On the relation between call-by-value and call-by-name

Dylan McDermott

Reykjavik University

Joint work with Alan Mycroft (Cambridge)

Goal

Suppose we replace one evaluation order (e.g. call-by-value) with another (e.g. call-by-name)

How does this affect the behaviour of programs?

- No side-effects: nothing changes
- Recursion, exceptions, state, nondeterminism, . . . : ???

Call-by-value (CBV) and call-by-name (CBN)

$$\begin{array}{llllllll} \text{CBV:} & e\,e' & \rightsquigarrow_{\mathrm{v}}^{*} & (\lambda x.\,e'')\,e' & \rightsquigarrow_{\mathrm{v}}^{*} & (\lambda x.\,e'')\,v & \rightsquigarrow_{\mathrm{v}} & e''[x \mapsto v] \\ \text{CBN:} & e\,e' & \rightsquigarrow_{\mathrm{n}}^{*} & (\lambda x.\,e'')\,e' & & & \rightsquigarrow_{\mathrm{n}} & e''[x \mapsto e'] \end{array}$$

Call-by-value (CBV) and call-by-name (CBN)

$$\begin{array}{llllllll} \text{CBV:} & e\,e' & \rightsquigarrow_{\mathrm{v}}^{*} & (\lambda x.\,e'')\,e' & \rightsquigarrow_{\mathrm{v}}^{*} & (\lambda x.\,e'')\,v & \rightsquigarrow_{\mathrm{v}} & e''[x \mapsto v] \\ \text{CBN:} & e\,e' & \rightsquigarrow_{\mathrm{n}}^{*} & (\lambda x.\,e'')\,e' & & & \rightsquigarrow_{\mathrm{n}} & e''[x \mapsto e'] \end{array}$$

So if Ω reduces to itself:

- $(\lambda x.\,42)\,\Omega$ doesn't terminate in CBV:

$$(\lambda x.\,42)\,\Omega \;\rightsquigarrow_{\mathrm{v}}\; (\lambda x.\,42)\,\Omega \;\rightsquigarrow_{\mathrm{v}}\; (\lambda x.\,42)\,\Omega \;\rightsquigarrow_{\mathrm{v}}\; \cdots$$

- But does terminate in CBN:

$$(\lambda x.\,42)\,\Omega \;\rightsquigarrow_{\mathrm{n}}\; 42$$

CBV and CBN don't have the same behaviour in general

Goal

CBV and CBN don't in general have the same behaviour

But for programs[1]:

- No side-effects: CBV and CBN are the same
- Only recursion: if CBV terminates, then CBN terminates with same result
- Only nondeterminism: if CBV can terminate with result v, CBN can terminate with result v

How can we prove these?

[1] Program = closed expression of ground type

Method

Use an intermediate language that captures various evaluation orders:

1. Translate from source language to intermediate language

call-by-name intermediate term

$$\begin{tikzcd} & \llparenthesis e \rrparenthesis^{\mathrm{n}} \\ e \arrow[ur, mapsto] \arrow[dr, mapsto] & \\ & \llparenthesis e \rrparenthesis^{\mathrm{v}} \end{tikzcd}$$

source expression

call-by-value intermediate term

2. Prove relationship between two translations

$$\llparenthesis e \rrparenthesis^{\mathrm{v}} \;\preccurlyeq_{\mathrm{ctx}}\; \llparenthesis e \rrparenthesis^{\mathrm{n}}$$

Method

Use an intermediate language that captures various evaluation orders:

1. Translate from source language to intermediate language

call-by-name intermediate term

$$\begin{tikzcd} & (\!|e|\!)^{\mathrm{n}} \\ e \arrow[ur, mapsto] \arrow[dr, mapsto] & \\ & (\!|e|\!)^{\mathrm{v}} \end{tikzcd}$$

source expression

call-by-value intermediate term

2. Prove relationship between two translations

$$(\!|e|\!)^{\mathrm{v}} \preccurlyeq_{\mathrm{ctx}} \phi((\!|e|\!)^{\mathrm{n}})$$

Subtlety: two translations have different types

$$(\!|e|\!)^{\mathrm{n}} \longmapsto \phi((\!|e|\!)^{\mathrm{n}})$$

another intermediate term

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

Evaluation order is explicit

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

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

  is a computation

- Can thunk computations: if M is a computation then

$$\mathbf{thunk}\, M$$

  is a value

$\Rightarrow$ can do call-by-value and call-by-name

Call-by-push-value syntax

Value types:

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

Value terms:

$$\begin{aligned} V, W ::=\ & c \mid \ldots && \text{constants, products, etc.} \\ \mid\ & \mathbf{thunk}\, M && \text{thunks} \\ \mid\ & x \end{aligned}$$

Computation types:

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

Computation terms:

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

Call-by-push-value syntax

Value types:

$$A, B ::= \ldots$$

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

Value terms:

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

$$\mid \textbf{thunk}\,M \qquad \text{thunks}$$

$$\mid x$$

Computation types:

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

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

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

Computation terms:

$$M, N ::= \ldots \qquad \text{products, etc.}$$

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

$$\mid \textbf{return}\,V \mid M_1 \textbf{ to } x.\,M_2 \qquad \text{returners}$$

$$\mid \textbf{force}\,V$$

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

Call-by-push-value syntax

Value types:

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

Value terms:

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

Computation types:

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

Computation terms:

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

$$\boxed{\text{Typing contexts:} \quad \Gamma ::= \diamond \mid x : A}$$

Call-by-push-value syntax

Value types:

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

Value terms:

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

Computation types:

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

Computation terms:

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

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

Some side-effects

Recursion:

$$\frac{\Gamma, x : \mathbf{U}\,\underline{C} \vdash M : \underline{C}}{\Gamma \vdash \mathbf{rec}\, x\!:\!\mathbf{U}\,\underline{C}.\, M : \underline{C}}$$

Nondeterminism:

$$\frac{\Gamma \vdash M_1 : \underline{C} \qquad \Gamma \vdash M_2 : \underline{C}}{\Gamma \vdash M_1\ \mathbf{or}\ M_2 : \underline{C}}$$

Call-by-value and call-by-name

Source language types:

$$\tau ::= \mathbf{unit} \mid \mathbf{bool} \mid \tau \to \tau'$$

Translations from value and name into CBPV:

$$\begin{array}{rcl} \tau & \mapsto & \text{value type } (\!|\tau|\!)^{\mathrm{v}} \\ \mathbf{unit} & \mapsto & \mathbf{unit} \\ \mathbf{bool} & \mapsto & \mathbf{bool} \\ (\tau \to \tau') & \mapsto & \mathbf{U}((\!|\tau|\!)^{\mathrm{v}} \to \mathbf{F}\,(\!|\tau'|\!)^{\mathrm{v}}) \end{array} \qquad \begin{array}{rcl} \tau & \mapsto & \text{computation type } (\!|\tau|\!)^{\mathrm{n}} \\ \mathbf{unit} & \mapsto & \mathbf{F}\ \mathbf{unit} \\ \mathbf{bool} & \mapsto & \mathbf{F}\ \mathbf{bool} \\ (\tau \to \tau') & \mapsto & ((\mathbf{U}\,(\!|\tau|\!)^{\mathrm{n}}) \to (\!|\tau'|\!)^{\mathrm{n}}) \end{array}$$

$$\Gamma, x : \tau \;\mapsto\; (\!|\Gamma|\!)^{\mathrm{v}}, x : (\!|\tau|\!)^{\mathrm{v}} \qquad \Gamma, x : \tau \;\mapsto\; (\!|\Gamma|\!)^{\mathrm{n}}, x : \mathbf{U}\,(\!|\tau|\!)^{\mathrm{n}}$$

$$\Gamma \vdash e : \tau \;\mapsto\; (\!|\Gamma|\!)^{\mathrm{v}} \underline{\vdash} (\!|e|\!)^{\mathrm{v}} : \mathbf{F}\,(\!|\tau|\!)^{\mathrm{v}} \qquad \Gamma \vdash e : \tau \;\mapsto\; (\!|\Gamma|\!)^{\mathrm{n}} \underline{\vdash} (\!|e|\!)^{\mathrm{n}} : (\!|\tau|\!)^{\mathrm{n}}$$

Call-by-value and call-by-name

Want to relate

$$(\!|\Gamma|\!)^{\mathrm{v}} \xrightarrow{(\!|e|\!)^{\mathrm{v}}} \mathbf{F}\,(\!|\tau|\!)^{\mathrm{v}}$$

to

$$(\!|\Gamma|\!)^{\mathrm{n}} \xrightarrow{(\!|e|\!)^{\mathrm{n}}} (\!|\tau|\!)^{\mathrm{n}}$$

ON THE RELATION BETWEEN DIRECT AND CONTINUATION SEMANTICS†

John C. Reynolds

Systems and Information Science

Syracuse University

ABSTRACT: The use of continuations in the definition of programming languages has gained considerable currency recently, particularly in conjunction with the lattice-theoretic methods of D. Scott. Although continuations are apparently needed to provide a mathematical semantics for non-applicative control features, they are unnecessary for the definition of a purely applicative language, even when call-by-value occurs. This raises the question of the relationship between the direct and the continuation semantic functions for a purely applicative language. We give two theorems which specify this relationship and show that, in a precise sense, direct semantics are included in continuation semantics.

The heart of the problem is the construction of a relation which must be a fixed-point of a non-monotonic "relational functor." A general method is given for the construction of such relations between recursively defined domains.

Call-by-value and call-by-name

Want to relate

$$(\!|\Gamma|\!)^{\mathrm{v}} \xrightarrow{(\!|e|\!)^{\mathrm{v}}} \mathbf{F}\,(\!|\tau|\!)^{\mathrm{v}}$$

to

$$(\!|\Gamma|\!)^{\mathrm{v}} \longrightarrow (\!|\Gamma|\!)^{\mathrm{n}} \xrightarrow{(\!|e|\!)^{\mathrm{n}}} (\!|\tau|\!)^{\mathrm{n}} \longrightarrow \mathbf{F}\,(\!|\tau|\!)^{\mathrm{v}}$$

Call-by-value and call-by-name

Define maps between call-by-value and call-by-name computations?

$$\Gamma \underline{\vdash} M : \mathbf{F} \llparenthesis \tau \rrparenthesis^{\mathrm{v}} \quad \mapsto \quad \Gamma \underline{\vdash} \Phi_\tau M : \llparenthesis \tau \rrparenthesis^{\mathrm{n}}$$

$$\Gamma \underline{\vdash} N : \llparenthesis \tau \rrparenthesis^{\mathrm{n}} \quad \mapsto \quad \Gamma \underline{\vdash} \Psi_\tau N : \mathbf{F} \llparenthesis \tau \rrparenthesis^{\mathrm{v}}$$

Call-by-value and call-by-name

Define maps between call-by-value and call-by-name computations?

$$\Gamma \vdash M : \mathbf{F}\,(\!|\tau|\!)^{\mathrm{v}} \quad \mapsto \quad \Gamma \vdash \Phi_\tau M : \;(\!|\tau|\!)^{\mathrm{n}}$$

$$\Gamma \vdash N : \;(\!|\tau|\!)^{\mathrm{n}} \quad \mapsto \quad \Gamma \vdash \Psi_\tau N : \mathbf{F}\,(\!|\tau|\!)^{\mathrm{v}}$$

Example: for $\tau = \mathbf{unit} \to \mathbf{unit}$

$$M \quad \overset{\text{CBV to CBN}}{\mapsto} \quad M\ \mathbf{to}\ f.\,\lambda x.\,\mathbf{force}\,x\ \mathbf{to}\ z.\,z\,\text{‘}\,\mathbf{force}\,f$$

$$N \quad \overset{\text{CBN to CBV}}{\mapsto} \quad \mathbf{return}\,\mathbf{thunk}\,\lambda x.\,(\mathbf{thunk}\,\mathbf{return}\,x)\,\text{‘}\,N$$

Call-by-value and call-by-name

Want to relate

$$(\!|\Gamma|\!)^{\mathrm{v}} \xrightarrow{(\!|e|\!)^{\mathrm{v}}} \mathbf{F}\,(\!|\tau|\!)^{\mathrm{v}}$$

to

$$\color{red}{(\!|\Gamma|\!)^{\mathrm{v}} \longrightarrow} \color{black}{(\!|\Gamma|\!)^{\mathrm{n}} \xrightarrow{(\!|e|\!)^{\mathrm{n}}} (\!|\tau|\!)^{\mathrm{n}}} \color{red}{\longrightarrow \mathbf{F}\,(\!|\tau|\!)^{\mathrm{v}}}$$

Denotational semantics

Assume some denotational semantics for CBPV:

- If $\Gamma \vdash M : \underline{C}$ then $[\![M]\!] : [\![\Gamma]\!] \to [\![\underline{C}]\!]$
- Require order-enrichment: $[\![M]\!] \sqsubseteq [\![N]\!]$

Examples:

	$[\![\Gamma]\!]$	$[\![\underline{C}]\!]$	$[\![M]\!]$	$[\![M]\!] \sqsubseteq [\![N]\!]$
No side-effects	set	set	function	equality
Recursion	cpo	cpo with $\bot$	continuous func.	pointwise
Nondeterminism	poset	join-semilattice	monotone func.	pointwise

(All of these are adequate: if $[\![M]\!] \sqsubseteq [\![N]\!]$ then $M \preccurlyeq_{\mathrm{ctx}} N$)

Maps between CBV and CBN, semantically

Interpret Φ_τ and Ψ_τ

$$\Gamma \vdash M : \mathbf{F}\,(\!|\tau|\!)^{\mathrm{v}} \quad \mapsto \quad \Gamma \vdash \Phi_\tau M : (\!|\tau|\!)^{\mathrm{n}}$$

$$\Gamma \vdash N : (\!|\tau|\!)^{\mathrm{n}} \quad \mapsto \quad \Gamma \vdash \Psi_\tau N : \mathbf{F}\,(\!|\tau|\!)^{\mathrm{v}}$$

in the denotational semantics

$$\phi_\tau : [\![\mathbf{F}\,(\!|\tau|\!)^{\mathrm{v}}]\!] \longrightarrow [\![(\!|\tau|\!)^{\mathrm{n}}]\!]$$
$$\psi_\tau : [\![(\!|\tau|\!)^{\mathrm{n}}]\!] \longrightarrow [\![\mathbf{F}\,(\!|\tau|\!)^{\mathrm{v}}]\!]$$

Want to show:

$$[\![(\!|e|\!)^{\mathrm{v}}]\!] \sqsubseteq \psi_\tau \circ [\![(\!|e|\!)^{\mathrm{n}}]\!] \circ \phi_\Gamma$$

Galois connection between CBV and CBN?

If $(\phi_\tau \circ -, \psi_\tau \circ -)$ is a Galois connection, i.e.

$$id \sqsubseteq \psi_\tau \circ \phi_\tau \qquad \phi_\tau \circ \psi_\tau \sqsubseteq id$$

for each τ, then

$$[\![(\!| e |\!)^{\mathrm{v}}]\!] \sqsubseteq \psi_\tau \circ [\![(\!| e |\!)^{\mathrm{n}}]\!] \circ \phi_\Gamma$$

Galois connection between CBV and CBN?

These do not hold in all cases!

$$id \sqsubseteq \psi_\tau \circ \phi_\tau \qquad \phi_\tau \circ \psi_\tau \sqsubseteq id$$

- Don't hold for: exceptions, printing, state (even if read-only)
- Do hold for: no side-effects, recursion, nondeterminism

This is where the side-effects matter

Galois connection between CBV and CBN?

Definition (Thunkable [Führmann '99])

A computation $\Gamma \vDash M : \mathbf{F}\,A$ is (lax) *thunkable* if

$$[\![M \ \mathbf{to}\ x.\,\mathbf{return}\,(\mathbf{thunk}\,(\mathbf{return}\,x))]\!] \ \sqsubseteq \ [\![\mathbf{return}\,(\mathbf{thunk}\,M)]\!]$$

- Essentially: we're allowed to suspend the computation M
- Implies M commutes with other computations, is (lax) discardable, (lax) copyable

Lemma

If everything is thunkable, then $(\phi_\tau \circ -, \psi_\tau \circ -)$ is a Galois connection.

(In adjunction models, assumption is $UF\eta_Y \sqsubseteq \eta_{UFY}$)

How to relate call-by-value to call-by-name

If every computation is thunkable then

$$[\![(\!| e |\!)^{\mathrm{v}}]\!] \;\sqsubseteq\; \psi_\tau \circ [\![(\!| e |\!)^{\mathrm{n}}]\!] \circ \phi_\Gamma$$

for each e.

And if e is a program then

$$[\![(\!| e |\!)^{\mathrm{v}}]\!] \sqsubseteq [\![(\!| e |\!)^{\mathrm{n}}]\!]$$

Examples

If e is a program:

- No side-effects: $(\!|e|\!)^{\mathrm{v}}$ and $(\!|e|\!)^{\mathrm{n}}$ reduce to the same values
- Nontermination: if $(\!|e|\!)^{\mathrm{v}}$ reduces to v, then so does $(\!|e|\!)^{\mathrm{n}}$
- Nondeterminism: if $(\!|e|\!)^{\mathrm{v}}$ can reduce to v, then $(\!|e|\!)^{\mathrm{n}}$ can also reduce to v

But this doesn't prove anything about exceptions, state, ...

What else can we show?

Local restriction on side-effects: what if the language has other side-effects, but e does not?

- Effect systems

What about other evaluation orders?

- Use the same technique?

Effect systems

Goal: place upper bound on side-effects of computations

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

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

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

- New theorem (for recursion): if e is a closed program with effect $\varepsilon \subseteq \{\mathsf{diverge}\}$ then

$$(\!|e|\!)^{\mathrm{v}} \rightsquigarrow^{*} v \quad \Rightarrow \quad (\!|e|\!)^{\mathrm{n}} \rightsquigarrow^{*} v$$

Call-by-need and call-by-name

Call-by-need improves on call-by-name by sharing computations:

$$\textbf{let}\ x = 2 + 2\ \textbf{in}\ x + x$$

Call-by-need and call-by-name

Call-by-need improves on call-by-name by sharing computations:

$$\textbf{let } x = \boxed{2+2} \textbf{ in } \boxed{x} + x$$

Call-by-need and call-by-name

Call-by-need improves on call-by-name by sharing computations:

$$\begin{aligned} & \textbf{let}\ x = 2 + 2\ \textbf{in}\ x + x \\ \rightsquigarrow_{\text{need}}\ & \textbf{let}\ x = 4\ \textbf{in}\ x + x \end{aligned}$$

Call-by-need and call-by-name

Call-by-need improves on call-by-name by sharing computations:

$$
\begin{aligned}
& \mathbf{let}\ x = 2 + 2\ \mathbf{in}\ x + x \\
\rightsquigarrow_{\text{need}}\ & \mathbf{let}\ x = 4\ \mathbf{in}\ x + x \\
\rightsquigarrow_{\text{need}}\ & \mathbf{let}\ x = 4\ \mathbf{in}\ 4 + x
\end{aligned}
$$

Call-by-need and call-by-name

Call-by-need improves on call-by-name by sharing computations:

$$\begin{aligned} & \textbf{let } x = 2 + 2 \textbf{ in } x + x \\ \rightsquigarrow_{\text{need}} \ & \textbf{let } x = 4 \textbf{ in } x + x \\ \rightsquigarrow_{\text{need}} \ & \textbf{let } x = 4 \textbf{ in } 4 + x \end{aligned}$$

Call-by-need and call-by-name

Call-by-need improves on call-by-name by sharing computations:

$$
\begin{aligned}
& \textbf{let } x = 2 + 2 \textbf{ in } x + x \\
\rightsquigarrow_{\text{need}} \; & \textbf{let } x = 4 \textbf{ in } x + x \\
\rightsquigarrow_{\text{need}} \; & \textbf{let } x = 4 \textbf{ in } 4 + x \\
\rightsquigarrow_{\text{need}} \; & \textbf{let } x = 4 \textbf{ in } 4 + 4
\end{aligned}
$$

Call-by-need and call-by-name

Call-by-need improves on call-by-name by sharing computations:

$$
\begin{aligned}
& \mathbf{let}\ x = 2 + 2\ \mathbf{in}\ x + x \\
\rightsquigarrow_{\text{need}}\ & \mathbf{let}\ x = 4\ \mathbf{in}\ x + x \\
\rightsquigarrow_{\text{need}}\ & \mathbf{let}\ x = 4\ \mathbf{in}\ 4 + x \\
\rightsquigarrow_{\text{need}}\ & \mathbf{let}\ x = 4\ \mathbf{in}\ 4 + 4
\end{aligned}
$$

Call-by-need and call-by-name

Call-by-need improves on call-by-name by sharing computations:

$$
\begin{aligned}
& \mathbf{let}\ x = 2 + 2\ \mathbf{in}\ x + x \\
\rightsquigarrow_{\text{need}}\ & \mathbf{let}\ x = 4\ \mathbf{in}\ x + x \\
\rightsquigarrow_{\text{need}}\ & \mathbf{let}\ x = 4\ \mathbf{in}\ 4 + x \\
\rightsquigarrow_{\text{need}}\ & \mathbf{let}\ x = 4\ \mathbf{in}\ 4 + 4 \\
\rightsquigarrow_{\text{need}}\ & \mathbf{let}\ x = 4\ \mathbf{in}\ 8
\end{aligned}
$$

Call-by-need and call-by-name

Call-by-need improves on call-by-name by sharing computations:

$$
\begin{aligned}
& \textbf{let } x = 2 + 2 \textbf{ in } x + x \\
\rightsquigarrow_{\text{need}}\ & \textbf{let } x = 4 \textbf{ in } x + x \\
\rightsquigarrow_{\text{need}}\ & \textbf{let } x = 4 \textbf{ in } 4 + x \\
\rightsquigarrow_{\text{need}}\ & \textbf{let } x = 4 \textbf{ in } 4 + 4 \\
\rightsquigarrow_{\text{need}}\ & \textbf{let } x = 4 \textbf{ in } 8
\end{aligned}
$$

In some cases (e.g. only recursion), call-by-need and call-by-name should be the same

Call-by-need and call-by-name

Call-by-need is hard: "action at a distance"

- Extend CBPV with new construct

$$M \ \mathbf{need}\ \underline{x}.\, N$$

- Define a call-by-need translation
- Don't know how to do denotational semantics for call-by-need
  - Kripke logical relations of varying arity [Jung and Tiuryn '93]

$$\mathcal{R}[\![\underline{C}]\!]\, \Gamma \subseteq \mathrm{Term}^{\Gamma}_{\underline{C}} \times \mathrm{Term}^{\Gamma}_{\underline{C}}$$

Overview

How to relate evaluation orders:

1. Translate from source language to intermediate language
2. Define maps between evaluation orders
3. Relate terms:

$$[\![(\!| e |\!)^{\mathrm{v}}]\!] \sqsubseteq \psi_\tau \circ [\![(\!| e |\!)^{\mathrm{n}}]\!] \circ \phi_\Gamma$$

- Works for call-by-value, call-by-name
  - Call-by-need by extending CBPV
- Also works for local restrictions on side-effects using an effect system