

# 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

---

## Functional Programming Languages

- OCaml, F#, SML, Haskell, Clean

## Functional Programming Languages

- OCaml, F#, SML, Haskell, Clean

## Evaluation Strategies

- Call-by-value (cbv)
  - Evaluate the argument first, then substitute
  - OCaml, F#, SML

## 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  $f(3 + 3)$

**cbn:**

**cbv:**

## Example cbn and cbv

Let  $f$  be defined as

$$f(x) = x * x$$

**cbn** and **cbv** evaluation of  $f(3 + 3)$

**cbn:**

$$f(3 + 3) \rightarrow (3 + 3) * (3 + 3)$$

**cbv:**

$$f(3 + 3) \rightarrow f(6)$$



## Example cbn and cbv

Let  $f$  be defined as

$$f(x) = x * x$$

**cbn** and **cbv** evaluation of  $f(3 + 3)$

**cbn:**

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

**cbv:**

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

## Example cbn and cbv

Let  $f$  be defined as

$$f(x) = x * x$$

**cbn** and **cbv** evaluation of  $f(3 + 3)$

**cbn:**

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

**cbv:**

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

## Example cbn and cbv

Let  $f$  be defined as

$$f(x) = x * x$$

**cbn** and **cbv** evaluation of  $f(3 + 3)$

**cbn:**

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

**cbv:**

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

# Reasons to Unify cbn and cbv

## Pros and Cons 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 and Cons 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

## Thunks and CPS Translation

## Thunks and CPS Translation

- Wrappers around expressions to delay evaluation

## 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

## Thunks and CPS Translation

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

## Linear Logic

- Explicit control over resources

## Thunks and CPS Translation

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

## Linear Logic

- Explicit control over resources
- Arguments correspond to resource usage

# 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 resource usage

## Modal Logic

# 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 resource usage

## Modal Logic

- Adds box modality

# 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 resource usage

## Modal Logic

- Adds box modality
- Boxed terms are treated as values

- **Agda**: programming language and proof assistant

# Formalisation in Agda

- Agda: programming language and proof assistant
- Unification of `cbn` and `cbv` into a universal framework



# Formalisation in Agda

- Agda: programming language and proof assistant
- Unification of `cbn` and `cbv` into a universal framework
- Formalise this framework in Agda

# Formalisation in Agda

- Agda: programming language and proof assistant
- Unification of `cbn` and `cbv` into a universal framework
- Formalise this framework in Agda
- Proof of the main properties of the framework

We formalised the unification of call-by-name and call-by-value using modal logic in Agda

# Background

---

# Lambda Calculus ( $\lambda$ -calculus)

Simple and formal **model of computation**

$\lambda$ -terms and types

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

# Lambda Calculus ( $\lambda$ -calculus)

Simple and formal **model of computation**

$\lambda$ -terms and types

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

Example **terms** and **types**

# Lambda Calculus ( $\lambda$ -calculus)

Simple and formal **model of computation**

$\lambda$ -terms and types

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

Example **terms** and **types**

$x$

# Lambda Calculus ( $\lambda$ -calculus)

Simple and formal **model of computation**

$\lambda$ -terms and types

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

Example **terms** and **types**

$x$

$\lambda x.x$



# Lambda Calculus ( $\lambda$ -calculus)

Simple and formal **model of computation**

$\lambda$ -terms and types

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

Example **terms** and **types**

$x$

$\lambda x.x$

$\lambda y.yz$

# Lambda Calculus ( $\lambda$ -calculus)

Simple and formal **model of computation**

$\lambda$ -terms and types

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

Example **terms** and **types**

$x$

$\lambda x.x$

$\lambda y.yz$

$(\lambda y.y)z$

# Lambda Calculus ( $\lambda$ -calculus)

Simple and formal **model of computation**

$\lambda$ -terms and types

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

Example **terms** and **types**

$x$

$\lambda x.x$

$\lambda y.yz$

$(\lambda y.y)z$

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

# Lambda Calculus ( $\lambda$ -calculus)

Simple and formal **model of computation**

$\lambda$ -terms and types

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

Example **terms** and **types**

$x$

$\lambda x.x$

$\lambda y.yz$

$(\lambda y.y)z$

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

$X$

# Lambda Calculus ( $\lambda$ -calculus)

Simple and formal **model of computation**

$\lambda$ -terms and types

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

Example **terms** and **types**

$x$

$\lambda x.x$

$\lambda y.yz$

$(\lambda y.y)z$

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

$X$

$X \rightarrow X$

# Lambda Calculus ( $\lambda$ -calculus)

Simple and formal **model of computation**

$\lambda$ -terms and types

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

Example **terms** and **types**

$x$

$\lambda x.x$

$\lambda y.yz$

$(\lambda y.y)z$

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

$X$

$X \rightarrow X$

$(X \rightarrow X) \rightarrow X$

# Lambda Calculus ( $\lambda$ -calculus)

Simple and formal **model of computation**

$\lambda$ -terms and types

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

Example **terms** and **types**

$x$

$\lambda x.x$

$\lambda y.yz$

$(\lambda y.y)z$

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

$X$

$X \rightarrow X$

$(X \rightarrow X) \rightarrow X$

$(X \rightarrow X) \rightarrow (X \rightarrow X)$

# Call-by-name evaluation $\lambda$ -calculus

$\beta$ -reduction

$$(\lambda x.M)N \rightarrow M[x \mapsto N] \quad (\beta_n)$$



# Call-by-name evaluation $\lambda$ -calculus

$\beta$ -reduction

$$(\lambda x.M)N \rightarrow M[x \mapsto N] \quad (\beta_n)$$

Closure rule

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

# Call-by-name evaluation $\lambda$ -calculus

$\beta$ -reduction

$$(\lambda x.M)N \rightarrow M[x \mapsto N] \quad (\beta_n)$$

Closure rule

$$\frac{M \rightarrow M'}{MN \rightarrow M'N} \quad (\mu)$$

cbn evaluation

$\beta$ -reduction together with the  $\mu$  closure rule

## $\lambda$ -terms

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

# Call-by-box $\lambda$ -calculus

## $\lambda$ -terms

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

## Idea

- $\varepsilon$ : **unbox** operator
- $\text{box}$ : **box** operator
- parameters of abstractions are **boxed**

# Call-by-box $\lambda$ -calculus

## $\lambda$ -terms

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

## Idea

- $\varepsilon$ : **unbox** operator
- $\text{box}$ : **box** operator
- parameters of abstractions are **boxed**

## Types

$$A ::= X \mid B \rightarrow A \mid B \qquad B ::= \Box A$$

$\beta$ -reduction

$$(\lambda x.M)\text{box}(N) \rightarrow M[\varepsilon(x) \mapsto N] \quad (\beta_b)$$

# Call-by-box evaluation

## $\beta$ -reduction

$$(\lambda x.M)\text{box}(N) \rightarrow M[\varepsilon(x) \mapsto N] \quad (\beta_b)$$

## Closure rules

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

$$\frac{N \rightarrow N'}{MN \rightarrow MN'} (\nu)$$

# Call-by-box evaluation

## $\beta$ -reduction

$$(\lambda x.M)\text{box}(N) \rightarrow M[\varepsilon(x) \mapsto N] \quad (\beta_b)$$

## Closure rules

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

$$\frac{N \rightarrow N'}{MN \rightarrow MN'} (\nu)$$

## Call-by-box (cbb) evaluation

$\beta$ -reduction together with the  $\mu$  and  $\nu$  closure rule



## Embeddings into $\lambda_b$

---

# Girard's and Gödel's Translation

- Translate standard  $\lambda$ -calculus into  $\lambda_b$
- Girard's translation:  $\lambda_n$  into  $\lambda_b$
- Gödel's translation:  $\lambda_v$  into  $\lambda_b$
- Cbb simulates
  - cbn under Girard's embedding
  - cbv under Gödel's embedding
- Notation:  $M^\circ$  is the Girard translation of  $\lambda$ -term  $M$

# Girard's Embedding

Translation from cbn  $\lambda$ -terms to  $\lambda_b$

$$X^\circ = X$$

$$x^\circ = \varepsilon(x)$$

$$(A_1 \rightarrow A_2)^\circ = \Box A_1^\circ \rightarrow A_2^\circ$$

$$(\lambda x.M)^\circ = \lambda x.M^\circ$$

$$(MN)^\circ = M^\circ \text{box}(N^\circ)$$

# Girard's Embedding

Translation from cbn  $\lambda$ -terms to  $\lambda_b$

$$X^\circ = X$$

$$x^\circ = \varepsilon(x)$$

$$(A_1 \rightarrow A_2)^\circ = \Box A_1^\circ \rightarrow A_2^\circ$$

$$(\lambda x.M)^\circ = \lambda x.M^\circ$$

$$(MN)^\circ = M^\circ \text{box}(N^\circ)$$

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

$$((\lambda x.x)y)^\circ = (\lambda x.x)^\circ \text{box}(y^\circ)$$

# Girard's Embedding

Translation from cbn  $\lambda$ -terms to  $\lambda_b$

$$X^\circ = X$$

$$x^\circ = \varepsilon(x)$$

$$(A_1 \rightarrow A_2)^\circ = \Box A_1^\circ \rightarrow A_2^\circ$$

$$(\lambda x.M)^\circ = \lambda x.M^\circ$$

$$(MN)^\circ = M^\circ \text{box}(N^\circ)$$

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

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

# Girard's Embedding

Translation from cbn  $\lambda$ -terms to  $\lambda_b$

$$X^\circ = X$$

$$x^\circ = \varepsilon(x)$$

$$(A_1 \rightarrow A_2)^\circ = \Box A_1^\circ \rightarrow A_2^\circ$$

$$(\lambda x.M)^\circ = \lambda x.M^\circ$$

$$(MN)^\circ = M^\circ \text{box}(N^\circ)$$

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

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

# Girard's Embedding

## Translation from cbn $\lambda$ -terms to $\lambda_b$

$$X^\circ = X$$

$$x^\circ = \varepsilon(x)$$

$$(A_1 \rightarrow A_2)^\circ = \Box A_1^\circ \rightarrow A_2^\circ$$

$$(\lambda x.M)^\circ = \lambda x.M^\circ$$

$$(MN)^\circ = M^\circ \text{box}(N^\circ)$$

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

$$((\lambda x.x)y)^\circ = (\lambda x.x)^\circ \text{box}(y^\circ)$$

$$= (\lambda x.x)^\circ \text{box}(\varepsilon(y))$$

$$= (\lambda x.x^\circ) \text{box}(\varepsilon(y))$$

$$= (\lambda x.\varepsilon(x)) \text{box}(\varepsilon(y))$$

## Well-typed terms after translation

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



## Well-typed terms after translation

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

- Eliminates the  $\nu$  closure rule
- Only the  $\mu$  rule remains
- Call-by-box becomes **deterministic**
- If  $M$  evaluates to  $N$  with **cbn**, then  $M^\circ$  evaluates to  $N^\circ$  with **cbb**

# Agda

---

# What is Agda?

- Functional programming language
- Inspired by Haskell
- Proof assistant
- Dependently typed
- Hole-based programming

### Girard's Embedding for Types

$$(A_1 \rightarrow A_2)^\circ = \Box A_1^\circ \rightarrow A_2^\circ$$

$$X^\circ = X$$

## Girard's Embedding for Types

$$(A_1 \rightarrow A_2)^\circ = \Box A_1^\circ \rightarrow A_2^\circ$$
$$X^\circ = X$$

## Embedding in Agda

```
embedType : Type → Typeb
embedType (C ⇒ D) = □ (embedType C) ⇒b embedType D
embedType X = X
```

- Variables
  - Use of de Bruijn indices
  - Variables represented as natural numbers

# Challenges

- Variables
  - Use of **de Bruijn indices**
  - Variables represented as natural numbers
- Ill-typed terms
  - $\lambda$ -term  $y(\lambda x.x)$  is not typeable
  - Solution: restrict to **well-typed**  $\lambda$ -terms

# Challenges

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



## Experience Working with Agda

- Interesting learning experience
- High learning curve
- Hole based programming is amazing
- Error messages are not always helpful
- But give it a try!

# Experience Working with Agda

- Interesting learning experience
- High learning curve
- Hole based programming is amazing
- Error messages 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  $\lambda$ -calculus
- Explained  $\lambda_b$
- Defined the unifying relation  $cbb$
- Girard's translation
- Formally proved the main properties of the translations

Questions?

---