

Extended call-by-push-value

Reasoning about effectful programs and evaluation order

Dylan McDermott Alan Mycroft

University of Cambridge

Goal

General framework for proving statements of the form

*If <restriction on side-effects> then <evaluation order 1>
is equivalent to <evaluation order 2>*

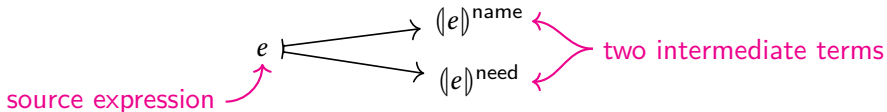
Examples:

- ▶ If 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



2. Prove contextual equivalence

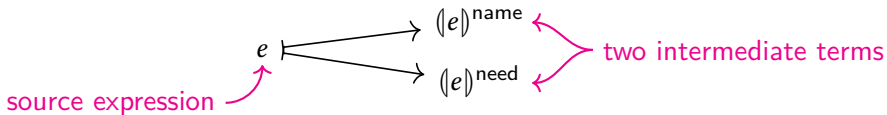
$$(e)^{\text{name}} \cong_{\text{ctx}} (e)^{\text{need}}$$

(where $M \cong_{\text{ctx}} M'$ is defined in terms of an equational theory $N \equiv N'$)

Method

Use an intermediate language that supports various evaluation orders:

1. Translate from source language to intermediate language



2. Prove contextual equivalence

$$\phi((e)^{\text{name}}) \cong_{\text{ctx}} (e)^{\text{need}}$$

Subtlety: two translations have different types

$$(e)^{\text{name}} \longmapsto \phi((e)^{\text{name}})$$

another intermediate term

Contributions

General framework for reasoning about evaluation order:

- ▶ New intermediate language: extension of Levy's call-by-push-value to capture **call-by-need**
- ▶ Method for proving equivalences between evaluation orders (assuming **global** (whole-language) restriction on side-effects)
 - ▶ Example: name and need are equivalent if only effect is nontermination
- ▶ Generalize to **local** (per expression) restrictions

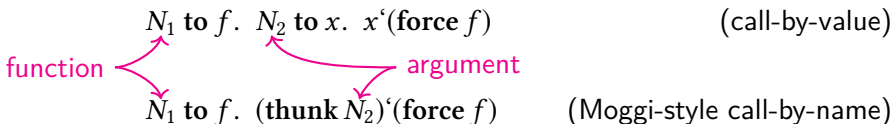
Call-by-push-value language [Levy '99]

- ▶ Split syntax into **values** (no side-effects) and **computations** (with side-effects)
- ▶ Evaluation order of computations is explicit:

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

(think monads: similar to $M_1 \gg \lambda x. M_2$ and `do` $x \leftarrow M_1; M_2$)

Translation of function application (N_1, N_2 are computations):



- ▶ Can't do call-by-need

Call-by-push-value syntax

Value types:

$A, B ::= \dots$

| $\mathbf{U}\underline{C}$

Value terms:

$V, W ::= \dots$

| **thunk** M

| x

products, etc.

thunks

Computation types:

$\underline{C}, \underline{D} ::= \dots$

| $A \rightarrow \underline{C}$

| $\mathbf{F}A$

Computation terms:

$M, N ::= \dots$

| $\lambda x. M$ | $V \text{ ' } M$

| **return** V | M_1 **to** $x. M_2$

| **force** V

products, etc.

functions

returners

Call-by-push-value syntax

Value types:

$A, B ::= \dots$

| $\underline{U}C$

Value terms:

$V, W ::= \dots$

| **thunk** M

| x

products, etc.

thunks

Computation types:

$\underline{C}, \underline{D} ::= \dots$

| $A \rightarrow \underline{C}$

| **F** A

Computation terms:

$M, N ::= \dots$

| $\lambda x. M$ | $V \text{ ' } M$

| **return** V | M_1 **to** $x. M_2$ returners

| **force** V

products, etc.

functions

returners

Call-by-push-value syntax

Value types:

$A, B ::= \dots$

| \mathbf{UC}

Value terms:

$V, W ::= \dots$

| **think** M

| x

products, etc.

thinks

Computation types:

$\underline{C}, \underline{D} ::= \dots$

| $A \rightarrow \underline{C}$

| **F** A

Computation terms:

$M, N ::= \dots$

| $\lambda x. M$ | $V \text{ ' } M$

| **return** V | M_1 **to** $x. M_2$

| **force** V

products, etc.

functions

returners

Call-by-push-value syntax

Value types:

$A, B ::= \dots$

| \mathbf{UC}

Value terms:

$V, W ::= \dots$

| **thunk** M

| x

products, etc.

thunks

Computation types:

$\underline{C}, \underline{D} ::= \dots$

| $A \rightarrow \underline{C}$

| **FA**

Computation terms:

$M, N ::= \dots$

| $\lambda x. M$ | $V \text{ ' } M$

| **return** V | M_1 **to** $x. M_2$ **returners**

| **force** V

products, etc.

functions

returners

Call-by-push-value typing

Thunks:

$$\frac{\Gamma \vdash M : \underline{C}}{\Gamma \vdash_{\mathbf{V}} \mathbf{thunk} M : \underline{UC}}$$

$$\frac{\Gamma \vdash_{\mathbf{V}} V : \underline{UC}}{\Gamma \vdash \mathbf{force} V : \underline{C}}$$

Returner types:

$$\frac{\Gamma \vdash_{\mathbf{V}} V : A}{\Gamma \vdash \mathbf{return} V : \mathbf{FA}}$$

$$\frac{\Gamma \vdash M_1 : \mathbf{FA} \quad \Gamma, x : A \vdash M_2 : \underline{C}}{\Gamma \vdash M_1 \mathbf{to} x. M_2 : \underline{C}}$$

Variables:

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash_{\mathbf{V}} x : A}$$

Extended call-by-push-value (ECBPV)

New computation forms:

$$M, N ::= \dots$$

\underline{x}	computation variables
$M_1 \mathbf{need} \underline{x}. M_2$	call-by-need sequencing

Typing rules:

$$\frac{(\underline{x} : \mathbf{F}A) \in \Gamma}{\Gamma \vdash \underline{x} : \mathbf{F}A} \qquad \frac{\Gamma \vdash M_1 : \mathbf{F}A \quad \Gamma, \underline{x} : \mathbf{F}A \vdash M_2 : \underline{C}}{\Gamma \vdash M_1 \mathbf{need} \underline{x}. M_2 : \underline{C}}$$

Important equation:

$$M_1 \mathbf{need} \underline{x}. \underline{x} \mathbf{to} y. M_2 \equiv M_1 \mathbf{to} y. M_2[\underline{x} \mapsto \mathbf{return} y]$$

Extended call-by-push-value

Given

$$\Gamma \vdash M_1 : \mathbf{F}A$$

$$\Gamma, \underline{x} : \mathbf{F}A \vdash M_2 : \underline{C}$$

have various evaluation orders:

- ▶ Call-by-value: $M_1 \mathbf{value} \underline{x}. M_2 \equiv M_1 \mathbf{to} y. M_2[\underline{x} \mapsto \mathbf{return} y]$
- ▶ Call-by-name: $M_1 \mathbf{name} \underline{x}. M_2 \equiv M_2[\underline{x} \mapsto M_1]$
- ▶ Call-by-need: $M_1 \mathbf{need} \underline{x}. M_2$ (builtin)

Call-by-need translation

$$e \mapsto \llbracket e \rrbracket^{\text{need}}$$

$$e e' \quad \llbracket e \rrbracket^{\text{need}} \text{ to } f. (\text{think}(\llbracket e' \rrbracket^{\text{need}})) \text{ ' } (\text{force } f)$$

$$\lambda x. e \quad \text{return}(\text{think}(\lambda x'. \\ \text{(force } x') \text{ need } \underline{x}. \llbracket e \rrbracket^{\text{need}})))$$

Two nice properties:

- ▶ Applying lambdas

$$\llbracket (\lambda x. e) e' \rrbracket^{\text{need}} \equiv \llbracket e' \rrbracket^{\text{need}} \text{ need } \underline{x}. \llbracket e \rrbracket^{\text{need}}$$

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

$$e \xrightarrow{\text{need}} e' \quad \Rightarrow \quad \llbracket e \rrbracket^{\text{need}} \equiv \llbracket e' \rrbracket^{\text{need}}$$

[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** $\Omega_{\underline{C}}$
2. Prove internal equivalence:

$$M_1 \mathbf{name} \underline{x}. M_2 \cong_{\text{ctx}} M_1 \mathbf{need} \underline{x}. M_2$$

3. Corollary:

$$\langle e \rangle^{\text{name}} [x_1 \mapsto \mathbf{think} \underline{x}_1, \dots, x_n \mapsto \mathbf{think} \underline{x}_n] \cong_{\text{ctx}} \langle e \rangle^{\text{need}}$$

Internal equivalence: proof idea

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

Proof: use logical relations

- ▶ Reasoning about **to**:

diverging computation Ω_{FA} $\text{to } x. M_2 \equiv \Omega_{\underline{C}}$ $\text{return } V \text{ to } x. M_2 \equiv M_2[x \mapsto V]$ pure computation

- ▶ Don't have similar equations for **need**:

$$\Omega_{\text{FA}} \text{ need } \underline{x}. M_2 \neq \Omega_{\underline{C}}$$

- ▶ Relate **open** terms: Kripke logical relations of varying arity

$$R_A^\Gamma \subseteq \text{Term}_A^\Gamma \times \text{Term}_A^\Gamma$$

Global restriction on side-effects

If whole language restricted to nontermination, then

$$M_1 \mathbf{name} \underline{x}. M_2 \cong_{ctx} M_1 \mathbf{need} \underline{x}. M_2$$

Local restriction on side-effects

If ~~whole language~~ M_1 restricted to nontermination, then

$$M_1 \mathbf{name} \underline{x}. M_2 \cong_{ctx} M_1 \mathbf{need} \underline{x}. M_2$$

Effect system for (E)CBPV

Goal: place **upper** bound on side-effects of computations

- ▶ Replace returner types $\mathbf{F} A$ with $\langle \varepsilon \rangle A$
- ▶ Track **effects** $\varepsilon \subseteq \Sigma$

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

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

- ▶ Internal equivalence (with effect system):

If $M_1 : \langle \varepsilon \rangle A$ for $\varepsilon \subseteq \{\text{diverge}\}$, then

$$M_1 \mathbf{name} \underline{x}. M_2 \quad \cong_{ctx} \quad M_1 \mathbf{need} \underline{x}. M_2$$

Effect system for (E)CBPV

$$\frac{\Gamma \vdash M : \underline{C} \quad \underline{C} <: \underline{D}}{\Gamma \vdash M : \underline{D}}$$

Subtyping $\underline{C} <: \underline{D}$

$\langle \varepsilon \rangle A <: \langle \varepsilon' \rangle B$ if $\varepsilon \subseteq \varepsilon'$ and $A <_{\varepsilon} B$

$$\frac{\Gamma \vdash M_1 : \langle \varepsilon \rangle A \quad \Gamma, x : A \vdash M_2 : \underline{C}}{\Gamma \vdash M_1 \text{ to } x. M_2 : \langle \varepsilon \rangle \underline{C}}$$

Preordered monoid action: $\langle \varepsilon \rangle \underline{C}$

$\langle \varepsilon \rangle (\langle \varepsilon' \rangle A) := \langle \varepsilon \cup \varepsilon' \rangle A$

$\langle \varepsilon \rangle (A \rightarrow \underline{C}) := A \rightarrow \langle \varepsilon \rangle \underline{C}$

Overview

- ▶ Extended call-by-push-value
 - ▶ Captures call-by-value, call-by-name, **call-by-need**, ...
- ▶ Method for proving equivalences between evaluation orders (assuming **global** restriction on side-effects)
 - ▶ Example: name and need are equivalent if only effect is nontermination
 - ▶ Prove internal equivalence using logical relations

$$M_1 \mathbf{name} \underline{x}. M_2 \cong_{\text{ctx}} M_1 \mathbf{need} \underline{x}. M_2$$

- ▶ Prove source-language equivalence by translating into ECBPV

$$\phi(|e|^{\mathbf{name}}) \cong_{\text{ctx}} |e|^{\mathbf{need}}$$

- ▶ Generalize to **local** restrictions on side-effects, using effect system

Benefits

Language-level reasoning about evaluation order, instead of ad hoc techniques