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

$$\begin{tikzcd} & (\!|e|\!)^{\mathsf{name}} \\ e \arrow[ur, mapsto] \arrow[dr, mapsto] & \\ & (\!|e|\!)^{\mathsf{need}} \end{tikzcd}$$

two intermediate terms

source expression

2. Prove contextual equivalence

$$(\!|e|\!)^{\mathsf{name}} \;\cong_{\mathsf{ctx}}\; (\!|e|\!)^{\mathsf{need}}$$

(where $M \cong_{\mathsf{ctx}} M'$ 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

$$\begin{tikzcd} & \llparenthesis e \rrparenthesis^{\mathsf{name}} \\ e \arrow[ur, mapsto] \arrow[dr, mapsto] & \\ & \llparenthesis e \rrparenthesis^{\mathsf{need}} \end{tikzcd}$$

two intermediate terms

source expression

2. Prove contextual equivalence

$$\phi(\llparenthesis e \rrparenthesis^{\mathsf{name}}) \;\cong_{\mathsf{ctx}}\; \llparenthesis e \rrparenthesis^{\mathsf{need}}$$

Subtlety: two translations have different types

$$\llparenthesis e \rrparenthesis^{\mathsf{name}} \longmapsto \phi(\llparenthesis e \rrparenthesis^{\mathsf{name}})$$

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:

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

(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):

$N_1\ \mathbf{to}\ f.\ \ N_2\ \mathbf{to}\ x.\ \ x\text{‘}(\mathbf{force}\ f)$ (call-by-value)

function argument

$N_1\ \mathbf{to}\ f.\ \ (\mathbf{thunk}\ N_2)\text{‘}(\mathbf{force}\ f)$ (Moggi-style call-by-name)

- Can't do call-by-need

Call-by-push-value syntax

Value types:

$$A, B ::= \ldots \mid \mathbf{U}\,\underline{C}$$

Value terms:

$$V, W ::= \ldots \quad \text{products, etc.}$$
$$\mid \mathbf{thunk}\,M \quad \text{thunks}$$
$$\mid x$$

Computation types:

$$\underline{C}, \underline{D} ::= \ldots \mid A \to \underline{C} \mid \mathbf{F}\,A$$

Computation terms:

$$M, N ::= \ldots \quad \text{products, etc.}$$
$$\mid \lambda x.\, M \mid V\,\text{`}\,M \quad \text{functions}$$
$$\mid \mathbf{return}\,V \mid M_1\ \mathbf{to}\ x.\, M_2 \quad \text{returners}$$
$$\mid \mathbf{force}\,V$$

Call-by-push-value syntax

Value types:

$A, B ::= \ldots$

$\quad \mid \mathbf{U}\,\underline{C}$

Value terms:

$V, W ::= \ldots$ products, etc.

$\quad \mid \mathbf{thunk}\,M$ thunks

$\quad \mid x$

Computation types:

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

$\quad \mid A \to \underline{C}$

$\quad \mid \mathbf{F}\,A$

Computation terms:

$M, N ::= \ldots$ products, etc.

$\quad \mid \lambda x.\, M \mid V \text{ ` } M$ functions

$\quad \mid \mathbf{return}\,V \mid M_1 \ \mathbf{to}\ x.\, M_2$ returners

$\quad \mid \mathbf{force}\,V$

Call-by-push-value syntax

Value types:

$$A, B ::= \ldots \mid \mathbf{U}\,\underline{C}$$

Value terms:

$V, W ::= \ldots$	products, etc.
$\mid \ \mathbf{thunk}\, M$	thunks
$\mid x$	

Computation types:

$$\underline{C}, \underline{D} ::= \ldots \mid A \to \underline{C} \mid \mathbf{F}\, A$$

Computation terms:

$M, N ::= \ldots$	products, etc.
$\mid \lambda x.\, M \mid V \text{ ` } M$	functions
$\mid \ \mathbf{return}\, V \mid M_1 \ \mathbf{to}\ x.\, M_2$	returners
$\mid \ \mathbf{force}\, V$	

Call-by-push-value syntax

Value types:

$$A, B ::= \ldots$$
$$\mid \mathbf{U}\,\underline{C}$$

Value terms:

$V, W ::= \ldots$	products, etc.
$\mid \ \mathbf{thunk}\, M$	thunks
$\mid x$	

Computation types:

$$\underline{C}, \underline{D} ::= \ldots$$
$$\mid A \to \underline{C}$$
$$\mid \ \mathbf{F}\, A$$

Computation terms:

$M, N ::= \ldots$	products, etc.
$\mid \lambda x.\, M \mid V \text{ ` } M$	functions
$\mid \ \mathbf{return}\, V \mid M_1\ \mathbf{to}\ x.\, M_2$	returners
$\mid \ \mathbf{force}\, V$	

Call-by-push-value typing

Thunks:

$$\frac{\Gamma \vdash M : \underline{C}}{\Gamma \vdash_{\mathsf{v}} \mathbf{thunk}\, M : \mathbf{U}\, \underline{C}} \qquad \frac{\Gamma \vdash_{\mathsf{v}} V : \mathbf{U}\, \underline{C}}{\Gamma \vdash \mathbf{force}\, V : \underline{C}}$$

Returner types:

$$\frac{\Gamma \vdash_{\mathsf{v}} V : A}{\Gamma \vdash \mathbf{return}\, V : \mathbf{F}\, A} \qquad \frac{\Gamma \vdash M_1 : \mathbf{F}\, A \qquad \Gamma, x : A \vdash M_2 : \underline{C}}{\Gamma \vdash M_1\ \mathbf{to}\ x.\, M_2 : \underline{C}}$$

Variables:

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash_{\mathsf{v}} x : A}$$

Extended call-by-push-value (ECBPV)

New computation forms:

$$\begin{aligned} M, N &::= \dots \\ &\mid \underline{x} && \text{computation variables} \\ &\mid M_1 \ \mathbf{need}\ \underline{x}.\, M_2 && \text{call-by-need sequencing} \end{aligned}$$

Typing rules:

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

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

$$\Gamma \vdash M_1 : \mathbf{F}\,A \qquad\qquad \Gamma, \underline{x} : \mathbf{F}\,A \vdash M_2 : \underline{C}$$

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

$$e \longmapsto (\!|e|\!)^{\mathsf{need}}$$

$$e\,e' \qquad (\!|e|\!)^{\mathsf{need}}\ \mathbf{to}\ f.\,(\mathbf{thunk}(\!|e'|\!)^{\mathsf{need}})\ \text{`}\ (\mathbf{force}\ f)$$

$$\lambda x.\,e \qquad \begin{array}{l}\mathbf{return}\,(\mathbf{thunk}\,(\lambda x'.\\ \quad (\mathbf{force}\,x')\ \mathbf{need}\ \underline{x}.\ (\!|e|\!)^{\mathsf{need}}))\end{array}$$

Two nice properties:

- Applying lambdas

$$(\!|(\lambda x.\,e)\ e'|\!)^{\mathsf{need}} \equiv (\!|e'|\!)^{\mathsf{need}}\ \mathbf{need}\ \underline{x}.\ (\!|e|\!)^{\mathsf{need}}$$

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

$$e \overset{\mathsf{need}}{\rightsquigarrow} e' \qquad \Longrightarrow \qquad (\!|e|\!)^{\mathsf{need}} \equiv (\!|e'|\!)^{\mathsf{need}}$$

[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:

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

3. Corollary:

$$(\!|e|\!)^{\mathsf{name}}[x_1 \mapsto \mathbf{thunk}\, \underline{x}_1, \ldots, x_n \mapsto \mathbf{thunk}\, \underline{x}_n] \quad \cong_{\mathsf{ctx}} \quad (\!|e|\!)^{\mathsf{need}}$$

Internal equivalence: proof idea

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

Proof: use logical relations

- Reasoning about **to**:

$$\Omega_{\mathbf{F}A}\ \mathbf{to}\ x.\, M_2 \equiv \Omega_{\underline{C}} \qquad \mathbf{return}\, V\ \mathbf{to}\ x.\, M_2 \equiv M_2[x \mapsto V]$$

diverging computation

pure computation

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

$$\Omega_{\mathbf{F}A}\ \mathbf{need}\ \underline{x}.\, M_2 \not\equiv \Omega_{\underline{C}}$$

- Relate open terms: Kripke logical relations of varying arity

$$R^{\Gamma}_{A} \subseteq \mathrm{Term}^{\Gamma}_{A} \times \mathrm{Term}^{\Gamma}_{A}$$

Global restriction on side-effects

If whole language restricted to nontermination, then

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

Local restriction on side-effects

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

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

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 $\varepsilon \subseteq \Sigma$

$$\Sigma \coloneqq \{\mathsf{diverge}, \mathsf{get}, \mathsf{put}, \mathsf{raise}, \dots\}$$

$$\Omega : \langle\{\mathsf{diverge}\}\rangle A \qquad \mathbf{get} : \langle\{\mathsf{get}\}\rangle \mathbf{bool} \qquad \cdots$$

- Internal equivalence (with effect system):

  If $M_1 : \langle\varepsilon\rangle A$ *for* $\varepsilon \subseteq \{\mathsf{diverge}\}$, *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

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

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

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

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

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

$$\langle\varepsilon\rangle(\langle\varepsilon'\rangle A) \coloneqq \langle\varepsilon \cup \varepsilon'\rangle A$$

$$\langle\varepsilon\rangle(A \to \underline{C}) \coloneqq A \to \langle\varepsilon\rangle\underline{C}$$

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

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

  - Prove source-language equivalence by translating into ECBPV

$$\phi(\llparenthesis e \rrparenthesis^{\mathsf{name}}) \ \cong_{\mathsf{ctx}} \ \llparenthesis e \rrparenthesis^{\mathsf{need}}$$

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

Benefits

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