Lightweight Agda Formalization of Denotational Semantics

Peter D Mosses

TU Delft (visitor)
Swansea University (emeritus)

TYPES 2025, Glasgow, Scotland, 9–13 June 2025

About the topic

– Lightweight Agda Formalization of Denotational Semantics

About the topic

– Lightweight Agda Formalization of Denotational Semantics

Lightweight Agda

- requiring ***relatively little effort*** or Agda ***expertise***

About the topic

– Lightweight Agda Formalization of Denotational Semantics

Lightweight Agda

- requiring ***relatively little effort*** or Agda ***expertise***

Formalization

- of (new or existing) ***mathematical*** definitions

About the topic

– Lightweight Agda Formalization of Denotational Semantics

Lightweight Agda

- requiring ***relatively little effort*** or Agda ***expertise***

Formalization

- of (new or existing) ***mathematical*** definitions

Denotational semantics

- with ***recursively-defined Scott-domains***, ***fixed points***, ***λ-notation***

Original motivation

A Denotational Semantics of Inheritance and its Correctness

(1963–2021)

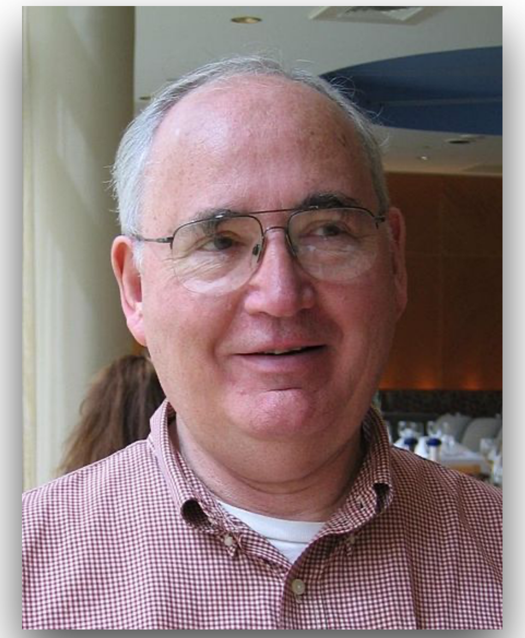
William Cook*
Department of Computer Science
Box 1910 Brown University

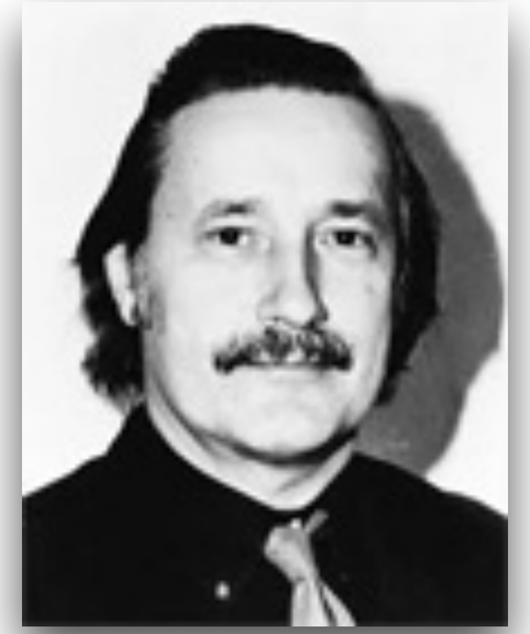
Jens Palsberg
Computer Science Department
Aarhus University

This paper presents a denotational model of inheritance. The model is based on an intuitive motivation of the purpose of inheritance. The correctness of the model is demonstrated by proving it equivalent to an operational semantics of inheritance based upon the method-lookup algorithm of object-oriented languages. ...

OOPSLA '89: Conference proceedings on Object-oriented programming systems, languages and applications

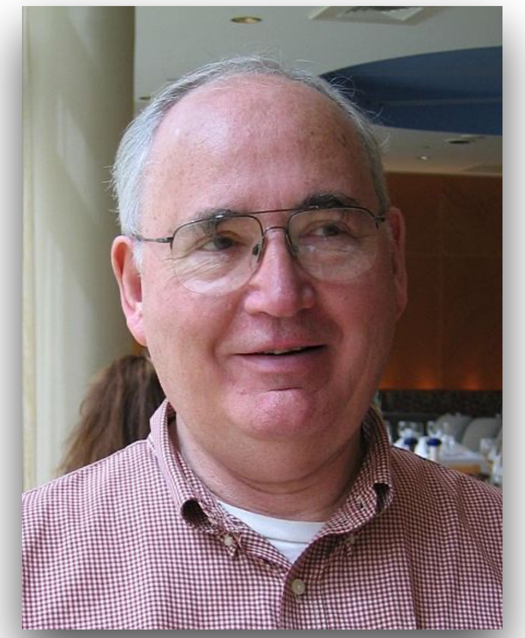
Denotational semantics
– Scott–Strachey style

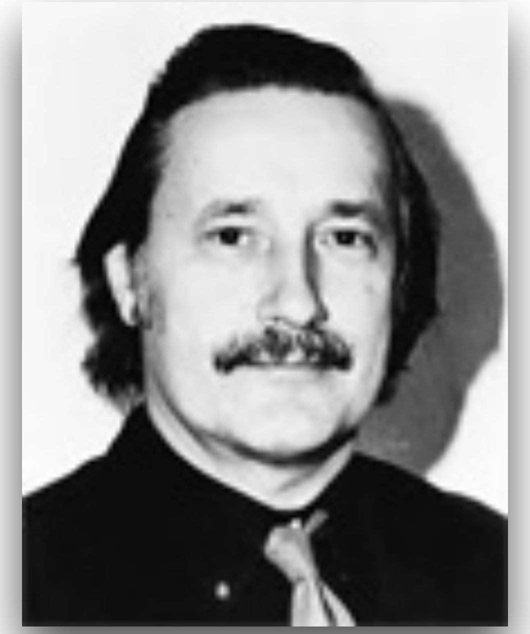




Denotational semantics

– Scott–Strachey style

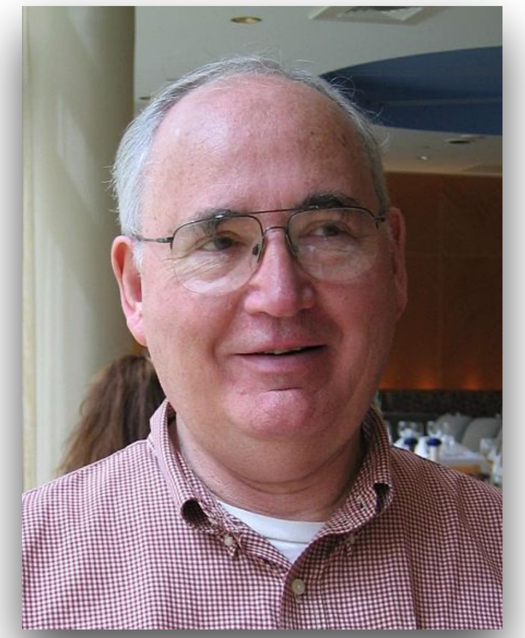


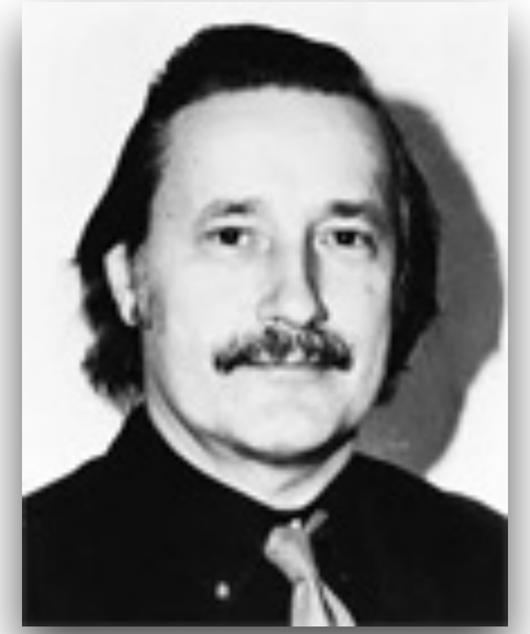


Types of denotations are (Scott-)domains

- ***pointed cpos*** (e.g, ω-complete, directed-complete, continuous lattices)
- ***recursively defined*** – without guards, up to isomorphism

Denotational semantics
– Scott–Strachey style





Types of denotations are (Scott-)domains

- ***pointed cpos*** (e.g, ω-complete, directed-complete, continuous lattices)
- ***recursively defined*** – without guards, up to isomorphism

Denotations are defined in typed λ-notation

- functions on domains are ***continuous maps***
- endofunctions on domains have least ***fixed points***

Models of the untyped λ-calculus
– based on Scott's domain D_∞

Models of the untyped λ-calculus

– based on Scott's domain D_∞

Some mathematical presentations:

- *Dana Scott* (1970,1972): continuous lattices, D_∞
- *Joseph Stoy* (1977): universal domain $\mathscr{P}\omega$
- *Samson Abramsky and Achim Jung* (1994): (pre)domain theory
- *John Reynolds* (2009): *Theories of Programming Languages*, cpos, D_∞

Models of the untyped λ-calculus

– based on Scott's domain D_∞

Some mathematical presentations:

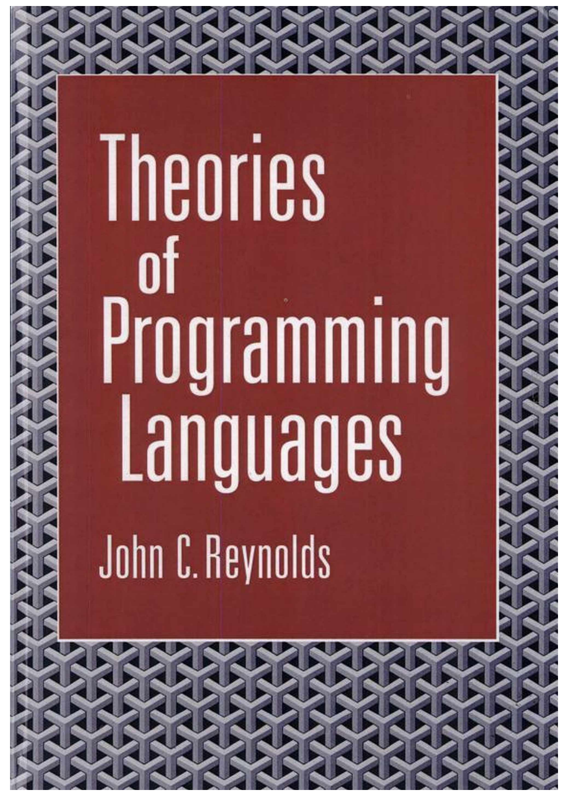
- *Dana Scott* (1970,1972): continuous lattices, D_∞
- *Joseph Stoy* (1977): universal domain $\mathscr{P}\omega$
- *Samson Abramsky and Achim Jung* (1994): (pre)domain theory
- *John Reynolds* (2009): *Theories of Programming Languages*, cpos, D_∞

Some formalizations:

- *Bernhard Reus* (1994): using *Extended Calculus of Constructions*, in *Lego*
- *Tom de Jong* (2021): using *Univalent Type Theory* (TypeTopology), in *Agda*

Reynolds: Theories of Programming Languages

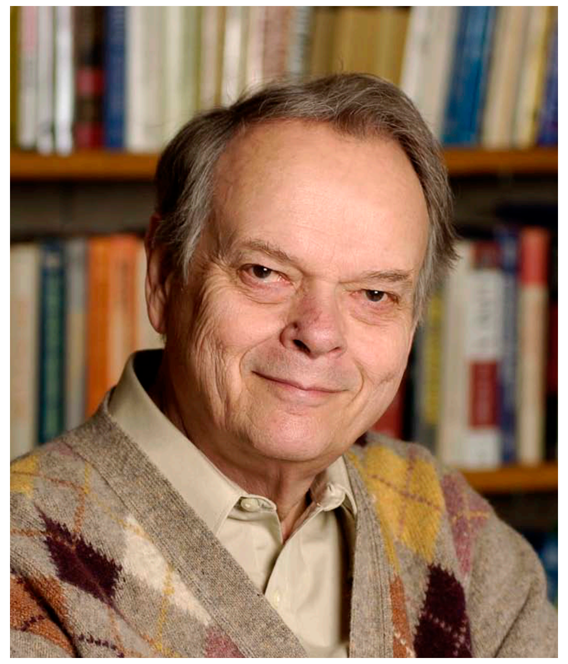
– denotational semantics of the untyped λ-calculus



isomorphism

continuous maps

$$D_\infty \underset{\psi}{\overset{\phi}{\rightleftarrows}} [D_\infty \to D_\infty]$$

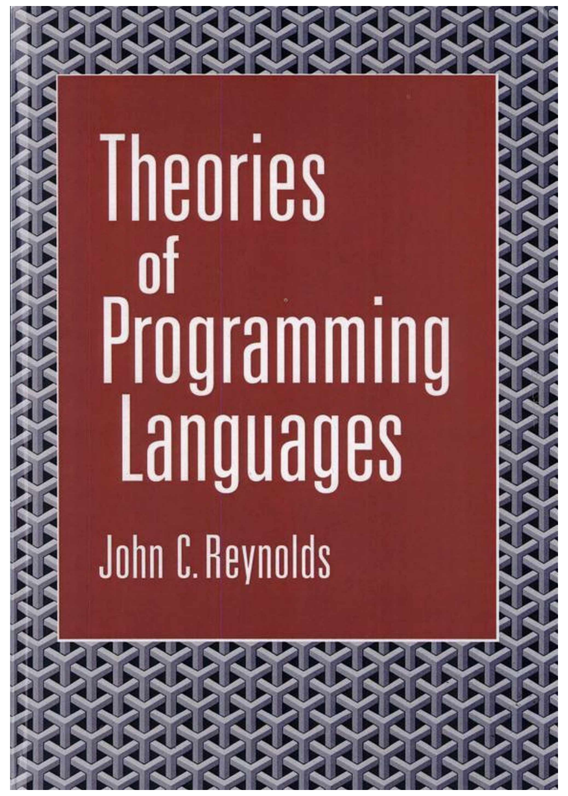


$$\begin{aligned}
[\![-]\!] &\in exp \to [(var \to D_\infty) \to D_\infty] \\
[\![v]\!]\,\eta &= \eta\, v \\
[\![\lambda v.\, e]\!]\,\eta &= \psi\,(\lambda x \in D_\infty.\, [\![e]\!]\,[\eta \mid v : x]) \\
[\![e\, e']\!]\,\eta &= \phi\,([\![e]\!]\,\eta)\,([\![e']\!]\,\eta)
\end{aligned}$$

Copied from www.cs.yale.edu/homes/hudak/CS430F07/LectureSlides/Reynolds-ch10.pdf

Reynolds: Theories of Programming Languages

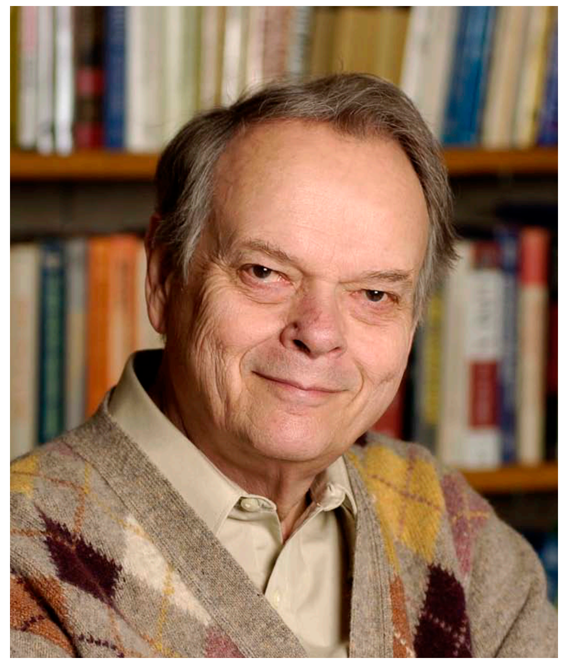
– denotational semantics of the untyped λ-calculus



isomorphism

continuous maps

$$D_\infty \underset{\psi}{\overset{\phi}{\rightleftarrows}} [D_\infty \to D_\infty]$$



$$[\![-]\!] \in exp \to [(var \to D_\infty) \to D_\infty]$$

$$\begin{aligned}
[\![v]\!]\,\eta &= \eta\, v \\
[\![\lambda v.\, e]\!]\,\eta &= \boxed{\psi}(\lambda x \in D_\infty.\, [\![e]\!]\,[\eta \mid v : x]) \\
[\![e\, e']\!]\,\eta &= \boxed{\phi}([\![e]\!]\,\eta)\,([\![e']\!]\,\eta)
\end{aligned}$$

Copied from www.cs.yale.edu/homes/hudak/CS430F07/LectureSlides/Reynolds-ch10.pdf

Agda formalization

– using TypeTopology/DomainTheory (Tom de Jong)

Agda formalization

– using TypeTopology/DomainTheory (Tom de Jong)

We have the non-trivial domain $\mathcal{D}_\infty$ and isomorphism $\mathcal{D}_\infty \simeq^{\mathrm{dcpo}} (\mathcal{D}_\infty \Longrightarrow^{\mathrm{dcpo}} \mathcal{D}_\infty)$.

Agda formalization

– using TypeTopology/DomainTheory (Tom de Jong)

We have the non-trivial domain $\mathcal{D}_\infty$ and isomorphism $\mathcal{D}_\infty \simeq^{\text{dcpo}} (\mathcal{D}_\infty \Longrightarrow^{\text{dcpo}} \mathcal{D}_\infty)$.

$$\mathsf{abs} : \langle\, \mathcal{D}_\infty \Longrightarrow^{\text{dcpo}} \mathcal{D}_\infty \,\rangle \to \langle\, \mathcal{D}_\infty \,\rangle$$

$$\mathsf{app} : \langle\, \mathcal{D}_\infty \,\rangle \to \langle\, \mathcal{D}_\infty \,\rangle \to \langle\, \mathcal{D}_\infty \,\rangle$$

Agda formalization

– using TypeTopology/DomainTheory (Tom de Jong)

We have the non-trivial domain $\mathcal{D}_\infty$ and isomorphism $\mathcal{D}_\infty \simeq^{\mathrm{dcpo}} (\mathcal{D}_\infty \Longrightarrow^{\mathrm{dcpo}} \mathcal{D}_\infty)$.

$$\mathsf{abs} : \langle\ \mathcal{D}_\infty \Longrightarrow^{\mathrm{dcpo}} \mathcal{D}_\infty\ \rangle \to \langle\ \mathcal{D}_\infty\ \rangle$$
$$\mathsf{abs} = [\ \mathcal{D}_\infty \Longrightarrow^{\mathrm{dcpo}} \mathcal{D}_\infty\ ,\ \mathcal{D}_\infty\ ]\langle\ \pi\text{-}\mathsf{exp}_\infty\text{'}\ \rangle$$

$$\mathsf{app} : \langle\ \mathcal{D}_\infty\ \rangle \to \langle\ \mathcal{D}_\infty\ \rangle \to \langle\ \mathcal{D}_\infty\ \rangle$$
$$\mathsf{app} = \mathsf{underlying\text{-}function}\ \mathcal{D}_\infty\ \mathcal{D}_\infty \circ [\ \mathcal{D}_\infty\ ,\ \mathcal{D}_\infty \Longrightarrow^{\mathrm{dcpo}} \mathcal{D}_\infty\ ]\langle\ \varepsilon\text{-}\mathsf{exp}_\infty\text{'}\ \rangle$$

Agda formalization
– using TypeTopology/DomainTheory (Tom de Jong)

We have the non-trivial domain $\mathcal{D}_\infty$ and isomorphism $\mathcal{D}_\infty \sim^{\text{dcpo}} (\mathcal{D}_\infty \Longrightarrow^{\text{dcpo}} \mathcal{D}_\infty)$.

$\mathsf{abs} : \langle\, \mathcal{D}_\infty \Longrightarrow^{\text{dcpo}} \mathcal{D}_\infty \,\rangle \to \langle\, \mathcal{D}_\infty \,\rangle$
$\mathsf{abs} = [\, \mathcal{D}_\infty \Longrightarrow^{\text{dcpo}} \mathcal{D}_\infty \,,\, \mathcal{D}_\infty \,]\langle\, \pi\text{-}\mathsf{exp}_\infty' \,\rangle$

$\mathsf{app} : \langle\, \mathcal{D}_\infty \,\rangle \to \langle\, \mathcal{D}_\infty \,\rangle \to \langle\, \mathcal{D}_\infty \,\rangle$
$\mathsf{app} = \mathsf{underlying\text{-}function}\ \mathcal{D}_\infty\ \mathcal{D}_\infty \circ [\, \mathcal{D}_\infty \,,\, \mathcal{D}_\infty \Longrightarrow^{\text{dcpo}} \mathcal{D}_\infty \,]\langle\, \varepsilon\text{-}\mathsf{exp}_\infty' \,\rangle$

a continuous function is a ***pair:***
- an *underlying* function and
- a *proof* of its continuity

Agda formalization
– using TypeTopology/DomainTheory (Tom de Jong)

$[\![\_]\!] : \mathsf{Exp} \to \mathsf{Env} \to \langle\, \mathcal{D}_\infty \,\rangle$
$ƛ\text{-is-continuous} : \forall\, e\ \rho\ v \to \text{is-continuous}\ \mathcal{D}_\infty\ \mathcal{D}_\infty\ (\lambda\, x \to [\![\, e \,]\!]\ (\rho\ [\, x \,/\, v \,]))$

$[\![\ \mathsf{var}\ v\ ]\!]\ \rho = \rho\ v$
$[\![\ ƛ\ v \cdot e\ ]\!]\ \rho = \mathsf{abs}\ (\ (\lambda\, x \to [\![\, e \,]\!]\ (\rho\ [\, x \,/\, v \,]))\ ,\ ƛ\text{-is-continuous}\ e\ \rho\ v\ )$
$[\![\ e_1 \cdot e_2\ ]\!]\ \rho = \mathsf{app}\ (\ [\![\ e_1\ ]\!]\ \rho\ )\ (\ [\![\ e_2\ ]\!]\ \rho\ )$

Agda formalization

– using TypeTopology/DomainTheory (Tom de Jong)

```
⟦_⟧ : Exp → Env → ⟨ 𝒟∞ ⟩
ƛ-is-continuous : ∀ e ρ v → is-continuous 𝒟∞ 𝒟∞ (λ x → ⟦ e ⟧ (ρ [ x / v ]))

⟦ var v ⟧ ρ = ρ v
⟦ ƛ v · e ⟧ ρ = abs ( (λ x → ⟦ e ⟧ (ρ [ x / v ])) , ƛ-is-continuous e ρ v)
⟦ e₁ · e₂ ⟧ ρ = app ( ⟦ e₁ ⟧ ρ ) ( ⟦ e₂ ⟧ ρ )

ƛ-is-continuous e ρ v = {!  !}
```

Lightweight Agda formalization

– modules

Abstract syntax grammar

- inductive ***datatype definitions***

'Domain' definitions

- ***postulated isomorphisms*** between ***type names*** and ***type terms***

Semantic functions

- functions defined ***inductively*** in ***λ-notation***

Auxiliary definitions

Lightweight Agda formalization

– abstract syntax

```
data Exp : Set where
  var_ : Var → Exp
  lam  : Var → Exp → Exp
  app  : Exp → Exp → Exp
```

Lightweight Agda formalization

– a 'domain'

```
open import Function
  using (Inverse; _↔_) public
open Inverse {{ ... }}
  using (to; from) public

postulate
  D∞ : Set
postulate
  instance iso : D∞ ↔ (D∞ → D∞)
```

Lightweight Agda formalization
– a 'domain'

```
open import Function
  using (Inverse; _↔_) public
open Inverse {{ ... }}
  using (to; from) public

postulate
  D∞ : Set
postulate
  instance iso : D∞ ↔ (D∞ → D∞)
```

Lightweight Agda formalization

– a 'domain'

```
open import Function
  using (Inverse; _↔_) public
open Inverse {{ ... }}
  using (to; from) public

postulate
  D∞ : Set
postulate
  instance iso : D∞ ↔ (D∞ → D∞)
```

Lightweight Agda formalization

– a 'domain'

```
open import Function
  using (Inverse; _↔_) public
open Inverse {{ ... }}
  using (to; from) public

postulate
  D∞ : Set
postulate
  instance iso : D∞ ↔ (D∞ → D∞)
```

bijection

Agda functions

Lightweight Agda formalization

– a 'domain'

```
open import Function
  using (Inverse; _↔_) public
open Inverse {{ ... }}
  using (to; from) public

postulate
  D∞ : Set
postulate
  instance iso : D∞ ↔ (D∞ → D∞)
```

bijection

Agda functions

Lightweight Agda formalization

– semantic function

$$\mathsf{Env} = \mathsf{Var} \to D_\infty$$

$$[\![\_]\!] : \mathsf{Exp} \to \mathsf{Env} \to D_\infty$$

$$\begin{aligned}
[\![\ \mathsf{var}\ v\ ]\!]\ \rho &= \rho\ v \\
[\![\ \mathsf{lam}\ v\ e\ ]\!]\ \rho &= \boxed{\mathsf{from}}\ (\ \lambda\ d \to [\![\ e\ ]\!]\ (\rho\ [\ d\ /\ v\ ])\ ) \\
[\![\ \mathsf{app}\ e_1\ e_2\ ]\!]\ \rho &= \boxed{\mathsf{to}}\ (\ [\![\ e_1\ ]\!]\ \rho\ )\ (\ [\![\ e_2\ ]\!]\ \rho\ )
\end{aligned}$$

Lightweight Agda formalization

– semantic function

$$\text{Env} = \text{Var} \to D_\infty$$

$$[\![\_]\!] : \text{Exp} \to \text{Env} \to D_\infty$$

$$[\![\ \text{var}\ v\ ]\!]\ \rho = \rho\ v$$

$$[\![\ \text{lam}\ v\ e\ ]\!]\ \rho = \text{from}\ (\ \lambda\ d \to [\![\ e\ ]\!]\ (\rho\ [\ d\ /\ v\ ])\ )$$

$$[\![\ \text{app}\ e_1\ e_2\ ]\!]\ \rho = \text{to}\ (\ [\![\ e_1\ ]\!]\ \rho\ )\ (\ [\![\ e_2\ ]\!]\ \rho\ )$$

$$D_\infty \underset{\psi}{\overset{\phi}{\rightleftarrows}} [D_\infty \to D_\infty]$$

$$[\![-]\!] \in exp \to [(var \to D_\infty) \to D_\infty]$$

$$[\![v]\!]\,\eta = \eta\, v$$

$$[\![\lambda v.\, e]\!]\,\eta = \psi\,(\lambda x \in D_\infty.\, [\![e]\!][\eta \mid v : x])$$

$$[\![e\, e']\!]\,\eta = \phi\,([\![e]\!]\,\eta)\,([\![e']\!]\,\eta)$$

Lightweight Agda formalization

– testing denotations

$$\text{check-convergence} : (\lambda x_1 \,.\, x_{42})((\lambda x_0 \,.\, x_0 x_0)(\lambda x_0 \,.\, x_0 x_0)) \equiv x_{42}$$

Lightweight Agda formalization
– testing denotations

check-convergence : $(\lambda x_1 \,.\, x_{42})((\lambda x_0 \,.\, x_0 x_0)(\lambda x_0 \,.\, x_0 x_0)) \equiv x_{42}$

```
check-convergence :
  ⟦ app (lam (x 1) (var x 42))
        (app (lam (x 0) (app (var x 0) (var x 0)))
             (lam (x 0) (app (var x 0) (var x 0)))) ⟧
  ≡ ⟦ var x 42 ⟧
check-convergence = refl
```

Lightweight Agda formalization

– testing denotations

```
to-from-elim : ∀ {f} → to (from f) ≡ f
to-from-elim = inverseˡ iso refl
{-# REWRITE to-from-elim #-}

check-convergence : (λx₁ . x₄₂)((λx₀ . x₀x₀)(λx₀ . x₀x₀)) ≡ x₄₂
  ⟦ app (lam (x 1) (var x 42))
        (app (lam (x 0) (app (var x 0) (var x 0)))
             (lam (x 0) (app (var x 0) (var x 0)))) ⟧
  ≡ ⟦ var x 42 ⟧
check-convergence = refl
```

Lightweight Agda formalization

– testing denotations

```
to-from-elim : ∀ {f} → to (from f) ≡ f
to-from-elim = inverse¹ iso refl
{-# REWRITE to-from-elim #-}
```

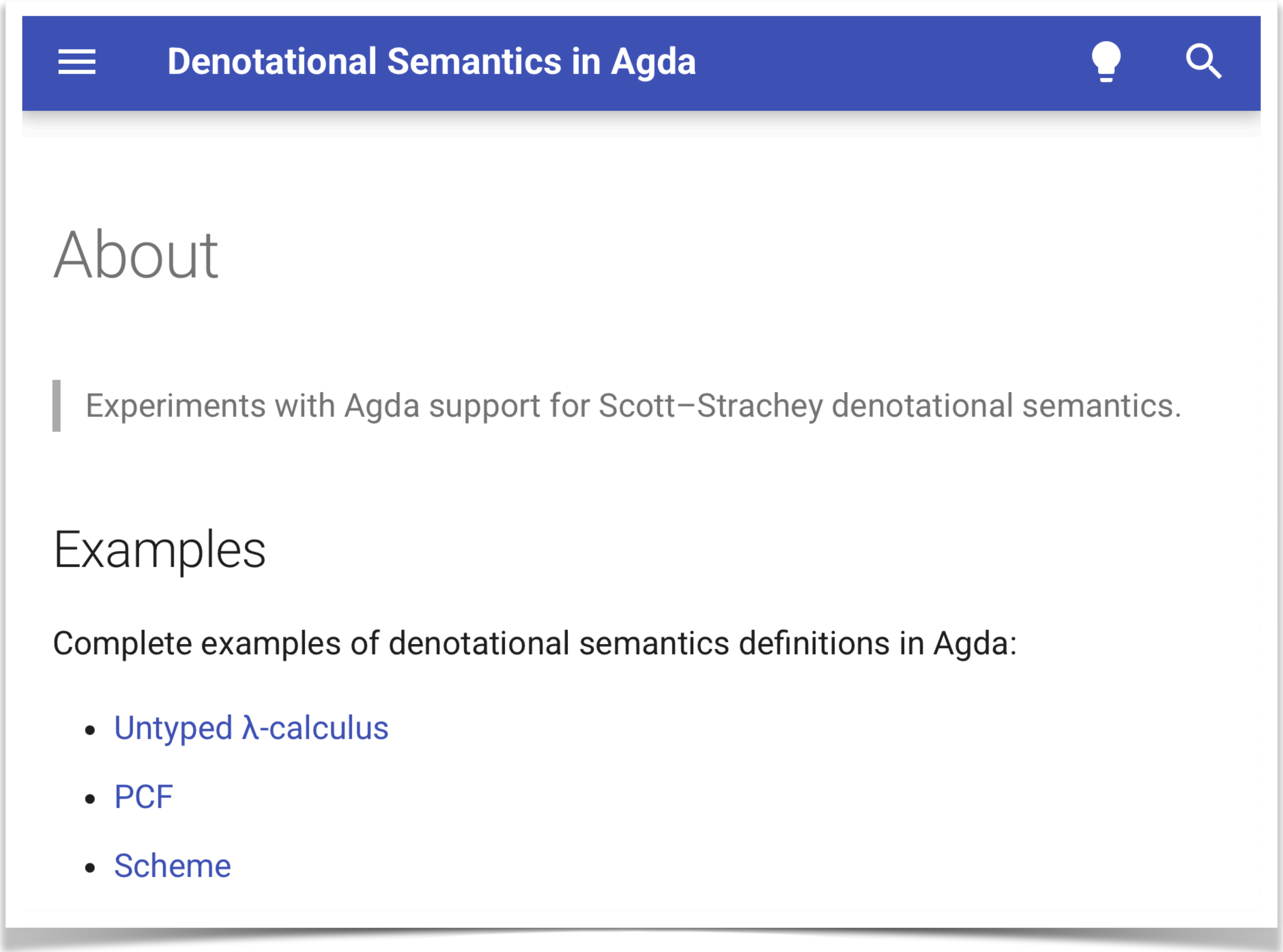
check-convergence : $(\lambda x_1 \,.\, x_{42})((\lambda x_0 \,.\, x_0 x_0)(\lambda x_0 \,.\, x_0 x_0)) \equiv x_{42}$

```
  ⟦ app (lam (x 1) (var x 42))
          (app (lam (x 0) (app (var x 0) (var x 0)))
               (lam (x 0) (app (var x 0) (var x 0)))) ⟧
  ≡ ⟦ var x 42 ⟧
check-convergence = refl
```

– potentially unsafe!

Other examples: PCF, Scheme

– pdmosses.github.io/xds-agda/



Denotational Semantics in Agda

About

> Experiments with Agda support for Scott–Strachey denotational semantics.

Examples

Complete examples of denotational semantics definitions in Agda:

- Untyped λ-calculus
- PCF
- Scheme

Safe **lightweight Agda formalization?**

– future work

Implement SDT (Synthetic Domain Theory)

- use ***plain*** Agda
- embed Agda types as ***predomains***
- assume only properties ***consistent*** with MLTT
- make functions ***implicitly*** continuous
- allow ***unrestricted*** recursive domain definitions
- …