Extracting Ω's programs from proofs in the calculus of constructions

Examples

by C. Paulin-Mohring

Sem Stammen and Ben van Wijngaarden

December 5, 2025

Overview

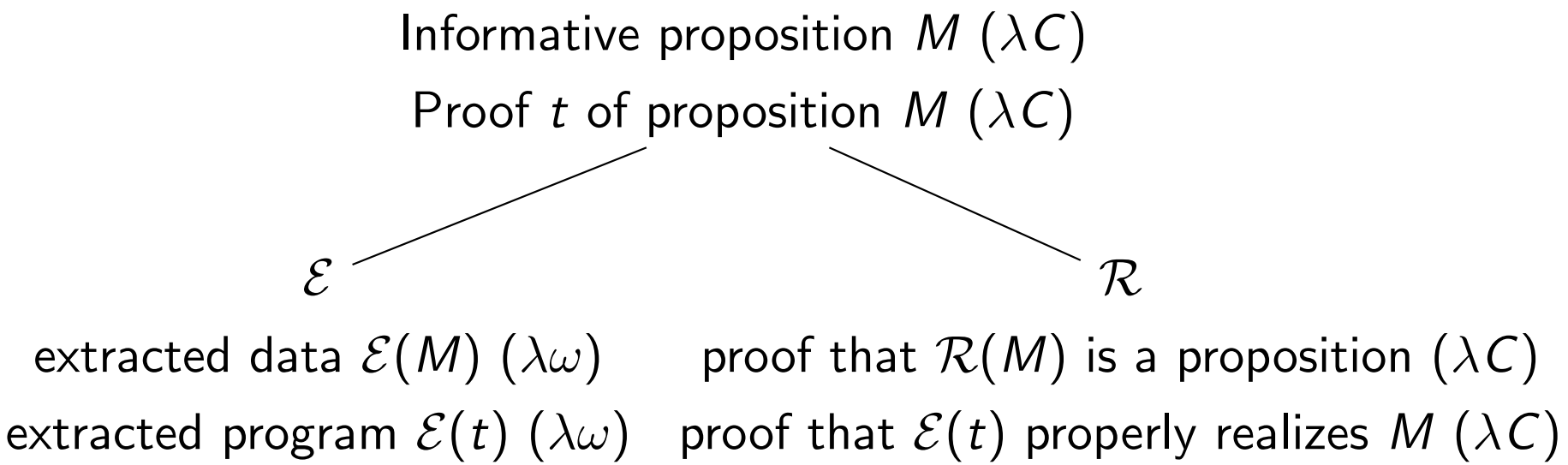
The aim of the paper (1)

Defining a notion of **realizability** for a **proof language** to a **programming language**

proof language	$= \lambda C$	$\approx$	Rocq
	$\downarrow$ extraction		
programming language	$= \lambda\omega$	$\approx$	Haskell/Ocaml

The aim of the paper (2)

- We use an extraction function $\mathcal{E}$
- And realizability predicate $\mathcal{R}$



Programming language: $\lambda\omega$

- Basic theory behind Haskell/Ocaml
- No dependant types
- Λ_D the set of all possible terms we can make in this system
- The programming language works with orders, data type schemes and programs

Proof language: λC

- λC is a language where we write proofs.

Proof language: λC

- λC is a language where we write proofs.
- Here λC is presented as the **Calculus of Constructions (CoC) with realizations**.

Proof language: λC

- λC is a language where we write proofs.
- Here λC is presented as the **Calculus of Constructions (CoC) with realizations**.
- In the original CoC, we have one type for propositions: Prop.

Proof language: λC

- λC is a language where we write proofs.
- Here λC is presented as the **Calculus of Constructions (CoC) with realizations**.
- In the original CoC, we have one type for propositions: Prop.
- CoC with realizations instead has two versions of propositional types: Spec and Prop.

Proof language: λC

- λC is a language where we write proofs.
- Here λC is presented as the **Calculus of Constructions (CoC) with realizations**.
- In the original CoC, we have one type for propositions: Prop.
- CoC with realizations instead has two versions of propositional types: Spec and Prop.
- The key idea: Spec and Prop are used to mark which propositions are meaningful in a programming context.

Proof language: λC

Three levels of terms:

1. Propositional types (terms of type Type)
   - $Spec : Type$
   - $Prop : Type$
   - $(\Pi x : A.B) : Type$ if $type(B) = Type$

Proof language: λC

Three levels of terms:

1. Propositional types (terms of type Type)
   - $Spec : Type$
   - $Prop : Type$
   - $(\Pi x : A.B) : Type$ if $type(B) = Type$
2. Propositional schemes (things of type a propositional type)
   - $even\ 2 : Prop$
   - $(\lambda x : Prop.x) : (\Pi x : Prop.Prop)$

Proof language: λC

Three levels of terms:

1. Propositional types (terms of type Type)
   - $Spec : Type$
   - $Prop : Type$
   - $(\Pi x : A.B) : Type$ if $type(B) = Type$
2. Propositional schemes (things of type a propositional type)
   - $even\ 2 : Prop$
   - $(\lambda x : Prop.x) : (\Pi x : Prop.Prop)$

   Propositions (terms of type Prop, Spec)
   - $even\ 2 : Prop$
   - $even\ 2 : Spec$

Proof language: λC

Three levels of terms:

1. Propositional types (terms of type Type)
   - $Spec : Type$
   - $Prop : Type$
   - $(\Pi x : A.B) : Type$ if $type(B) = Type$
2. Propositional schemes (things of type a propositional type)
   - $even\ 2 : Prop$
   - $(\lambda x : Prop.x) : (\Pi x : Prop.Prop)$

   Propositions (terms of type Prop, Spec)
   - $even\ 2 : Prop$
   - $even\ 2 : Spec$
3. Proofs (things of type a proposition)
   - $evenproof : even\ 2$

Terms form $\lambda\omega$ are also included, using a $_D$ notation (example: $\lambda x :_D A.x$)

Informative vs non-informative terms

For a term $M \in \Lambda_R$ we have that:

M is non-informative

Informative vs non-informative terms

For a term $M \in \Lambda_R$ we have that:

M is non-informative

if $M = \mathsf{Prop}$

Informative vs non-informative terms

For a term $M \in \Lambda_R$ we have that:

M is non-informative

if $M = \text{Prop}$

if $M = x,\ x : N$ with N non-informative

Informative vs non-informative terms

For a term $M \in \Lambda_R$ we have that:

M is non-informative

if $M = \mathsf{Prop}$

if $M = x,\ x : N$ with N non-informative

if $M = \Pi x : P.\, N$ or $\Pi x :_D P.\, N$ with N non-informative

Informative vs non-informative terms

For a term $M \in \Lambda_R$ we have that:

M is non-informative

if $M = \mathsf{Prop}$

if $M = x,\ x : N$ with N non-informative

if $M = \Pi x : P.\, N$ or $\Pi x :_D P.\, N$ with N non-informative

if $M = \lambda z : P.\, N$ or $\lambda x :_D P.\, N$ with N non-informative

Informative vs non-informative terms

For a term $M \in \Lambda_R$ we have that:

M is non-informative

if $M = \mathsf{Prop}$

if $M = x,\ x : N$ with N non-informative

if $M = \Pi x : P.\, N$ or $\Pi x :_D P.\, N$ with N non-informative

if $M = \lambda z : P.\, N$ or $\lambda x :_D P.\, N$ with N non-informative

if $M = (N\, P)$ or $({}_D N\, P)$ with N non-informative

Informative vs non-informative terms

For a term $M \in \Lambda_R$ we have that:

M is non-informative

- if $M = \text{Prop}$
- if $M = x,\ x : N$ with N non-informative
- if $M = \Pi x : P.\, N$ or $\Pi x :_D P.\, N$ with N non-informative
- if $M = \lambda z : P.\, N$ or $\lambda x :_D P.\, N$ with N non-informative
- if $M = (N\, P)$ or $({}_D N\, P)$ with N non-informative

otherwise M is informative

Realizability

- We can extract informative propositions (M : Spec) to get 'set' of candidate realizers ($x : \mathcal{E}(M)$)
- We build a realizability predicate to get the 'good' candidates

$$\vdash M : \text{Spec}$$
$$\vdash t : M$$

$$\mathcal{E} \qquad\qquad \mathcal{R}$$

$$\vdash_{\lambda\omega} \mathcal{E}(M) : \text{Data} \qquad\qquad x : \mathcal{E}(M) \vdash \mathcal{R}(M, x) : \text{Prop}$$
$$\vdash_{\lambda\omega} \mathcal{E}(t) : \mathcal{E}(M) \qquad\qquad \vdash u : \mathcal{R}(M, \mathcal{E}(t))$$

Realizability

We can do more

1. We can extract programs from proofs
2. We can realize non-informative propositions
3. We can describe properties we want and if they are realizable we can add them as axioms. (Consistency)

Extraction: $\mathcal{E}$

$\mathcal{E}$ is a function that returns the term, type, environment, but with non-informative parts removed.

Theorem

If $\Gamma \vdash M : N$ *with* M *an informative term, then* $\mathcal{E}(\Gamma) \vdash_{\lambda\omega} \mathcal{E}(M) : \mathcal{E}(N)$

Extraction on informative propositional types

$\mathcal{E}$ is defined for informative propositional types ($M = Spec$ or $M = \Pi X : A.B$)

- If $M = Spec$, then $\mathcal{E}(M) = Data$
- If $M = \Pi X : A.B$, then

$$\mathcal{E}(M) = \begin{cases} \mathcal{E}(A) \to \mathcal{E}(B) & \text{if } A \text{ is an informative propositional type or an order} \\ \mathcal{E}(B) & \text{otherwise} \end{cases}$$

Extraction on informative propositional schemes

$\mathcal{E}$ is also defined for informative propositional schemes

- If M is a variable X, then $\mathcal{E}(M) = \bar{X}$

Extraction on informative propositional schemes

$\mathcal{E}$ is also defined for informative propositional schemes

- If M is a variable X, then $\mathcal{E}(M) = \bar{X}$
- If $M = \Pi x : A.B$, then

$$\mathcal{E}(M) = \begin{cases} \Pi \bar{x} : \mathcal{E}(A).\mathcal{E}(B) & \text{if } A \text{ is informative or in } \Lambda_D \\ \mathcal{E}(B) & \text{otherwise} \end{cases}$$

Extraction on informative propositional schemes (2)

- If $M = \lambda x : A.B$, then

$$\mathcal{E}(M) = \begin{cases} \lambda x : \mathcal{E}(A).\mathcal{E}(B) & \text{if } A \text{ is informative propositional type or an order} \\ \mathcal{E}(B) & \text{otherwise} \end{cases}$$

Extraction on informative propositional schemes (2)

- If $M = \lambda x : A.B$, then

$$\mathcal{E}(M) = \begin{cases} \lambda x : \mathcal{E}(A).\mathcal{E}(B) & \text{if } A \text{ is informative propositional type or an order} \\ \mathcal{E}(B) & \text{otherwise} \end{cases}$$

- If $M = AB$, then

$$\mathcal{E}(M) = \begin{cases} \mathcal{E}(A)\mathcal{E}(B) & \text{if } B \text{ is an informative propositional scheme or a data type scheme} \\ \mathcal{E}(B) & \text{else} \end{cases}$$

Realizability predicate: $\mathcal{R}$

- $\mathcal{R}$ is defined for propositions
- Informally: $\mathcal{R}(M, t)$ is condition that 'checks' that t is a proper realizer of proposition M.

Definition

A term t *realizes* an informative proposition M in Γ if $\mathcal{R}(M, t)$ is provable in $\mathcal{R}(\Gamma)$. That is: $\mathcal{R}(\Gamma) \vdash u : \mathcal{R}(P, t)$.

Definition

An informative proposition M is realizable if there is a term t that realizes it.

Realizability predicate: $\mathcal{R}$

- If $P = \text{Spec}$, then

$$\mathcal{R}(\text{Spec}, r) = r \to \text{Prop} \quad \text{, where } r : \mathcal{E}(\text{Spec}) = \text{Data}.$$

Realizability predicate: $\mathcal{R}$

- If $P = \text{Spec}$, then

$$\mathcal{R}(\text{Spec}, r) = r \to \text{Prop} \quad \text{, where } r : \mathcal{E}(\text{Spec}) = \text{Data}.$$

- If $P = X$ and $X : M$, then

$$\mathcal{R}(X) = X \quad \text{, where } X : \mathcal{R}(M, \bar{X}) \text{ and } \bar{X} : \mathcal{E}(M).$$

Realizability predicate: $\mathcal{R}$

- If $P = \text{Spec}$, then

  $$\mathcal{R}(\text{Spec}, r) = r \to \text{Prop} \quad \text{, where } r : \mathcal{E}(\text{Spec}) = \text{Data}.$$

- If $P = X$ and $X : M$, then

  $$\mathcal{R}(X) = X \quad \text{, where } X : \mathcal{R}(M, \bar{X}) \text{ and } \bar{X} : \mathcal{E}(M).$$

- If $P = \Pi x : M.N$ with N an informative proposition, and let r be a variable of type $\mathcal{E}(P)$. then

  If M is informative:

  $$\mathcal{R}(P, r) = \Pi \bar{x} : \mathcal{E}(M).\Pi x : \mathcal{R}(M, \bar{x}).\mathcal{R}(N, (r\ \bar{x}))$$

Realizability predicate: $\mathcal{R}$

- If $P = \text{Spec}$, then

  $$\mathcal{R}(\text{Spec}, r) = r \to \text{Prop} \quad \text{, where } r : \mathcal{E}(\text{Spec}) = \text{Data}.$$

- If $P = X$ and $X : M$, then

  $$\mathcal{R}(X) = X \quad \text{, where } X : \mathcal{R}(M, \bar{X}) \text{ and } \bar{X} : \mathcal{E}(M).$$

- If $P = \Pi x : M.N$ with N an informative proposition, and let r be a variable of type $\mathcal{E}(P)$. then

  If M is informative:

  $$\mathcal{R}(P, r) = \Pi \bar{x} : \mathcal{E}(M).\Pi x : \mathcal{R}(M, \bar{x}).\mathcal{R}(N, (r\ \bar{x}))$$

  If M is non-informative:

  $$\mathcal{R}(P, r) = \Pi x : \mathcal{R}(M).\mathcal{R}(N, r)$$

Realizability predicate: $\mathcal{R}$

- If $P = \text{Spec}$, then

  $$\mathcal{R}(\text{Spec}, r) = r \to \text{Prop} \quad \text{, where } r : \mathcal{E}(\text{Spec}) = \text{Data}.$$

- If $P = X$ and $X : M$, then

  $$\mathcal{R}(X) = X \quad \text{, where } X : \mathcal{R}(M, \bar{X}) \text{ and } \bar{X} : \mathcal{E}(M).$$

- If $P = \Pi x : M.N$ with N an informative proposition, and let r be a variable of type $\mathcal{E}(P)$. then

  If M is informative:

  $$\mathcal{R}(P, r) = \Pi \bar{x} : \mathcal{E}(M).\Pi x : \mathcal{R}(M, \bar{x}).\mathcal{R}(N, (r\ \bar{x}))$$

  If M is non-informative:

  $$\mathcal{R}(P, r) = \Pi x : \mathcal{R}(M).\mathcal{R}(N, r)$$

  If $M \in \Lambda_D$:

  $$\Pi x : M.\mathcal{R}(N, r\ \bar{x})$$

Realizability predicate: $\mathcal{R}$

- If $P = \lambda x : M.N$ with N an informative propositional scheme:
- For propositional schemes M we have that $\mathcal{R}(M, r) = \mathcal{R}(M)\ r$

Realizability predicate: $\mathcal{R}$

- If $P = \lambda x : M.N$ with N an informative propositional scheme:
- For propositional schemes M we have that $\mathcal{R}(M, r) = \mathcal{R}(M)\ r$

if M is an informative propositional type

$$\mathcal{R}(P) = \lambda \bar{x} : \mathcal{E}(M).\lambda x : \mathcal{R}(M, \bar{x}).\mathcal{R}(N)$$

Realizability predicate: $\mathcal{R}$

- If $P = \lambda x : M.N$ with N an informative propositional scheme:
- For propositional schemes M we have that $\mathcal{R}(M, r) = \mathcal{R}(M)\ r$

if M is an informative propositional type

$$\mathcal{R}(P) = \lambda \bar{x} : \mathcal{E}(M).\lambda x : \mathcal{R}(M, \bar{x}).\mathcal{R}(N)$$

if M is non-informative propositional type

$$\mathcal{R}(P) = \lambda x : \mathcal{R}(M).\mathcal{R}(N)$$

Realizability predicate: $\mathcal{R}$

- If $P = \lambda x : M.N$ with N an informative propositional scheme:
- For propositional schemes M we have that $\mathcal{R}(M, r) = \mathcal{R}(M)\ r$

if M is an informative propositional type

$$\mathcal{R}(P) = \lambda \bar{x} : \mathcal{E}(M).\lambda x : \mathcal{R}(M, \bar{x}).\mathcal{R}(N)$$

if M is non-informative propositional type

$$\mathcal{R}(P) = \lambda x : \mathcal{R}(M).\mathcal{R}(N)$$

if M is an informative proposition

$$\mathcal{R}(P) = \lambda \bar{x} : \mathcal{E}(M).\mathcal{R}(N)$$

Realizability predicate: $\mathcal{R}$

- If $P = \lambda x : M.N$ with N an informative propositional scheme:
- For propositional schemes M we have that $\mathcal{R}(M, r) = \mathcal{R}(M)\ r$

if M is an informative propositional type

$$\mathcal{R}(P) = \lambda \bar{x} : \mathcal{E}(M).\lambda x : \mathcal{R}(M, \bar{x}).\mathcal{R}(N)$$

if M is non-informative propositional type

$$\mathcal{R}(P) = \lambda x : \mathcal{R}(M).\mathcal{R}(N)$$

if M is an informative proposition

$$\mathcal{R}(P) = \lambda \bar{x} : \mathcal{E}(M).\mathcal{R}(N)$$

if M is an non-informative proposition

$$\mathcal{R}(P) = \mathcal{R}(N)$$

Realizability predicate: $\mathcal{R}$

- If $P = \lambda x : M.N$ with N an informative propositional scheme:
- For propositional schemes M we have that $\mathcal{R}(M, r) = \mathcal{R}(M)\ r$

  if M is an informative propositional type
  $\mathcal{R}(P) = \lambda \bar{x} : \mathcal{E}(M).\lambda x : \mathcal{R}(M, \bar{x}).\mathcal{R}(N)$

  if M is non-informative propositional type
  $\mathcal{R}(P) = \lambda x : \mathcal{R}(M).\mathcal{R}(N)$

  if M is an informative proposition
  $\mathcal{R}(P) = \lambda \bar{x} : \mathcal{E}(M).\mathcal{R}(N)$

  if M is an non-informative proposition
  $\mathcal{R}(P) = \mathcal{R}(N)$

- "The other cases may be deduced easily"

Examples

Things we can do with this notion of Realizability

1. Extracting programs from proofs
   - Booleans
   - Non-informative Disjunction
   - Existential quantifier
2. Realizing non-informative propositions
   - Absurdity
3. Consistency
   - Subset

Booleans

$$\mathsf{bool} = \Pi C : \mathsf{Spec}.C \to C \to C$$

$$\mathsf{true} = \lambda C : \mathsf{Spec}.\lambda t : C.\lambda f : C.t$$

$$\vdash \mathsf{bool} : \mathsf{Spec}$$

$$\vdash \mathsf{true} : \mathsf{bool}$$

$\mathcal{E}$

$$\vdash_{\lambda\omega} \mathcal{E}(\mathsf{bool}) : \mathsf{Data}$$

$$\vdash_{\lambda\omega} \mathcal{E}(\mathsf{true}) : \mathcal{E}(\mathsf{bool})$$

$\mathcal{R}$

$$x : \mathcal{E}(\mathsf{bool}) \vdash \mathcal{R}(\mathsf{bool}, x) : \mathsf{Prop}$$

$$\vdash u : \mathcal{R}(\mathsf{bool}, \mathcal{E}(\mathsf{true}))$$

Booleans

$$\mathsf{bool} = \Pi C : \mathsf{Spec}.\, C \to C \to C$$

$$\mathsf{true} = \lambda C : \mathsf{Spec}.\lambda t : C.\lambda f : C.\, t$$

$$\begin{aligned}
\mathcal{E}(\mathsf{bool}) &= \mathcal{E}(\Pi C : \mathsf{Spec}.\, C \to C \to C) \\
&= \Pi \bar{C} : \mathcal{E}(\mathsf{Spec}).\mathcal{E}(C \to C \to C) \\
&= \Pi \bar{C} : \mathsf{Data}.\mathcal{E}(C) \to \mathcal{E}(C \to C) \\
&= \Pi \bar{C} : \mathsf{Data}.\mathcal{E}(C) \to \mathcal{E}(C) \to \mathcal{E}(C) \\
&= \Pi \bar{C} : \mathsf{Data}.\bar{C} \to \bar{C} \to \bar{C}
\end{aligned}$$

Booleans

$$\mathsf{bool} = \Pi C : \mathsf{Spec}.C \to C \to C$$

$$\mathcal{E}(\mathsf{bool}) = \Pi \bar{C} : \mathsf{Data}.\bar{C} \to \bar{C} \to \bar{C}$$

$$\mathsf{true} = \lambda C : \mathsf{Spec}.\lambda t : C.\lambda f : C.t$$

$$\begin{aligned}
\mathcal{E}(\mathsf{true}) &= \mathcal{E}(\lambda C : \mathsf{Spec}.\lambda t : C.\lambda f : C.t) \\
&= \lambda \bar{C} : \mathcal{E}(\mathsf{Spec})\mathcal{E}(\lambda t : C.\lambda f : C.t)) \\
&= \lambda \bar{C} : \mathsf{Data}\lambda \bar{t} : \mathcal{E}(C).\mathcal{E}(\lambda f : C.t) \\
&= \lambda \bar{C} : \mathsf{Data}\lambda \bar{t} : \mathcal{E}(C).\lambda \bar{f} : \mathcal{E}(C).\mathcal{E}(t) \\
&= \lambda \bar{C} : \mathsf{Data}\lambda \bar{t} : \bar{C}.\lambda \bar{f} : \bar{C}.\bar{t}
\end{aligned}$$

Booleans

$$\mathsf{bool} = \Pi C : \mathsf{Spec}.C \to C \to C$$
$$\mathcal{E}(\mathsf{bool}) = \Pi \bar{C} : \mathsf{Data}.\bar{C} \to \bar{C} \to \bar{C}$$
$$\mathsf{true} = \lambda C : \mathsf{Spec}.\lambda t : C.\lambda f : C.t$$
$$\mathcal{E}(\mathsf{true}) = \lambda \bar{C} : \mathsf{Data}\lambda \bar{t} : \bar{C}.\lambda \bar{f} : \bar{C}.\bar{t}$$

$$\begin{aligned}\mathcal{R}(bool, r) &= \mathcal{R}(\Pi C : \mathsf{Spec}.C \to C \to C, r)\\ &= \Pi \bar{C} : \mathcal{E}(\mathsf{Spec}).\Pi C_p : \mathcal{R}(\mathsf{Spec}, \bar{C}).\mathcal{R}(C \to C \to C, r\ \bar{C})\\ &= \Pi \bar{C} : \mathsf{Data}.\Pi C_p : \bar{C} \to \mathsf{Prop}.\mathcal{R}(C \to C \to C, r\ \bar{C})\end{aligned}$$

$$\begin{aligned}&\mathcal{R}(C \to C \to C, r\ \bar{C}) =\\ &= \Pi \bar{x} : \mathcal{E}(C).\mathcal{R}(C, \bar{x}) \to \mathcal{R}(C \to C, (r\ \bar{C})\ \bar{x})\\ &= \Pi \bar{x} : \mathcal{E}(C).\mathcal{R}(C, \bar{x}) \to \Pi \bar{y} : \mathcal{E}(C).\mathcal{R}(C, \bar{y}) \to \mathcal{R}(C, ((r\ \bar{C})\ \bar{x})\ \bar{y})\\ &= \Pi \bar{x} : \bar{C}.C_p\ \bar{x} \to \Pi \bar{y} : \bar{C}.C_p\ \bar{y} \to C_p\ ((r\ \bar{C})\ \bar{x})\ \bar{y})\end{aligned}$$

Booleans

$$\mathsf{bool} = \Pi C : \mathsf{Spec}.C \to C \to C$$

$$\mathcal{E}(\mathsf{bool}) = \Pi \bar{C} : \mathsf{Data}.\bar{C} \to \bar{C} \to \bar{C}$$

$$\mathsf{true} = \lambda C : \mathsf{Spec}.\lambda t : C.\lambda f : C.t$$

$$\mathcal{E}(\mathsf{true}) = \lambda \bar{C} : \mathsf{Data}\lambda \bar{t} : \bar{C}.\lambda \bar{f} : \bar{C}.\bar{t}$$

$$\mathcal{R}(bool, r) = \Pi \bar{C} : \mathsf{Data}.\Pi C_p : \bar{C} \to \mathsf{Prop}.$$
$$\Pi \bar{x} : \bar{C}.C_p\ \bar{x} \to \Pi \bar{y} : \bar{C}.C_p\ \bar{y} \to C_p\ ((r\ \bar{C})\ \bar{x})\ \bar{y})$$

We have that $\lambda \bar{C}.\lambda C_p.\lambda \bar{x}.\lambda p_1.\lambda \bar{y}.\lambda p_2.p_1 : \mathcal{R}(\mathsf{bool}, \mathcal{E}(\mathsf{true}))$ so $\mathcal{E}(\mathsf{true})$ is a proper realizer of bool.

Non-informative disjunction

A, B non-informative

$$A \vee B = \Pi C : \mathsf{Spec}.(A \to C) \to (B \to C) \to C$$

$$\mathsf{inl} = \lambda a : A.\lambda C : \mathsf{Spec}.\lambda f : A \to C.\lambda g : B \to C.f\ a$$

$$\vdash A \to A \vee B : \mathsf{Spec}$$

$$\vdash \mathsf{inl} : A \to A \vee B$$

$\mathcal{E}$ $\qquad$ $\mathcal{R}$

$$\vdash_{\lambda\omega} \mathcal{E}(A \to A \vee B) : \mathsf{Data}$$

$$\vdash_{\lambda\omega} \mathcal{E}(\mathsf{inl}) : \mathcal{E}(A \to A \vee B)$$

$$x : \mathcal{E}(A \to A \vee B) \vdash \mathcal{R}(A \to A \vee B, x) : \mathsf{Prop}$$

$$\vdash u : \mathcal{R}(A \to A \vee B, \mathcal{E}(\mathsf{inl}))$$

Non-informative disjunction

$$A, B \text{ non-informative}$$

$$A \vee B = \Pi C : \mathrm{Spec}.(A \to C) \to (B \to C) \to C$$

$$\mathrm{inl} = \Pi a : A.\lambda C : \mathrm{Spec}.\lambda f : A \to C.\lambda g : B \to C.f\ a$$

$$\begin{aligned}
\mathcal{E}(A \to A \vee B) &= \mathcal{E}(A \vee B) \\
&= \mathcal{E}(\Pi C : \mathrm{Spec}.(A \to C) \to (B \to C) \to C) \\
&= \Pi \bar{C} : \mathrm{Data}.\mathcal{E}(A \to C) \to \mathcal{E}(B \to C) \to \mathcal{E}(C)) \\
&= \Pi \bar{C} : \mathrm{Data}.\mathcal{E}(C) \to \mathcal{E}(C) \to \mathcal{E}(C)) \\
&= \Pi \bar{C} : \mathrm{Data}.\bar{C} \to \bar{C} \to \bar{C} \\
&= \mathrm{bool}
\end{aligned}$$

Non-informative disjunction

$$A, B \text{ non-informative}$$

$$A \vee B = \Pi C : \mathsf{Spec}.(A \to C) \to (B \to C) \to C$$

$$\mathcal{E}(A \to A \vee B) = \Pi \bar{C} : \mathsf{Data}.\bar{C} \to \bar{C} \to \bar{C} = \mathsf{bool}$$

$$\mathsf{inl} = \lambda a : A.\lambda C : \mathsf{Spec}.\lambda f : A \to C.\lambda g : B \to C.f\ a$$

$$\begin{aligned}
\mathcal{E}(\mathsf{inl}) &= \mathcal{E}(\lambda a : A.\lambda C : \mathsf{Spec}.\lambda f : A \to C.\lambda g : B \to C.f\ a) \\
&= \mathcal{E}(\lambda C : \mathsf{Spec}.\lambda f : A \to C.\lambda g : B \to C.f\ a) \\
&= \lambda \bar{C} : \mathcal{E}(\mathsf{Spec}).\lambda \bar{f} : \mathcal{E}(A \to C).\lambda \bar{g} : \mathcal{E}(B \to C).\mathcal{E}(f\ a) \\
&= \lambda \bar{C} : \mathsf{Data}.\lambda \bar{f} : \mathcal{E}(C).\lambda \bar{g} : \mathcal{E}(C).\mathcal{E}(f) \\
&= \lambda \bar{C} : \mathsf{Data}.\lambda \bar{f} : \bar{C}.\lambda \bar{g} : \bar{C}.\bar{f} \\
&= \mathsf{true}
\end{aligned}$$

Non-informative disjunction

$$A, B \text{ non-informative}$$

$$A \vee B = \Pi C : \mathsf{Spec}.(A \to C) \to (B \to C) \to C$$

$$\mathcal{E}(A \to A \vee B) = \Pi \bar{C} : \mathsf{Data}.\bar{C} \to \bar{C} \to \bar{C} = \mathsf{bool}$$

$$\mathsf{inl} = \Pi a : A.\lambda C : \mathsf{Spec}.\lambda f : A \to C.\lambda g : B \to C.f\ a$$

$$\mathcal{E}(\mathsf{inl}) = \lambda \bar{C} : \mathsf{Data}.\lambda \bar{f} : \bar{C}.\lambda \bar{g} : \bar{C}.\bar{f} = \mathsf{true}$$

$$\begin{aligned}
&\mathcal{R}(A \to A \vee B, r) = \mathcal{R}(A \vee B, r) \\
&= \mathcal{R}(\Pi C : \mathsf{Spec}.(A \to C) \to (B \to C) \to C, r) \\
&= \Pi \bar{C} : \mathcal{E}(\mathsf{Spec}).\Pi C_p : \mathcal{R}(\mathsf{Spec}, \bar{C}).\mathcal{R}((A \to C) \to (B \to C) \to C, r\ \bar{C}) \\
&= \Pi \bar{C} : \mathsf{Data}.\Pi C_p : \bar{C} \to \mathsf{Prop}.\mathcal{R}((A \to C) \to (B \to C) \to C, r\ \bar{C})
\end{aligned}$$

Non-informative disjunction

$$\mathcal{R}(A \to A \vee B, r) = \mathcal{R}(A \vee B, r)$$
$$= \Pi \bar{C} : \mathsf{Data}.\Pi C_p : \bar{C} \to \mathsf{Prop}.\mathcal{R}((A \to C) \to (B \to C) \to C, r\ \bar{C})$$

$$\mathcal{R}((A \to C) \to (B \to C) \to C, r\ \bar{C}) =$$
$$\Pi \bar{x} : \mathcal{E}(A \to C).\mathcal{R}(A \to C, \bar{x}) \to \mathcal{R}((B \to C) \to C, (r\ \bar{C})\ \bar{x})$$
$$= \Pi \bar{x} : \mathcal{E}(A \to C).\mathcal{R}(\mathbf{A} \to \mathbf{C}, \bar{\mathbf{x}}) \to$$
$$\Pi \bar{y} : \mathcal{E}(B \to C).\mathcal{R}(\mathbf{B} \to \mathbf{C}, \bar{\mathbf{y}}) \to \mathcal{R}(C, ((r\ \bar{C})\ \bar{x})\ \bar{y})$$
$$= \Pi \bar{x} : \mathcal{E}(A \to C).(\mathcal{R}(\mathbf{A}) \to \mathcal{R}(\mathbf{C}, \bar{\mathbf{x}})) \to$$
$$\Pi \bar{y} : \mathcal{E}(B \to C).(\mathcal{R}(\mathbf{B}) \to \mathcal{R}(\mathbf{C}, \bar{\mathbf{y}})) \to \mathcal{R}(C, ((r\ \bar{C})\ \bar{x})\ \bar{y})$$
$$= \Pi \bar{x} : \bar{C}.(\mathcal{R}(A) \to C_p\ \bar{x}) \to$$
$$\Pi \bar{y} : \bar{C}.(\mathcal{R}(B) \to C_p\ \bar{y}) \to C_p\ (((r\ \bar{C})\ \bar{x})\ \bar{y})$$

Non-informative disjunction

$$A, B \text{ non-informative}$$

$$A \vee B = \Pi C : \mathsf{Spec}.(A \to C) \to (B \to C) \to C$$

$$\mathcal{E}(A \to A \vee B) = \Pi \bar{C} : \mathsf{Data}.\bar{C} \to \bar{C} \to \bar{C} = \mathsf{bool}$$

$$\mathsf{inl} = \Pi a : A.\lambda C : \mathsf{Spec}.\lambda f : A \to C.\lambda g : B \to C.f\ a$$

$$\mathcal{E}(\mathsf{inl}) = \lambda \bar{C} : \mathsf{Data}.\lambda \bar{f} : \bar{C}.\lambda \bar{g} : \bar{C}.\bar{f} = \mathsf{true}$$

$$\mathcal{R}(A \to A \vee B, r) =$$
$$\Pi \bar{C} : \mathsf{Data}.\Pi C_p : \bar{C} \to \mathsf{Prop}.$$
$$\Pi \bar{x} : \bar{C}.(\mathcal{R}(A) \to C_p\ \bar{x}) \to$$
$$\Pi \bar{y} : \bar{C}.(\mathcal{R}(B) \to C_p\ \bar{y}) \to C_p\ (((r\ \bar{C})\ \bar{x})\ \bar{y})$$

We have that $\lambda \bar{C}.\lambda C_p.\lambda \bar{x}.\lambda f.\lambda \bar{y}.\lambda g.C_p\ \bar{x} : \mathcal{R}(A \to A \vee B, \mathcal{E}(\mathsf{inl}))$ so $\mathcal{E}(\mathsf{inl}) = \mathsf{true}$ is a proper realizer of $A \vee B$.

Existential quantifier

We define an existential quantifier:

$$\Sigma_A(\Phi) \equiv \Pi C : \mathit{Spec}.(\Pi x : A\ .(\Phi\ x) \to C) \to C$$

With environment $A : \mathit{Spec},\ \Phi : A \to \mathit{Prop}$

Existential quantifier

$$\Sigma_A(\Phi) \equiv \Pi C : \mathit{Spec}.(\Pi x : A\ .(\Phi\ x) \to C) \to C$$

Existential quantifier

$$\Sigma_A(\Phi) \equiv \Pi C : Spec.(\Pi x : A\ .(\Phi\ x) \to C) \to C$$

$$\Sigma_A(\Phi) \to A \equiv (\Pi C : Spec.(\Pi x : A\ .(\Phi\ x) \to C) \to C) \to A$$

Existential quantifier

$$\Sigma_A(\Phi) \equiv \Pi C : Spec.(\Pi x : A\ .(\Phi\ x) \to C) \to C$$
$$\Sigma_A(\Phi) \to A \equiv (\Pi C : Spec.(\Pi x : A\ .(\Phi\ x) \to C) \to C) \to A$$
$$t \equiv \lambda f : \Sigma_A(\Phi).f\ A\ (\lambda a : A.\lambda p : (\Phi\ x).a)$$

Existential quantifier

$$\Sigma_A(\Phi) \equiv \Pi C : Spec.(\Pi x : A\ .(\Phi\ x) \to C) \to C$$
$$\Sigma_A(\Phi) \to A \equiv (\Pi C : Spec.(\Pi x : A\ .(\Phi\ x) \to C) \to C) \to A$$
$$t \equiv \lambda f : \Sigma_A(\Phi).f\ A\ (\lambda a : A.\lambda p : (\Phi\ x).a)$$

$$\vdash \Sigma_A(\Phi) \to A : \text{Spec}$$
$$\vdash t : \Sigma_A(\Phi) \to A$$

$\mathcal{E}$ $\qquad$ $\mathcal{R}$

$\vdash_{\lambda\omega} \mathcal{E}(\Sigma_A(\Phi) \to A) : \text{Data}$
$\vdash_{\lambda\omega} \mathcal{E}(t) : \mathcal{E}(\Sigma_A(\Phi) \to A)$

$x : \mathcal{E}(\Sigma_A(\Phi) \to A) \vdash \mathcal{R}(\Sigma_A(\Phi) \to A, x) : \text{Prop}$
$\vdash u : \mathcal{R}(\Sigma_A(\Phi) \to A, \mathcal{E}(t))$

Absurdity

$$\bot = \Pi C : \mathsf{Prop}.C, \quad \text{non-informative}$$

$$\mathsf{except} = \Pi C : \mathsf{Spec}.\bot \to C, \quad \text{informative}$$

$$\begin{aligned}
\mathcal{E}(\mathsf{except}) &= \mathcal{E}(\Pi C : \mathsf{Spec}.(\Pi C : \mathsf{Prop}.C) \to C) \\
&= \Pi \bar{C} : \mathcal{E}(\mathsf{Spec}).\mathcal{E}((\Pi C : \mathsf{Prop}.C) \to C) \\
&= \Pi \bar{C} : \mathsf{Data}.\mathcal{E}(C) \\
&= \Pi \bar{C} : \mathsf{Data}.\bar{C}
\end{aligned}$$

Absurdity

$$\bot \equiv \Pi C : \mathsf{Prop}.C, \quad \mathcal{E}(\bot) = \mathsf{undefined}, \quad \mathcal{R}(\bot) = \bot$$

$$\mathsf{except} = \Pi C : \mathsf{Spec}.\bot \to C$$

$$\mathcal{E}(\mathsf{except}) = \Pi \bar{C} : \mathsf{Data}.\bar{C} = \bot$$

$$\begin{aligned}
\mathcal{R}(\mathsf{except}, r) &= \mathcal{R}(\Pi C : \mathsf{Spec}.\bot \to C, r) \\
&= \Pi \bar{C} : \mathsf{Data}.\Pi C : \mathcal{R}(\mathsf{Spec}, \bar{C}).\mathcal{R}(\bot \to C, (r\ \bar{C})) \\
&= \Pi \bar{C} : \mathsf{Data}.\Pi C : \bar{C} \to \mathsf{Prop}.\mathcal{R}(\bot \to C, (r\ \bar{C})) \\
&= \Pi \bar{C} : \mathsf{Data}.\Pi C : \bar{C} \to \mathsf{Prop}.\mathcal{R}(\bot) \to \mathcal{R}(C, (r\ \bar{C})) \\
&= \Pi \bar{C} : \mathsf{Data}.\Pi C : \bar{C} \to \mathsf{Prop}.\bot \to \mathcal{R}(C)\ (r\ \bar{C}) \\
&= \Pi \bar{C} : \mathsf{Data}.\Pi C : \bar{C} \to \mathsf{Prop}.\bot \to C\ (r\ \bar{C})
\end{aligned}$$

Absurdity

$$\bot \equiv \Pi C : \mathsf{Prop}.C, \quad \mathcal{E}(\bot) = \mathsf{undefined}, \quad \mathcal{R}(\bot) = \bot$$

$$\mathsf{except} = \Pi C : \mathsf{Spec}.\bot \to C$$

$$\mathcal{E}(\mathsf{except}) = \Pi \bar{C} : \mathsf{Data}.\bar{C} = \bot$$

$$\mathcal{R}(\mathsf{except}, r) = \Pi \bar{C} : \mathsf{Data}.\Pi C : \bar{C} \to \mathsf{Prop}.\bot \to C\ (r\ \bar{C})$$

Absurdity

$$\bot \equiv \Pi C : \mathsf{Prop}.C, \quad \mathcal{E}(\bot) = \mathsf{undefined}, \quad \mathcal{R}(\bot) = \bot$$

$$\mathsf{except} = \Pi C : \mathsf{Spec}.\bot \to C$$

$$\mathcal{E}(\mathsf{except}) = \Pi \bar{C} : \mathsf{Data}.\bar{C} = \bot$$

$$\mathcal{R}(\mathsf{except}, r) = \Pi \bar{C} : \mathsf{Data}.\Pi C : \bar{C} \to \mathsf{Prop}.\bot \to C\ (r\ \bar{C})$$

- $\bot$ has no closed inhabitant

Absurdity

$$\bot \equiv \Pi C : \mathsf{Prop}.C, \quad \mathcal{E}(\bot) = \mathsf{undefined}, \quad \mathcal{R}(\bot) = \bot$$

$$\mathsf{except} = \Pi C : \mathsf{Spec}.\bot \to C$$

$$\mathcal{E}(\mathsf{except}) = \Pi \bar{C} : \mathsf{Data}.\bar{C} = \bot$$

$$\mathcal{R}(\mathsf{except}, r) = \Pi \bar{C} : \mathsf{Data}.\Pi C : \bar{C} \to \mathsf{Prop}.\bot \to C\ (r\ \bar{C})$$

- $\bot$ has no closed inhabitant
- $\mathcal{R}(\mathit{except}, \mathcal{E}(\mathsf{except})$ also doesn't.

Absurdity

$$\bot \equiv \Pi C : \mathsf{Prop}.C, \quad \mathcal{E}(\bot) = \mathsf{undefined}, \quad \mathcal{R}(\bot) = \bot$$

$$\mathsf{except} = \Pi C : \mathsf{Spec}.\bot \to C$$

$$\mathcal{E}(\mathsf{except}) = \Pi \bar{C} : \mathsf{Data}.\bar{C} = \bot$$

$$\mathcal{R}(\mathsf{except}, r) = \Pi \bar{C} : \mathsf{Data}.\Pi C : \bar{C} \to \mathsf{Prop}.\bot \to C\ (r\ \bar{C})$$

- $\bot$ has no closed inhabitant
- $\mathcal{R}(\mathit{except}, \mathcal{E}(\mathsf{except})$ also doesn't.
- so $\mathcal{E}(\mathsf{except}) = \bot$ is not a proper realizer of except

Absurdity

$$\bot \equiv \Pi C : \mathsf{Prop}.C, \quad \mathcal{E}(\bot) = \mathsf{undefined}, \quad \mathcal{R}(\bot) = \bot$$

$$\mathsf{except} = \Pi C : \mathsf{Spec}.\bot \to C$$

$$\mathcal{E}(\mathsf{except}) = \Pi \bar{C} : \mathsf{Data}.\bar{C} = \bot$$

$$\mathcal{R}(\mathsf{except}, r) = \Pi \bar{C} : \mathsf{Data}.\Pi C : \bar{C} \to \mathsf{Prop}.\bot \to C\ (r\ \bar{C})$$

- $\bot$ has no closed inhabitant
- $\mathcal{R}(\mathit{except}, \mathcal{E}(\mathsf{except})$ also doesn't.
- so $\mathcal{E}(\mathsf{except}) = \bot$ is not a proper realizer of except
- we can add except as an axiom

Consistency

- In the previous examples we have used extraction to turn proofs into programs.
- The following example will show that we can create types and axioms that cannot be proven in λC, but can be realized.
- This means that the proof system stays **consistent** i.e. by introducing the new stuff, we are not introducing contradictions.

Subset type

- We want to create a subset type $\{A|P\}$ with $A : \mathit{Data}$ and $P : A \to \mathrm{Prop}$.
- We want the type $\{A|P\}$ to be inhabited by all elements of type A for which the predicate P holds.
- We will discuss the following properties
  1. $\{A|P\} : \mathrm{Spec}$
  2. $\mathit{subintro} : \Pi x : A.(P\ x) \to \{A|P\}$

Subset type

- We want that

$$\{A|P\} : \text{Spec}$$

- This is not provable in CoC because there is no way to build a subset type.
- But we can **define realizability** i.e. define what $R(\{A|P\}, r)$ means.
- That is, we can define the type of r and define a function that says "r is a valid realizer".
- We say that the type of r is A.
- And we define the function "r is a valid realizer" as P.

Subset type

- We want that

$$\mathit{subintro} \ : \Pi x : A.\Pi y : P\ x.\{A|P\}$$

Subset type

- We want that

$$subintro \ : \Pi x : A.\Pi y : P\ x.\{A|P\}$$

- Not provable in CoC: there is no way to prove $\{A|P\} : \mathrm{Spec}$

Subset type

- We want that

$$\mathit{subintro} \ : \Pi x : A.\Pi y : P\ x.\{A|P\}$$

- Not provable in CoC: there is no way to prove $\{A|P\} : \mathrm{Spec}$
- But is realizable:

$$\begin{aligned} R(\mathit{subintro}, r) =& R(\Pi x : A.\Pi y : P\ x.\{A|P\}, r) \\ =& \Pi x : A.R(\Pi y : P\ x.\{A|P\}, r\ x) \\ =& \Pi x : A.\Pi y : R(P\ x).R(\{A|P\}, r\ x) \\ =& \Pi x : A.\Pi y : R(P\ x).R(P(r\ x)) \end{aligned}$$

Subset type

- We want that

$$subintro \ : \Pi x : A.\Pi y : P\ x.\{A|P\}$$

- Not provable in CoC: there is no way to prove $\{A|P\} : \mathrm{Spec}$
- But is realizable:

$$\begin{aligned} R(subintro, r) =& R(\Pi x : A.\Pi y : P\ x.\{A|P\}, r) \\ =& \Pi x : A.R(\Pi y : P\ x.\{A|P\}, r\ x) \\ =& \Pi x : A.\Pi y : R(P\ x).R(\{A|P\}, r\ x) \\ =& \Pi x : A.\Pi y : R(P\ x).R(P(r\ x)) \end{aligned}$$

- Realizer $r = \lambda x : A.x$ works.