended call-by-push-value

Important equation:

\[ M_1 \text{ need } x. x \text{ to } y. M_2 \equiv M_1 \text{ to } y. M_2[x \mapsto \text{return } y] \]

Associativity:

\( (M_1 \text{ to } x. M_2) \text{ to } y. M_3 \equiv M_1 \text{ to } x. (M_2 \text{ to } y. M_3) \)

\( (M_1 \text{ need } x. M_2) \text{ need } y. M_3 \equiv M_1 \text{ need } x. (M_2 \text{ need } y. M_3) \)

\( (M_1 \text{ need } x. M_2) \text{ to } y. M_3 \equiv M_1 \text{ need } x. (M_2 \text{ to } y. M_3) \)

\( (M_1 \text{ to } x. M_2) \text{ need } y. M_3 \not\equiv M_1 \text{ to } x. (M_2 \text{ need } y. M_3) \)
Extended call-by-push-value

Given

\[ \Gamma \vdash M_1 : \text{FA} \quad \Gamma, x : \text{FA} \vdash M_2 : C \]

have various evaluation orders:

- **Call-by-value**: \( M_1 \text{ value } x. M_2 \equiv M_1 \text{ to } y. M_2[x \leftarrow \text{return } y] \)
- **Call-by-name**: \( M_1 \text{ name } x. M_2 \equiv M_2[x \leftarrow M_1] \)
- **Call-by-need**: \( M_1 \text{ need } x. M_2 \) (builtin)
Call-by-need translation

\[
\begin{align*}
\tau & \mapsto \text{value type } (|\tau|)^\text{need} \\
\text{unit} & \mapsto \text{unit} \\
\text{bool} & \mapsto \text{bool} \\
(\tau \rightarrow \tau') & \mapsto U\left(U(F(|\tau|)^\text{need}) \rightarrow F(|\tau'|)^\text{need}\right) \\
\Gamma, x : \tau & \mapsto (|\Gamma|)^\text{need}, \; x : F(|\tau|)^\text{need}
\end{align*}
\]
Call-by-need translation

\[
\begin{align*}
\tau & \mapsto \text{value type } (|\tau|)_{\text{need}} \\
\text{unit} & \mapsto \text{unit} \\
\text{bool} & \mapsto \text{bool} \\
(\tau \rightarrow \tau') & \mapsto \text{U}[\text{U}(\text{F}(\tau)_{\text{need}}) \rightarrow \text{F}(\tau')_{\text{need}}] \\
\Gamma, x : \tau & \mapsto (|\Gamma|)_{\text{need}}, \; x : \text{F}(\tau)_{\text{need}}
\end{align*}
\]

This could also be call-by-name!
Call-by-need translation

\[ \Gamma \vdash e : \tau \quad \Rightarrow \quad \langle \Gamma \rangle^{\text{need}} \vdash \langle e \rangle^{\text{need}} : F \langle \tau \rangle^{\text{need}} \]

\[ e \ e' \]

\[ \langle e \rangle^{\text{need}} \text{ to } f \cdot (\text{thunk} \ (\langle e' \rangle^{\text{need}})) \quad \text{‘ (force } f \text{’)} \]

\[ \lambda x. \ e \]

\[ \text{return (thunk (\lambda x'. (force x') need} \ x. \ (\langle e \rangle^{\text{need}}))) \]

Two nice properties:

- Applying lambdas

\[ \langle (\lambda x. \ e) \ e' \rangle^{\text{need}} \equiv \langle e' \rangle^{\text{need}} \text{ need } x. \ (\langle e \rangle^{\text{need}}) \]

- Translation is sound (wrt small-step operational semantics)

\[ e^{\text{need}} \Rightarrow e' \quad \Rightarrow \quad \langle e \rangle^{\text{need}} \equiv \langle e' \rangle^{\text{need}} \]

[Ariola & Felleisen '97]
Proving an equivalence

If the only effect is nontermination, call-by-name is equivalent to call-by-need

Method:
1. Instantiate ECBPV: add constants that induce diverging computations \( \Omega_C \)
2. Prove internal equivalence:

\[
M_1 \text{name } x. M_2 \quad \cong_{\text{ctx}} \quad M_1 \text{ need } x. M_2
\]

3. Corollary:

\[
(e) \text{\moggi } \cong_{\text{ctx}} (e) \text{\need}
\]
Internal equivalence: proof idea

\[ M_1 \text{ name } x. M_2 \equiv_{\text{ctx}} M_1 \text{ need } x. M_2 \]

Proof: use logical relations

- Reasoning about to:
  \[ \Omega_{FA} \text{ to } x. M_2 \equiv \Omega_C \quad \text{return } V \text{ to } x. M_2 \equiv M_2[x \mapsto V] \]

- Don’t have similar equations for need:
  \[ \Omega_{FA} \text{ need } x. M_2 \not\equiv \Omega_C \]

- Relate open terms: Kripke logical relations of varying arity
  [Jung and Tiuryn ’93]

\[ \mathcal{R}[A] \Gamma \subseteq \text{Term}_A^\Gamma \times \text{Term}_A^\Gamma \]
Global restriction on side-effects

If whole language restricted to nontermination, then

\[ M_1 \text{name } x. \ M_2 \equiv_{ctx} M_1 \text{ need } x. \ M_2 \]
Local restriction on side-effects

If whole-language $M_1$ restricted to nontermination, then

$$M_1 \text{name } x. \ M_2 \equiv_{\text{ctx}} \ M_1 \text{ need } x. \ M_2$$
Effect system for (E)CBPV

Goal: place upper bound on side-effects of computations

- Replace returner types $FA$ with $\langle \varepsilon \rangle A$
- Track effects $\varepsilon \subseteq \Sigma$

$$\Sigma := \{ \text{diverge, get, put, raise, } \ldots \}$$

$$\Omega : \langle \{\text{diverge}\} \rangle A \quad \text{get} : \langle \{\text{get}\} \rangle \text{bool} \quad \ldots$$

- Internal equivalence (with effect system):
  If $M_1 : \langle \varepsilon \rangle A$ for $\varepsilon \subseteq \{\text{diverge}\}$, then

$$M_1 \text{name }x. M_2 \quad \equiv_{\text{ctx}} \quad M_1 \text{ need }x. M_2$$
Effect system for (E)CBPV

\[
\Gamma \vdash M : C \quad \text{(Subtyping) } C <: D
\]
\[
\Gamma \vdash M : D
\]
\[
\langle \varepsilon \rangle A <: \langle \varepsilon' \rangle B \text{ if } \varepsilon \subseteq \varepsilon' \text{ and } A <: B
\]
\[
\Gamma \vdash M_1 : \langle \varepsilon \rangle A
\]
\[
\Gamma, x : A \vdash M_2 : C
\]
\[
\Gamma \vdash M_1 \text{ to } x. M_2 : \langle \varepsilon \rangle C
\]
\[
\langle \varepsilon \rangle (\langle \varepsilon' \rangle A) := \langle \varepsilon \cup \varepsilon' \rangle A
\]
\[
\langle \varepsilon \rangle (A \to C) := A \to \langle \varepsilon \rangle C
\]
Overview

How to prove an equivalence between evaluation orders:

1. Translate from source language to intermediate language
2. Prove contextual equivalence

\[
\begin{array}{c}
\langle \Gamma \rangle^v \quad (e)^v \\
\downarrow \quad \downarrow
\end{array}
\Rightarrow
\begin{array}{c}
F(\langle \tau \rangle)^v \\
\uparrow \quad \uparrow
\end{array}
\]

\[
\begin{array}{c}
\langle \Gamma \rangle^n \quad (e)^n \\
\downarrow \quad \downarrow
\end{array}
\Leftrightarrow_{\text{ctx}}
\begin{array}{c}
\langle \tau \rangle^n \\
\uparrow \quad \uparrow
\end{array}
\]

- Works for call-by-value, call-by-name
  - Call-by-need using extended call-by-push-value
- Also works for local restrictions on side-effects using an effect system
A slightly less trivial example

C-style undefined behaviour

\[ \text{undef}_C \preceq M \quad \text{undef}_{FA} \text{ to } x. M \equiv \text{undef}_C \]

Logical relation:

\[ \mathcal{R}[FA] := \{(\text{return } V, \text{return } V') \mid (V, V') \in \mathcal{R}[A]\} \]
\[ \cup \{(\text{undef}_{FA}, M)\} \]

Can replace value with name (but not name with value)