Realizability Models for Type Theories

By Bernhard Reus (sections 2 and 3.1)

Thijs van de Griendt (s1040379)
Marloes Steenbergen (s1053015)

Radboud University

26-11-2025

Structure

1. Simply typed λ-calculus
2. Goal
3. Overview
4. Preliminaries
    1. PCA (recap)
    2. D-Set (morphisms, nabla ∇)
    3. Mod_D
    4. PER_D
5. A D-Set model for simply typed λ-calculus
6. Q&A

Simply typed λ-calculus

Recall simply typed λ-calculus consists of:

- Types: $\rightarrow \quad | \; T_{base}$
- Terms: var $\quad | \quad \lambda \quad | \quad \cdot$

Simply typed λ-calculus

Recall simply typed λ-calculus consists of:

- Types: $\rightarrow \quad | \; T_{base}$
- Terms: var $\quad | \quad \lambda \quad | \quad \cdot$

For the terms, the following rules apply:

- The variable rule (var):

$$\Gamma, x : A \vdash x : A$$

- The abstraction rule (λ):

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : A \rightarrow B}$$

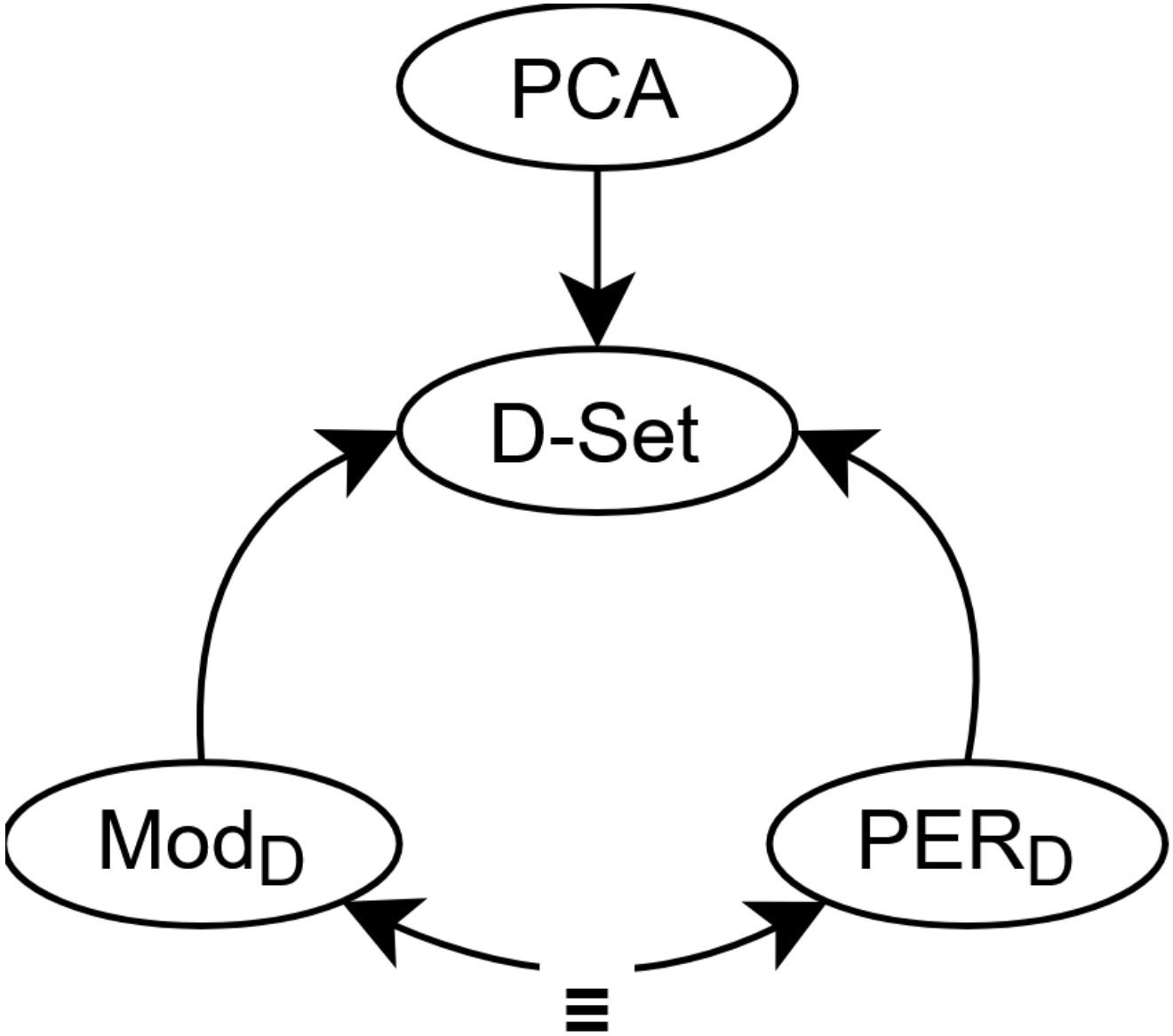
- The application rule ($\cdot$):

$$\frac{\Gamma \vdash f : A \rightarrow B \qquad \Gamma \vdash t : A}{\Gamma \vdash (f\ t) : B}$$

Goal

- Our goal is to create a model for simply typed λ-calculus using D-Sets.

Overview



Partial combinatory algebra (1)

Recall from the first presentation that:

Definition partial combinatory algebra

A quadruple $\langle D, \cdot, k, s \rangle$ where D is a set and $\_ \cdot \_ : D \times D \rightharpoonup D$ a partial application function is called a pca when $k, s \in D$ such that for any $x, y, z \in D$ it holds that:

$$k \cdot x \cdot y = x$$

$$s \cdot x \cdot y \downarrow \wedge s \cdot x \cdot y \cdot z \simeq (x \cdot z) \cdot (y \cdot z)$$

- $t \downarrow$: t is defined
- $t \uparrow$: t is undefined
- $s \simeq t$: $t \downarrow \wedge s \downarrow \wedge s = t$ or $t \uparrow \wedge s \uparrow$

Partial combinatory algebra (2)

- Using k, s and application we can create any function.
- Example: $s \cdot k \cdot k$ is the identity function.

Partial combinatory algebra (2)

- Using k, s and application we can create any function.
- Example: $s \cdot k \cdot k$ is the identity function.

Example lambda calculus

$k : \lambda xy.x$

$s : \lambda xyz.(xz)(yz)$

Partial combinatory algebra (2)

- Using k, s and application we can create any function.
- Example: $s \cdot k \cdot k$ is the identity function.

Example lambda calculus

$k : \lambda xy.x$
$s : \lambda xyz.(xz)(yz)$

- Pca's are combinatory complete.
- This allows us to write expressions like: $\Lambda n.t$.

D-Set (1)

Definition D-Set

Pick any pca for D. A D-set is denoted as $(X, \Vdash)$, where X is a set and $\Vdash$ a realizability relation $\Vdash \subseteq D \times X$ such that:

$$\forall x : X . \exists d : D . d \Vdash x$$

- $d \Vdash x$ is pronounced "d realizes x".
- Here d is the realizer that realizes entity x.

D-Set (2)

From a programmer's point of view, the realizer is the code that results in a mathematical entity.

Example D-Sets

- D: Any pca
- $X : \mathbb{N}$

Realizer	Entity
$\Lambda x.x$	0
$\Lambda x.fx$	1
$\Lambda x.f(fx)$	2
$\Lambda x.f^{\circ n}x$	$n\ (n > 0)$

D-Set (3)

Definition morphism between D-Sets

A morphism between D-sets $f : (X_1, \Vdash_1) \to (X_2, \Vdash_2)$ is a function $|f| : X_1 \to X_2$, such that there exists an $e \in D$ that realizes $|f|$:

$$\forall x : X_1 . \forall d : D . d \Vdash_1 x \implies (e \cdot d) \Vdash_2 |f|(x)$$

and $e \cdot d \downarrow$

D-Set (4)

Example morphism between D-Sets

- Say we have D-Sets: $(\mathbb{N}, \Vdash_{\mathbb{N}})$ and $(\mathbb{B}, \Vdash_{\mathbb{B}})$. We have $d \in D$ and $n \in \mathbb{N}$.
- even is a function from $\mathbb{N} \to \mathbb{B}$.
- There exists a realizer e for even.
- $d \Vdash_{\mathbb{N}} n$
- $e \cdot d \Vdash_{\mathbb{B}} \text{even}(n)$

- $e \cdot d \downarrow$

Intermezzo: Nabla and D-Sets

Any set can be made into a D-set using the full realizability relation $\nabla : \mathrm{Set} \to \text{D-Set}$.

Example morphism with ∇

- Say we have D-set $S : (A, \Vdash_S)$ and a set X. We have $d \in D$ and $a \in A$.
- $f : A \to X$ is a morphism between D-Sets A and ∇X.
- There exists a realizer $n \in D$.
- $d \Vdash_S a$
- $n \cdot d \Vdash_{\nabla X} f(a)$

Mod_D (1)

Definition modest D-Set

A D-Set $(X, \Vdash)$ is called modest if for any $n \in D$:

$$\forall x, x' : X . n \Vdash x \wedge n \Vdash x' \implies x = x'$$

The collection of all modest sets is called Mod_D.

Mod_D (2)

Clarification modest D-Set

We have D-Set $(X, \Vdash)$, where $d_1 \in D$ and $x_1, x_2 \in X$.

$$\left.\begin{array}{l} d_1 \Vdash x_1 \\ d_1 \Vdash x_2 \end{array}\right\} \text{then } x_1 = x_2$$

So a D-Set is modest, if every realizer realizes a single entity.

$\Lambda x.f^{\circ n}\mathrm{x} \Vdash \mathrm{n} \qquad (\text{for } n > 0)$

PER_D (1)

PER_D stands for partial equivalence relation (per) on realizers in D.

Definition PER_D-object

A D-set $(X, \Vdash_X)$ is called a PER_D-object when:

- $X \subseteq \mathcal{P}(D) \backslash \{\emptyset\}$
- $\forall A, B : X. A \neq B \implies A \cap B = \emptyset$
- $d \in A \iff d \Vdash_X A$

- PER_D is the set of all PER_D-objects.
- Mod_D and PER_D are equivalent.

PER_D (2)

Example PER_D-object

We have a D-Set $(X, \Vdash_X)$ with $x_1, x_2 \in X$ and $D := \{d_1, d_2, d_3\}$. We know that:

$$d_1 \Vdash x_1$$
$$d_2 \Vdash x_2$$
$$d_3 \Vdash x_2$$

- The power set of D is:
  $\{\{d_1\}, \{d_2\}, \{d_3\}, \{d_1, d_2\}, \{d_1, d_3\}, \{d_2, d_3\}, \{d_1, d_2, d_3\}\}$
- So X must be: $\{\{d_1\}, \{d_2, d_3\}\}$

Difference between Mod_D and PER_D

- Mod_D is the collection of all modest D-Sets.
- This collection is incredibly large.

- PER_D is the set of all PER_D-objects.
- Thus, PER_D is more tangible than Mod_D.

A D-Set model for simply typed λ-calculus

Recall that $\lambda \to$ consists of:

- types: $\to \quad | \; \mathsf{T}_{base}$
- terms: var $\quad | \quad \lambda \quad | \quad \cdot$

A D-Set model for simply typed λ-calculus

Recall that $\lambda \to$ consists of:

- types: $\to \quad | \; \mathsf{T}_{base}$
- terms: var $\quad | \quad \lambda \quad | \quad \cdot$

Now we want to define complete semantics for $\lambda \to$ in terms of D-sets.

A D-Set model for simply typed λ-calculus

Recall that $\lambda \to$ consists of:

- types: $\to \quad | \; \mathsf{T}_{base}$
- terms: var $\quad | \quad \lambda \quad | \quad \cdot$

Now we want to define complete semantics for $\lambda \to$ in terms of D-sets.

- $$[\![A]\!] \quad , \quad [\![B]\!] \quad \text{and} \quad [\![A \to B]\!]$$

A D-Set model for simply typed λ-calculus

Recall that $\lambda \to$ consists of:

- types: $\to \quad | \; \mathsf{T}_{base}$
- terms: var $\quad | \quad \lambda \quad | \quad \cdot$

Now we want to define complete semantics for $\lambda \to$ in terms of D-sets.

- $$[\![A]\!] \quad , \quad [\![B]\!] \quad \text{and} \quad [\![A \to B]\!]$$
- $$\text{var: } [\![x_1 : A_1, \ldots, x_i : A_i, \ldots x_n : A_n \vdash x_i : A_i]\!]$$

A D-Set model for simply typed λ-calculus

Recall that $\lambda \to$ consists of:

- types: $\to \quad | \; \mathsf{T}_{base}$
- terms: var $\quad | \quad \lambda \quad | \quad \cdot$

Now we want to define complete semantics for $\lambda \to$ in terms of D-sets.

- $$[\![A]\!] \quad , \quad [\![B]\!] \quad \text{and} \quad [\![A \to B]\!]$$
- $$\text{var: } [\![x_1 : A_1, \ldots, x_i : A_i, \ldots x_n : A_n \vdash x_i : A_i]\!]$$
- $$\text{abs: } \left[\!\!\left[\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.\, t : A \to B} \right]\!\!\right]$$

A D-Set model for simply typed λ-calculus

Recall that $\lambda \to$ consists of:

- types: $\to \quad | \; \mathsf{T}_{base}$
- terms: var $\quad | \quad \lambda \quad | \quad \cdot$

Now we want to define complete semantics for $\lambda \to$ in terms of D-sets.

- $$[\![A]\!] \quad , \quad [\![B]\!] \quad \text{and} \quad [\![A \to B]\!]$$
- $$\text{var: } [\![x_1 : A_1, \ldots, x_i : A_i, \ldots x_n : A_n \vdash x_i : A_i]\!]$$
- $$\text{abs: } \left[\!\!\left[\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.\, t : A \to B} \right]\!\!\right]$$
- $$\text{app: } \left[\!\!\left[\frac{\Gamma \vdash t_1 : A \to B \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1 \; t_2 : B} \right]\!\!\right]$$

Semantics for (Function) Types

How can we define the representation of arbitrary types A and B?

Semantics for (Function) Types

How can we define the representation of arbitrary types A and B?

We take $[\![A]\!] = (A, \vdash_A)$ and $[\![B]\!] = (B, \vdash_B)$

Semantics for (Function) Types

How can we define the representation of arbitrary types A and B?

We take $[\![\mathrm{A}]\!] = (\mathrm{A}, \vdash_A)$ and $[\![\mathrm{B}]\!] = (\mathrm{B}, \vdash_B)$

Now how can we define $[\![A \to B]\!]$?

Semantics for (Function) Types

How can we define the representation of arbitrary types A and B?

We take $[\![\mathrm{A}]\!] = (\mathrm{A}, \vdash_A)$ and $[\![\mathrm{B}]\!] = (\mathrm{B}, \vdash_B)$

Now how can we define $[\![A \to B]\!]$?

We can create a new D-set over the set F of morphisms $(A, \vdash_A) \to (B, \vdash_B)$
. . . including the according realisability relations.

$[\![A \to B]\!] = (F, \vdash_F)$
such that $e \vdash_F f$, with $e : D$ and $f : F$

Context and Judgement

Say we want to represent a judgement $[\![\Gamma \vdash t : B]\!]$
Then we need a way to represent context $\Gamma = x_1 : A_1, \ldots, x_n : A_n$

We can do this by taking Cartesian products:
$[\![\Gamma]\!] = [\![A_1]\!] * \ldots * [\![A_n]\!] = (A_1 * \ldots * A_n, \vdash_1 * \ldots * \vdash_n)$
(here: empty context excluded)

Now we can represent the full judgement $[\![\Gamma \vdash t : B]\!]$
by taking the D-Set over all

Variable Rule

We want to represent the variable rule with D-sets:
$[\![x_1 : A_1, \ldots, x_i : A_i, \ldots x_n : A_n \vdash x_i : A_i]\!]$

We know how to represent the context and the consequence:
$[\![\Gamma]\!] = (A_1 * \ldots * A_n, \vdash_1 * \ldots * \vdash_n)$
$[\![T]\!] = (A_i, \vdash_i)$

We can represent the variable rule as the morphism between these D-sets:
$[\![\Gamma]\!] \to [\![T]\!]$

Recall that a morphism between D-sets is a function f and a realiser e that tracks f

$f = A_1 * \cdots * A_i * \cdots * A_n \to A_i$
$e = \Lambda x.\pi_i x$

Lambda Abstraction Rule

We want to represent the abstraction rule with D-sets:

$$\left[\!\!\left[\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.\, t : A \to B} \right]\!\!\right]$$

Assume we have $[\![\Gamma, x : A \vdash t : B]\!]$
Prove that $[\![\Gamma \vdash \lambda x : A.\, t : A \to B]\!]$ follows.

$$[\![t]\!] = [\![\Gamma]\!] \times [\![A]\!] \longrightarrow [\![B]\!]$$

$$[\![\lambda x.\, t]\!] = [\![\Gamma]\!] \to ([\![A]\!] \to [\![B]\!])$$

$$\text{curry} : (X \times A \to B) \to (X \to (A \to B)).$$

$$[\![\lambda x.\, t]\!] = \text{curry}([\![t]\!]).$$

Application Rule

We want to represent the application rule with D-sets:

$$\text{app: } \left[\!\!\left[\frac{\Gamma \vdash t_1 : A \to B \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1\ t_2 : B} \right]\!\!\right]$$

This means that we can assume we have $[\![\Gamma \vdash t_1 : A \to B]\!]$ and $[\![\Gamma \vdash t_2 : A]\!]$
We need to prove that from this $[\![\Gamma \vdash t_1\ t_2 : B]\!]$ follows.

This has been left as an exercise for the observer.

Different models

"Note that analogously one could obtain a model working with PER_D-objects, modest D-sets, or even just D-sets instead of pers." [1]

[1] Reus, B. (1999). Realizability Models for Type Theories. Electronic Notes in Theoretical Computer Science, 23(1), 128–158. https://doi.org/10.1016/s1571-0661(04)00108-2

Q&A

Are there any questions?