Reasoning about effectful programs and evaluation order

Dylan McDermott

University of Cambridge

Joint work with Alan Mycroft
Goal

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

   \[ e \overset{\text{call-by-name intermediate term}}{\rightarrow} \langle e \rangle^n \]

   \[ e \overset{\text{call-by-value intermediate term}}{\rightarrow} \langle e \rangle^v \]

   source expression

2. Prove contextual equivalence

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

Use an intermediate language that supports various evaluation orders:

1. Translate from source language to intermediate language

   ![Diagram](source-expression-to-intermediate-language)

   - Call-by-value intermediate term
   - Call-by-name intermediate term

2. Prove contextual equivalence

   \[ \phi(|e|^n) \xrightarrow{\text{ctx}} |e|^v \]

   Subtlety: two translations have different types

   \[ |e|^n \xleftarrow{\phi} \phi(|e|^n) \]
Outline

How do we prove evaluation order equivalences (assuming \textit{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 \textit{call-by-need}
- Example: name and need are equivalent if only effect is nontermination

How do we do \textit{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)
- Values correspond to call-by-value, computations correspond to call-by-name
Call-by-push-value [Levy ’99]

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

$$M_1 \text{ to } x. M_2$$

is a computation

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

$$\text{thunk } M$$

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:

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

\[ \text{UC} \]

Value terms:

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

\[ \text{thunk} M \quad \text{thunks} \]

\[ x \]

Computation types:

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

\[ A \rightarrow C \]

\[ \text{FA} \]

Computation terms:

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

\[ \lambda x. M \mid V \leftarrow M \quad \text{functions} \]

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

\[ \text{force} V \]
Call-by-push-value syntax

Value types:

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

Computation types:

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

Value terms:

\[ V, W ::= c \mid \ldots \] constants, products, etc.
\[ \mid thunk M \] thunks
\[ \mid x \]

Computation terms:

\[ M, N ::= \ldots \] products, etc.
\[ \mid \lambda x. M \mid V \to M \] functions
\[ \mid return V \mid M_1 \text{ to } x. M_2 \] returners
\[ \mid force V \]
Call-by-push-value syntax

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

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

Computation types:
\[ C, D ::= \ldots \]
\[ | A \to C \]
\[ | FA \]

Computation terms:
\[ M, N ::= \ldots \]
\[ | \lambda x. M | V \cdot M \]
\[ | \text{return} V | M_1 \text{ to } x. M_2 \]
\[ | \text{force} V \]
Call-by-push-value syntax

Value types:

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

\[ | \text{UC} \]

Value terms:

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

\[ | \text{thunk } M \]

\[ | x \]

Computation types:

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

\[ | A \rightarrow C \]

\[ | FA \]

Computation terms:

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

\[ | \lambda x. M | V \!

\[ \text{functions} \]

\[ | \text{return } V | M_1 \overset{x}{\rightarrow} M_2 \]

\[ | \text{returners} \]

\[ | \text{force } V \]
Call-by-push-value typing

\[ \Gamma ::= \emptyset \mid x : A \]

Thunks:

\[ \Gamma \vdash M : C \quad \Rightarrow \quad \Gamma \vdash \text{thunk } M : UC \]

\[ \Gamma \vdash V : UC \quad \Rightarrow \quad \Gamma \vdash \text{force } V : C \]

Returner types:

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

\[ \Gamma \vdash M_1 : FA \quad \Gamma, x : A \vdash M_2 : C \quad \Rightarrow \quad \Gamma \vdash M_1 \text{ to } x. M_2 : C \]
Call-by-push-value equational theory

We also have an equational theory

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

Use this to define contextual equivalence

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

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:

\[ \tau ::= \text{unit} | \text{bool} | \tau \rightarrow \tau' \]

Translations from value and name into CBPV:

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

\[ \text{unit} \mapsto \text{unit} \]
\[ \text{bool} \mapsto \text{bool} \]
\[ (\tau \rightarrow \tau') \mapsto U(|\tau|^v \rightarrow F(|\tau'|)^v) \]
\[ (\tau \rightarrow \tau') \mapsto (U(|\tau|^n) \rightarrow |\tau'|)^n) \]

\[ \Gamma, x : \tau \mapsto (|\Gamma|^v, x : |\tau|^v) \]
\[ \Gamma, x : \tau \mapsto (|\Gamma|^n, x : U(|\tau|^n) \]

\[ \Gamma \vdash e : \tau \mapsto (|\Gamma|^v \vdash (e)^v : F(|\tau|^v) \]
\[ \Gamma \vdash e : \tau \mapsto (|\Gamma|^n \vdash (e)^n : (|\tau|^n) \]
Call-by-value and call-by-name

\[
\begin{array}{c}
\langle \Gamma \rangle^v \quad \langle e \rangle^v \\
\downarrow \quad \cong_{\text{ctx}} \\
\langle \Gamma \rangle^n \quad \langle e \rangle^n
\end{array}
\quad \xrightarrow{\quad \text{F} \quad}
\begin{array}{c}
\langle \tau \rangle^v \\
\langle \tau \rangle^n
\end{array}
\]
Call-by-value and call-by-name

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

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

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

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

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

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

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

We’ll want

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

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

Assume:

▶ Defined in usual way on type formers excluding $F$

\[
\mathcal{R}[\text{UC}] = \{(\text{thunk } M, \text{thunk } M') | (M, M') \in \mathcal{R}[C]\}
\]
\[
\mathcal{R}[A \rightarrow C] = \{(M, M') | \forall (V, V') \in \mathcal{R}[A]. \ (V'M, V''M') \in \mathcal{R}[C]\}
\]

▶ Closed under return:

\[
(V, V') \in \mathcal{R}[A] \quad \Rightarrow \quad (\text{return } V, \text{return } V') \in \mathcal{R}[FA]
\]

▶ Closed under $\text{to}$: if $x : A \vdash N, N' : \underline{C}$ and

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

then

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

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

▶ Transitivity
Logical relations for CBPV

Lemma (Fundamental)

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

\[
(M[x_1 \mapsto V_1, \ldots, x_n \mapsto V_n], M[x_1 \mapsto V'_1, \ldots, x_n \mapsto V'_n]) \in \mathcal{R}[C]
\]
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 $\mathcal{R}[F(U(FA))]$.

This implies:

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

Lemma

*If everything is thunkable and $M : \langle \tau \rangle^n$ then*

$$\Phi_\tau(\Psi_\tau M) \quad \mathcal{R}[\langle \tau \rangle^n] \quad M$$
The equivalence

Want to show that

\[
\begin{array}{c}
(\Gamma)^v \xrightarrow{(e)^v} F(\tau)^v \\
\downarrow \hspace{1cm} \downarrow \\
(\Gamma)^n \xrightarrow{(e)^n} (\tau)^n
\end{array}
\]

Meaning:

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

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

\[
(e)^v \equiv (e)^n
\]
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

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

For no side-effects:

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

Read-only state

get : F bool

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

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 (ECPBV)

New computation forms:

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

\[ | \ x \ |

\[ | M_1 \text{ need } x. M_2 | \]

computation variables

call-by-need sequencing

Typing:

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

\[ (x : FA) \in \Gamma \]

\[ \frac{}{\Gamma \vdash x : FA} \]

\[ \frac{\Gamma \vdash M_1 : FA \quad \Gamma, x : FA \vdash M_2 : C}{\Gamma \vdash M_1 \text{ need } x. M_2 : C} \]
Extended call-by-push-value

Important equation:

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

Associativity:

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

Given
\[ \Gamma \vdash M_1 : FA \quad \Gamma, x : FA \vdash M_2 : C \]

have various evaluation orders:

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

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

\[ \tau \mapsto \text{value type } (|\tau|)^{\text{need}} \]

\[ \text{unit} \mapsto \text{unit} \]

\[ \text{bool} \mapsto \text{bool} \]

\[ (\tau \rightarrow \tau') \mapsto \text{U}
\left( \text{U}(\text{F}(|\tau|)^{\text{need}}) \rightarrow \text{F}(|\tau'|)^{\text{need}} \right) \]

\[ \Gamma, x : \tau \mapsto (|\Gamma|)^{\text{need}}, \ x : \text{F}(|\tau|)^{\text{need}} \]

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

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

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

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

Two nice properties:

▶ Applying lambdas

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

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

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

[Ariola & Felleisen ’97]
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:

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

3. Corollary:

$$\langle e \rangle^{\text{moggi}} \equiv_{\text{ctx}} \langle e \rangle^{\text{need}}$$
Internal equivalence: proof idea

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

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

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

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

$$M_1 \text{ name } x. M_2 \simeq_{\text{ctx}} M_1 \text{ need } x. M_2$$
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 $\epsilon \subseteq \Sigma$

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

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

- 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

\[\Gamma \vdash M : C \quad C <: D\]
\[\Gamma \vdash M : D\]

Subtyping \(C <: D\)
\(\langle \epsilon \rangle A <: \langle \epsilon' \rangle B\) if \(\epsilon \subseteq \epsilon'\) and \(A <: B\)

\[\Gamma \vdash M_1 : \langle \epsilon \rangle A\]
\[\Gamma, x : A \vdash M_2 : C\]
\[\Gamma \vdash M_1 \text{ to } x. M_2 : \langle \epsilon \rangle C\]

Preordered monoid action:
\(\langle \epsilon \rangle C\)
\(\langle \epsilon \rangle(\langle \epsilon' \rangle A) := \langle \epsilon \cup \epsilon' \rangle A\)
\(\langle \epsilon \rangle(A \rightarrow C) := A \rightarrow \langle \epsilon \rangle C\)
Overview

How to prove an equivalence between evaluation orders:

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

\[
\begin{align*}
&\frac{\langle \Gamma \rangle^v \xrightarrow{(e)^v} \mathbf{F} \langle \tau \rangle^v}{\mathbf{F} \xrightarrow{\approx_{\text{ctx}}} \langle \tau \rangle^n} \\
&\frac{\langle \Gamma \rangle^n \xrightarrow{(e)^n} \langle \tau \rangle^n}{\langle \Gamma \rangle^v \xrightarrow{(e)^v} \mathbf{F} \langle \tau \rangle^v}
\end{align*}
\]

- 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)