Formalising Modal Embeddings of Call-by-Name and Call-by-Value

Floris Reuvers

Supervisor: dr. N.M. van der Weide
Second Reader: dr. E.G.M. Hubbers

June 18, 2025

Radboud University

Introduction

Context

Functional Programming Languages

- OCaml, F#, SML, Haskell, Clean

Evaluation Strategies

- **Call-by-value (cbv)**
  - Evaluate the argument first, then substitute
  - OCaml, F#, SML
- **Call-by-name (cbn)** / lazy
  - Substitute the argument, evaluate only when needed
  - Haskell, Clean

Example cbn and cbv

Let f be defined as

$f(x) = x * x$

Example cbn and cbv

Let f be defined as

$f(x) = x * x$

cbn and cbv evaluation of f (3 + 3)

Example cbn and cbv

Let f be defined as

$f(x) = x * x$

cbn and cbv evaluation of f (3 + 3)

cbn: **cvb:**

Example cbn and cbv

Let f be defined as

$f(x) = x * x$

cbn and cbv evaluation of f (3 + 3)

cbn:	**cvb:**
$f(3+3) \to (3+3)*(3+3)$	$f(3+3) \to f(6)$

Example cbn and cbv

Let f be defined as

$f(x) = x * x$

cbn and cbv evaluation of f (3 + 3)

cbn:	**cvb:**
$f(3+3) \rightarrow (3+3)*(3+3)$	$f(3+3) \rightarrow f(6)$
$\rightarrow 6*(3+3)$	$\rightarrow 6*6$

Example cbn and cbv

Let f be defined as

$f(x) = x * x$

cbn and cbv evaluation of f (3 + 3)

cbn:

$$
\begin{aligned}
f(3+3) &\to (3+3)*(3+3) \\
&\to 6*(3+3) \\
&\to 6*6
\end{aligned}
$$

cvb:

$$
\begin{aligned}
f(3+3) &\to f(6) \\
&\to 6*6 \\
&\to 36
\end{aligned}
$$

Example cbn and cbv

Let f be defined as

$f(x) = x * x$

cbn and cbv evaluation of f (3 + 3)

cbn:

$$
\begin{aligned}
f(3+3) &\to (3+3) * (3+3) \\
&\to 6 * (3+3) \\
&\to 6 * 6 \\
&\to 36
\end{aligned}
$$

cvb:

$$
\begin{aligned}
f(3+3) &\to f(6) \\
&\to 6 * 6 \\
&\to 36
\end{aligned}
$$

Reasons to Unify cbn and cbv

Pros/Conns of cbn and cbv

Sometimes arguments are used:

- **zero** times: **cbn** is more efficient
- **multiple** times: **cbv** is more efficient

Reasons to Unify cbn and cbv

Pros/Conns of cbn and cbv

Sometimes arguments are used:

- **zero** times: **cbn** is more efficient
- **multiple** times: **cbv** is more efficient

Reasons to Unify cbn and cbv

Pros/Conns of cbn and cbv

Sometimes arguments are used:

- **zero** times: **cbn** is more efficient
- **multiple** times: **cbv** is more efficient

Unification

- Best of both worlds
- Universal Framework

Approaches to Unification

Thunks and CPS Translation

- Wrappers around expressions to delay evaluation
- Continuous Passing Style translation

Approaches to Unification

Thunks and CPS Translation

Wrappers around expressions to delay evaluation

Continuous Passing Style translation

Linear Logic

Explicit control over resources

Arguments correspond to resources

Approaches to Unification

Thunks and CPS Translation

Wrappers around expressions to delay evaluation

Continuous Passing Style translation

Linear Logic

Explicit control over resources

Arguments correspond to resources

Modal Logic

Adds box modality

Boxed terms are treated as values

Main Contribution

A formalisation of the unification of call-by-name and call-by-value using modal logic

Background

Grammar Lambda Calculus (λ-calculus)

$\boldsymbol{\lambda}$-terms and Types

$$M, N, P, Q \quad ::= \quad x \mid \lambda x.M \mid MN \qquad A \quad ::= \quad X \mid A \to A'$$

Grammar Lambda Calculus (λ-calculus)

$\boldsymbol{\lambda}$-terms and Types

$$M, N, P, Q \quad ::= \quad x \mid \lambda x.M \mid MN \qquad\qquad A \quad ::= \quad X \mid A \to A'$$

Example terms

Grammar Lambda Calculus (λ-calculus)

$\boldsymbol{\lambda}$-terms and Types

$$M, N, P, Q \quad ::= \quad x \mid \lambda x.M \mid MN \qquad A \quad ::= \quad X \mid A \to A'$$

Example terms

x

Grammar Lambda Calculus (λ-calculus)

$\boldsymbol{\lambda}$-terms and Types

$$M, N, P, Q \quad ::= \quad x \mid \lambda x.M \mid MN \qquad A \quad ::= \quad X \mid A \to A'$$

Example terms

x

$\lambda x.x$

Grammar Lambda Calculus (λ-calculus)

$\boldsymbol{\lambda}$-terms and Types

$$M, N, P, Q \quad ::= \quad x \mid \lambda x.M \mid MN \qquad A \quad ::= \quad X \mid A \to A'$$

Example terms

x

$\lambda x.x$

$\lambda y.yz$

Grammar Lambda Calculus (λ-calculus)

$\boldsymbol{\lambda}$-terms and Types

$$M, N, P, Q \quad ::= \quad x \mid \lambda x.M \mid MN \qquad A \quad ::= \quad X \mid A \to A'$$

Example terms

x

$\lambda x.x$

$\lambda y.yz$

$(\lambda y.y)z$

Grammar Lambda Calculus (λ-calculus)

$\boldsymbol{\lambda}$-terms and Types

$$M, N, P, Q \quad ::= \quad x \mid \lambda x.M \mid MN \qquad A \quad ::= \quad X \mid A \to A'$$

Example terms

x

$\lambda x.x$

$\lambda y.yz$

$(\lambda y.y)z$

$\lambda x.\lambda t.xt(\lambda s.w)wy$

X

Grammar Lambda Calculus (λ-calculus)

$\boldsymbol{\lambda}$-terms and Types

$$M, N, P, Q \quad ::= \quad x \mid \lambda x.M \mid MN \qquad A \quad ::= \quad X \mid A \to A'$$

Example terms

$$x$$
$$\lambda x.x$$
$$\lambda y.yz$$
$$(\lambda y.y)z$$
$$\lambda x.\lambda t.xt(\lambda s.w)wy$$

$$X$$
$$X \to X$$

Grammar Lambda Calculus (λ-calculus)

$\boldsymbol{\lambda}$-terms and Types

$$M, N, P, Q \quad ::= \quad x \mid \lambda x.M \mid MN \qquad\qquad A \quad ::= \quad X \mid A \to A'$$

Example terms

$$x$$
$$\lambda x.x$$
$$\lambda y.yz$$
$$(\lambda y.y)z$$
$$\lambda x.\lambda t.xt(\lambda s.w)wy$$

$$X$$
$$X \to X$$
$$(X \to X) \to X$$

Grammar Lambda Calculus (λ-calculus)

$\boldsymbol{\lambda}$-terms and Types

$$M, N, P, Q \quad ::= \quad x \mid \lambda x.M \mid MN \qquad A \quad ::= \quad X \mid A \to A'$$

Example terms

$$x$$
$$\lambda x.x$$
$$\lambda y.yz$$
$$(\lambda y.y)z$$
$$\lambda x.\lambda t.xt(\lambda s.w)wy$$

$$X$$
$$X \to X$$
$$(X \to X) \to X$$
$$(X \to X) \to (X \to X)$$

Call-by-name evaluation λ-calculus

$\boldsymbol{\beta}$-reduction

$$(\lambda x.M)N \to M[x \mapsto N] \qquad (\beta_n)$$

Closure rule

$$\frac{M \to M'}{MN \to M'N}\,(\mu)$$

cbn evaluation

β-reduction together with the μ closure rule

Call-by-box λ-calculus

$\boldsymbol{\lambda}$-terms

$$M, N, P, Q \quad ::= \quad \varepsilon(x) \mid \lambda x.M \mid MN \mid \mathrm{box}(N)$$

Idea

- ε: unbox operator
- box: box operator
- paramters of abstractions are boxed

Types

$$A \quad ::= \quad X \mid B \to A \mid B \qquad\qquad B \quad ::= \quad \Box A$$

Call-by-box evaluation

β-reduction

$$(\lambda x.M)\mathrm{box}(N) \to M[\varepsilon(x) \mapsto N] \qquad (\beta_b)$$

Closure rules

$$\frac{M \to M'}{MN \to M'N}\,(\mu) \qquad\qquad \frac{N \to N'}{MN \to MN'}\,(\nu)$$

cbb evaluation

β-reduction together with the μ and ν closure rule

Embeddings into λ_b

Girard's and Gödel's Translation

- Translate standard λ-calculus into λ_b
- **Girard's** translation: λ_n into λ_b
- **Gödel's** translation: λ_v into λ_b
- Call-by-box evaluation simulates
  - **cbn** under **Girard's** embedding
  - **cbv** under **Gödel's** embedding
- Notation: M° is the Girard translation of λ-term M
- Main propositions

Girard's Embedding

Translation from cbn λ-terms to λ_b

$$X^\circ = X \qquad\qquad x^\circ = \varepsilon(x)$$
$$(A_1 \to A_2)^\circ = \Box A_1{}^\circ \to A_2{}^\circ \qquad\qquad (\lambda x.M)^\circ = \lambda x.M^\circ$$
$$(MN)^\circ = M^\circ \mathrm{box}(N^\circ)$$

Embedding of $(\lambda x.x)y$

$$\begin{aligned}((\lambda x.x)y)^\circ &= (\lambda x.x)^\circ \mathrm{box}(y^\circ)\\ &= (\lambda x.x^\circ)\mathrm{box}(\varepsilon(y))\\ &= (\lambda x.\varepsilon(x))\mathrm{box}(\varepsilon(y))\end{aligned}$$

Girard's Image

Well-typed terms after translation

$$M, N \quad ::= \quad \varepsilon(x) \mid \lambda x.M \mid M\mathrm{box}(N)$$

- Eliminates the ν closure rule
- Only the μ rule remains
- Call-by-box becomes **deterministic**
- If M evaluates to N with cbn, then M° evaluates to N° with cbb

Agda

What is Agda?

Functional programming language

What is Agda?

Functional programming language

Inspired by Haskell

What is Agda?

Functional programming language

Inspired by Haskell

Proof assistant

What is Agda?

Functional programming language

Inspired by Haskell

Proof assistant

Dependently typed

What is Agda?

Functional programming language

Inspired by Haskell

Proof assistant

Dependently typed

Hole-based programming

Some Agda code

Girard's Embedding for Types

$$(A_1 \to A_2)^\circ = \Box A_1{}^\circ \to A_2{}^\circ$$
$$X^\circ = X$$

Embedding in Agda

$$\begin{aligned}&\mathsf{embedType} : \mathsf{Type} \to \mathsf{Typeb}\\ &\mathsf{embedType}\ (C \Rightarrow D) = \Box\ (\mathsf{embedType}\ C) \Rightarrow\!\mathsf{b}\ \mathsf{embedType}\ D\\ &\mathsf{embedType}\ \mathsf{X} = \mathsf{X}\end{aligned}$$

Challenges

- Variables
  - Use of de Bruijn indices
  - Variables represented as natural numbers
- Ill-typed terms
  - λ-term $y(\lambda x.x)$ is not typeable
  - Solution: restrict to well-typed λ-terms
- Formal definition of substitution
- Formal definition of raise in Gödel's translation

Experience Working with Agda

- Intersting learning experience
- High learning curve
- Hole based programming is amazing
- Error message are not always helpful
- But give it a try!
  - Follow the online PLFA book (Programming Language Foundations in Agda)
  - `https://plfa.github.io/`

Conslusions

Conclusion

- Recap of the λ-calculus
- Explained λ_b
- Definition of the unifying relation cbb
- Girard's translation
- Formally proved the main properties of the translations