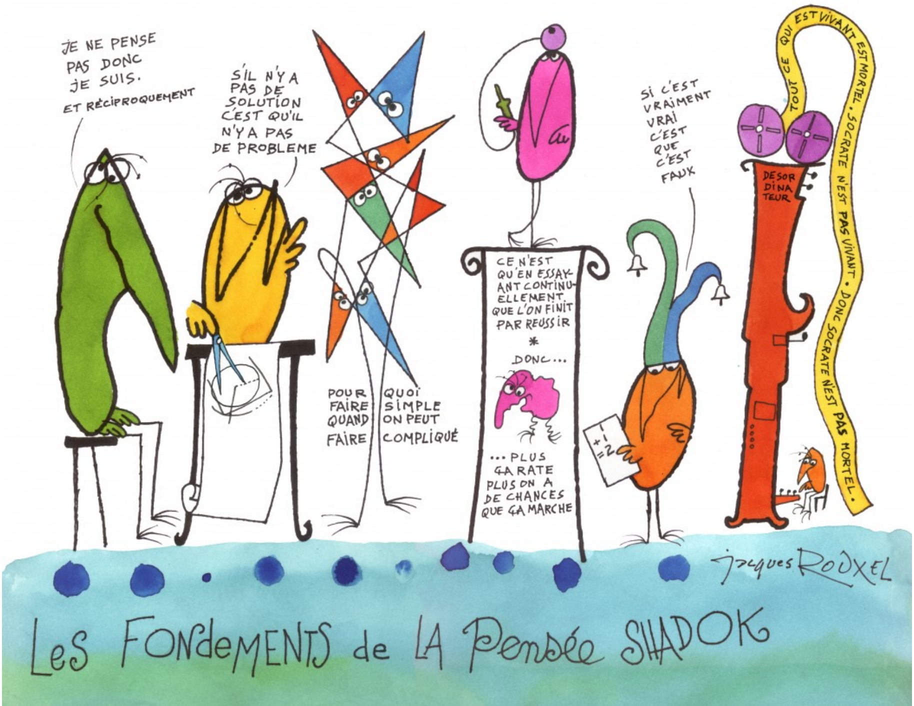
Failure is Not an Option

The Curry-Howard-Shadok correspondence

Pierre-Marie Pédrot
joint work with **Nicolas Tabareau**

Max Planck Institute for Software Systems

Séminaire PPS


JE NE PENSE PAS DONC JE SUIS. ET RÉCIPROQUEMENT
S'IL N'Y A PAS DE SOLUTION C'EST QU'IL N'Y A PAS DE PROBLEME
POUR FAIRE SIMPLE QUAND ON PEUT FAIRE COMPLIQUÉ
QUOI
CE N'EST QU'EN ESSAYANT CONTINUELLEMENT QUE L'ON FINIT PAR REUSSIR
*
DONC...
...PLUS ÇA RATE PLUS ON A DE CHANCES QUE ÇA MARCHE
SI C'EST VRAIMENT VRAI C'EST QUE C'EST FAUX
DESORDINATEUR
TOUT CE QUI EST VIVANT EST MORTEL. SOCRATE N'EST PAS VIVANT. DONC SOCRATE N'EST PAS MORTEL.
Jacques ROUXEL
Les FONdeMENTS de LA Pensée SHADOK

It's time to CIC ass and chew bubble-gum

CIC, the Calculus of Inductive Constructions.

It's time to CIC ass and chew bubble-gum

CIC, the Calculus of Inductive Constructions.

CIC, a very fancy **intuitionistic** **logical system**.

- Not just higher-order logic, not just first-order logic
- First class notion of computation and crazy inductive types

It's time to CIC ass and chew bubble-gum

CIC, the Calculus of Inductive Constructions.

CIC, a very fancy **intuitionistic** **logical system**.

- Not just higher-order logic, not just first-order logic
- First class notion of computation and crazy inductive types

CIC, a very powerful **functional** **programming language**.

- Finest types to describe your programs
- No clear phase separation between runtime and compile time

It’s time to CIC ass and chew bubble-gum

CIC, the Calculus of Inductive Constructions.

CIC, a very fancy **intuitionistic** **logical system**.

- Not just higher-order logic, not just first-order logic
- First class notion of computation and crazy inductive types

CIC, a very powerful **functional** **programming language**.

- Finest types to describe your programs
- No clear phase separation between runtime and compile time

The Pinnacle of the Curry-Howard correspondence

Un Coq qui fait de l'effet

My research has been focussed on the extension of CIC with **side-effects**.

My research has been focussed on the extension of CIC with **side-effects**.

To `Program` More!

- Obviously you want effects to program
- E.g. state, exceptions, non-termination, continuations...

My research has been focussed on the extension of CIC with **side-effects**.

To `Program` More!

- Obviously you want effects to program
- E.g. state, exceptions, non-termination, continuations...

To *Prove* More!

- A well-known fact here at PPS
- Curry-Howard $\vdash$ side-effects $\Leftrightarrow$ new axioms
- Archetypical example: `callcc` and classical logic (Griffin, Krivine)

Summary of the Previous Episodes

We already gave two instances of effectful variants of CIC.

Summary of the Previous Episodes

We already gave two instances of effectful variants of CIC.

Forcing (LICS 2016)

- Bread and butter categorical model factory
- *« Forcing: retour de l'être aimé – permis de conduire – désenvoûtement. »*
- Computationally: a glorified monotonous reader monad

Summary of the Previous Episodes

We already gave two instances of effectful variants of CIC.

Forcing (LICS 2016)

- Bread and butter categorical model factory
- *« Forcing: retour de l'être aimé – permis de conduire – désenvoûtement. »*
- Computationally: a glorified monotonous reader monad

Weaning (LICS 2017)

- A generic construction adding effects
- Handles a rather wide class of monads
- Somehow dual to forcing

You Can't Have Your Cake and Eat It

Effects make reduction strategies relevant.

You Can't Have Your Cake and Eat It

Effects make reduction strategies relevant.

Call-by-value	Call-by-name
	
☹ Weaker conversion rule	☺ Full conversion rule
☺ Full dependent elimination	☹ Weaker dependent elimination
☺ Good old ML semantics	☹ Strange PL realm

Last Propaganda Slide: A Flurry of Buzzwords

Recall that dependent elimination for booleans amounts to

$$\frac{\Gamma \vdash M : \mathbb{B} \qquad \Gamma \vdash N_1 : P\{\texttt{true}\} \qquad \Gamma \vdash N_2 : P\{\texttt{false}\}}{\Gamma \vdash \texttt{if } M \texttt{ then } N_1 \texttt{ else } N_2 : P\{M\}}$$

Last Propaganda Slide: A Flurry of Buzzwords

Recall that dependent elimination for booleans amounts to

$$\frac{\Gamma \vdash M : \mathbb{B} \qquad \Gamma \vdash N_1 : P\{\texttt{true}\} \qquad \Gamma \vdash N_2 : P\{\texttt{false}\}}{\Gamma \vdash \texttt{if } M \texttt{ then } N_1 \texttt{ else } N_2 : P\{M\}}$$

We proposed a generic restriction for effectful CBN dependent elimination.

P **must be linear** ($\cong$ CBV / algebra hom.)

- Generalizes Krivine's storage operators
- If you weren't at my Geocal-LAC talk, *tant pis pour vous*
- Towards a Linear Dependent {Big Data, Machine Learning, IoT}

Shameless Propaganda

Part I

An extension of CIC rooted in Shadok wisdom.

"The more it fails, the more likely it will eventually succeed."

Part I

An extension of CIC rooted in Shadok wisdom.

"THE MORE IT FAILS, THE MORE LIKELY IT WILL EVENTUALLY SUCCEED."

- ☺ Add a failure mechanism to CIC
- ☺ Fully computational exceptions
- ☺ Features full conversion
- ☺ Features full dependent elimination

Part I

An extension of CIC rooted in Shadok wisdom.

"The more it fails, the more likely it will eventually succeed."

- ☺ Add a failure mechanism to CIC
- ☺ Fully computational exceptions
- ☺ Features full conversion
- ☺ Features full dependent elimination
- 😕 Didn't I just say this was not possible???

Part I

An extension of CIC rooted in Shadok wisdom.

“THE MORE IT FAILS, THE MORE LIKELY IT WILL EVENTUALLY SUCCEED.”

- ☺ Add a failure mechanism to CIC
- ☺ Fully computational exceptions
- ☺ Features full conversion
- ☺ Features full dependent elimination
- ☹ Didn't I just say this was not possible???

The Exceptional Type Theory: Overview

The exceptional type theory extends vanilla CIC with

$$\begin{array}{rcl}\mathbf{E} & : & \square \\ \texttt{raise} & : & \Pi A : \square .\, \mathbf{E} \to A\end{array}$$

The Exceptional Type Theory: Overview

The exceptional type theory extends vanilla CIC with

$$\begin{array}{rcl} \mathbf{E} & : & \square \\ \mathtt{raise} & : & \Pi A : \square.\, \mathbf{E} \rightarrow A \end{array}$$

As hinted before, we need to be call-by-name to feature full conversion.

$$\begin{array}{lcl} \mathtt{raise}\ (\Pi x : A.\, B)\ e & \equiv & \lambda x : A.\, \mathtt{raise}\ B\ e \\ \mathtt{match}\ (\mathtt{raise}\ \mathcal{I}\ e)\ \mathtt{ret}\ P\ \mathtt{with}\ \vec{p} & \equiv & \mathtt{raise}\ (P\ (\mathtt{raise}\ \mathcal{I}\ e))\ e \end{array}$$

where $P : \mathcal{I} \rightarrow \square$.

The Exceptional Type Theory: Overview

The exceptional type theory extends vanilla CIC with

$$\begin{array}{lcl} \mathbf{E} & : & \square \\ \mathtt{raise} & : & \Pi A : \square.\, \mathbf{E} \to A \end{array}$$

As hinted before, we need to be call-by-name to feature full conversion.

$$\begin{array}{lcl} \mathtt{raise}\ (\Pi x : A.\, B)\ e & \equiv & \lambda x : A.\, \mathtt{raise}\ B\ e \\ \mathtt{match}\ (\mathtt{raise}\ \mathcal{I}\ e)\ \mathtt{ret}\ P\ \mathtt{with}\ \vec{p} & \equiv & \mathtt{raise}\ (P\ (\mathtt{raise}\ \mathcal{I}\ e))\ e \end{array}$$

where $P : \mathcal{I} \to \square$.

Remark that in call-by-name, if $M : A \to B$, in general

$$M\ (\mathtt{raise}\ A\ e) \quad \not\equiv \quad \mathtt{raise}\ B\ e$$

for otherwise we would not have $(\lambda x : A.\, M)\ N \equiv M\{x := N\}$.

Catch Me If You Can

Remember that on functions:

$$\texttt{raise}\ (\Pi x : A.\, B)\ e \quad \equiv \quad \lambda x : A.\, \texttt{raise}\ B\ e$$

It means catching exceptions is limited to positive datatypes!

Catch Me If You Can

Remember that on functions:

$$\texttt{raise}\ (\Pi x : A.\, B)\ e \quad \equiv \quad \lambda x : A.\, \texttt{raise}\ B\ e$$

It means catching exceptions is limited to positive datatypes!

For inductive types, this is a **generalized induction principle**.

$$\begin{array}{ll} \texttt{catch}_{\mathbb{B}} : & \Pi P : \mathbb{B} \to \square. \\ & \quad P\ \texttt{true} \to \\ & \quad P\ \texttt{false} \to \\ & \quad (\Pi e : \mathbf{E}.\, P\ (\texttt{raise}\ \mathbb{B}\ e)) \to \\ & \Pi b : \mathbb{B}.\, P\ b \end{array} \qquad \begin{array}{ll} \mathbb{B}_{\texttt{rect}} : & \Pi P : \mathbb{B} \to \square. \\ & \quad P\ \texttt{true} \to \\ & \quad P\ \texttt{false} \to \\ & \\ & \Pi b : \mathbb{B}.\, P\ b \end{array}$$

where

$$\begin{array}{lcl} \texttt{catch}_{\mathbb{B}}\ P\ p_t\ p_f\ p_e\ \texttt{true} & \equiv & p_t \\ \texttt{catch}_{\mathbb{B}}\ P\ p_t\ p_f\ p_e\ \texttt{false} & \equiv & p_f \\ \texttt{catch}_{\mathbb{B}}\ P\ p_t\ p_f\ p_e\ (\texttt{raise}\ \mathbb{B}\ e) & \equiv & p_e\ e \end{array}$$

It's not just randomly coming up with syntax though.

Mot d'Ordre: A Model

It's not just randomly coming up with syntax though.

- We want a justification for what we are doing
- What about normalization? Subject reduction? Other nice properties?

Mot d'Ordre: A Model

It's not just randomly coming up with syntax though.

- We want a justification for what we are doing
- What about normalization? Subject reduction? Other nice properties?
- … that's called a model.

We want a **model of** the exceptional type theory!

Kardashian Functors, Anyone?

Semantics of CIC has a fame of being horribly complex.

Kardashian Functors, Anyone?

Semantics of CIC has a fame of being horribly complex.

I won't lie: **it is**. But part of this fame is nonetheless due to its models.

Kardashian Functors, Anyone?

Semantics of CIC has a fame of being horribly complex.

I won't lie: **it is**. But part of this fame is nonetheless due to its models.

Set-theoretical models: because Sets are a (crappy) type theory.

- **Pro:** Sets!
- **Con:** Sets!

Kardashian Functors, Anyone?

Semantics of CIC has a fame of being horribly complex.

I won't lie: **it is**. But part of this fame is nonetheless due to its models.

Set-theoretical models: because Sets are a (crappy) type theory.

- **Pro:** Sets!
- **Con:** Sets!

Realizability models: construct programs that respect properties.

- **Pro:** Computational, computer-science friendly.
- **Con:** Not foundational (requires an alien meta-theory), not decidable.

Kardashian Functors, Anyone?

Semantics of CIC has a fame of being horribly complex.

I won't lie: **it is**. But part of this fame is nonetheless due to its models.

Set-theoretical models: because Sets are a (crappy) type theory.

- **Pro:** Sets!
- **Con:** Sets!

Realizability models: construct programs that respect properties.

- **Pro:** Computational, computer-science friendly.
- **Con:** Not foundational (requires an alien meta-theory), not decidable.

Categorical models: abstract description of type theory.

- **Pro:** Abstract, subsumes the two former ones.
- **Con:** Realizability + very low level, gazillion variants, intrisically typed, static.

Instead, let's look at what Curry-Howard provides in simpler settings.

Logical Interpretations $\Leftrightarrow$ Program Translations

Curry-Howard Orthodoxy

Instead, let's look at what Curry-Howard provides in simpler settings.

Logical Interpretations $\Leftrightarrow$ Program Translations

On the **programming** side, implement effects using e.g. the *monadic* style.

- A type transformer T, two combinators, a few equations
- Interpret mechanically effectful programs (e.g. in Haskell)

Curry-Howard Orthodoxy

Instead, let's look at what Curry-Howard provides in simpler settings.

Logical Interpretations $\Leftrightarrow$ Program Translations

On the **programming** side, implement effects using e.g. the *monadic* style.

- A type transformer T, two combinators, a few equations
- Interpret mechanically effectful programs (e.g. in Haskell)

On the **logic** side, extend expressivity through proof translation.

- Double-negation $\Rightarrow$ classical logic (`callcc`)
- Friedman's trick $\Rightarrow$ Markov's rule (exceptions)
- Forcing $\Rightarrow$ $\neg\mathrm{CH}$ (global monotonous cell)

Syntactic Models

Let us do the same thing with CIC: build **syntactic models**.

Let us do the same thing with CIC: build **syntactic models**.

Step 0: Fix a theory $\mathcal{T} := \mathrm{CIC}$.

Syntactic Models

Let us do the same thing with CIC: build **syntactic models**.

Step 0: Fix a theory $\mathcal{T} := \mathrm{CIC}$.

Step 1: Define $[\cdot]$ on the syntax of $\mathcal{T}$ and derive $[\![\cdot]\!]$ from it s.t.

$$\vdash_{\mathcal{T}} M : A \qquad \text{implies} \qquad \vdash_{\mathrm{CIC}} [M] : [\![A]\!]$$

Syntactic Models

Let us do the same thing with CIC: build **syntactic models**.

Step 0: Fix a theory $\mathcal{T} := \mathrm{CIC}$.

Step 1: Define $[\cdot]$ on the syntax of $\mathcal{T}$ and derive $[\![\cdot]\!]$ from it s.t.

$$\vdash_{\mathcal{T}} M : A \qquad \text{implies} \qquad \vdash_{\mathrm{CIC}} [M] : [\![A]\!]$$

Step 2: Flip views and actually pose

$$\vdash_{\mathcal{T}} M : A \qquad \triangleq \qquad \vdash_{\mathrm{CIC}} [M] : [\![A]\!]$$

Syntactic Models

Let us do the same thing with CIC: build **syntactic models**.

Step 0: Fix a theory $\mathcal{T} := \mathrm{CIC}$.

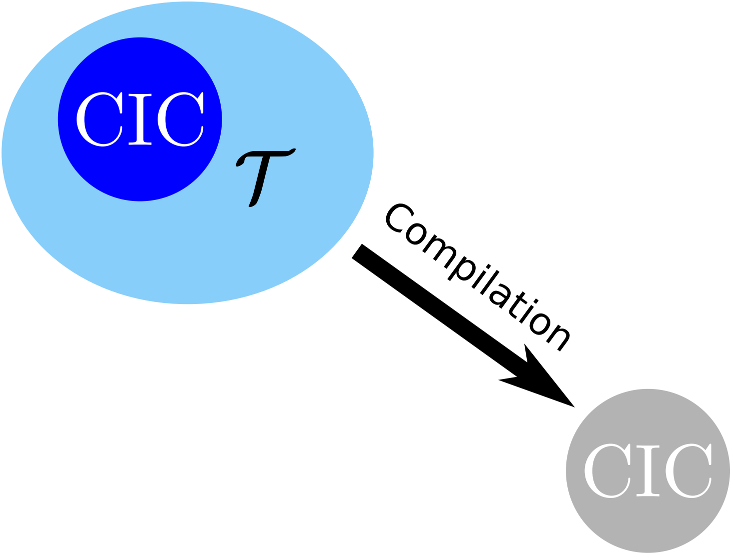
Step 1: Define $[\cdot]$ on the syntax of $\mathcal{T}$ and derive $[\![\cdot]\!]$ from it s.t.

$$\vdash_{\mathcal{T}} M : A \qquad \text{implies} \qquad \vdash_{\mathrm{CIC}} [M] : [\![A]\!]$$

Step 2: Flip views and actually pose

$$\vdash_{\mathcal{T}} M : A \qquad \triangleq \qquad \vdash_{\mathrm{CIC}} [M] : [\![A]\!]$$

Step 3: Expand $\mathcal{T}$ by going down to the CIC assembly language, implementing new terms given by the $[\cdot]$ translation.



« CIC, the LLVM of Type Theory »

The Exceptional Implementation

Let's implement the exceptional type theory into CIC!

The Exceptional Implementation

Let's implement the exceptional type theory into CIC!

- Source is a CBN theory, so usual monadic encoding won't work.
- We use a variant of our previous weaning translation.
- All typing and computations rules mentioned before hold for free.

The Exceptional Implementation

Let's implement the exceptional type theory into CIC!

- Source is a CBN theory, so usual monadic encoding won't work.
- We use a variant of our previous weaning translation.
- All typing and computations rules mentioned before hold for free.

Let's call the exceptional type theory $\mathcal{T}_{\mathbb{E}}$ to disambiguate it from CIC.

The Exceptional Implementation

Let's implement the exceptional type theory into CIC!

- Source is a CBN theory, so usual monadic encoding won't work.
- We use a variant of our previous weaning translation.
- All typing and computations rules mentioned before hold for free.

Let's call the exceptional type theory $\mathcal{T}_{\mathbb{E}}$ to disambiguate it from CIC.

Only parameter of the translation: a fixed type of exceptions in the target.

$$\vdash_{\mathrm{CIC}} \mathbb{E} : \square$$

The Exceptional Implementation, Negative case

Intuition: $\vdash_{\mathcal{T}_{\mathbb{E}}} A : \square \quad \rightsquigarrow \quad \vdash_{\mathrm{CIC}} [A] : \Sigma A : \square.\, \mathbb{E} \to A.$

Every exceptional type comes with its own implementation of failure!

The Exceptional Implementation, Negative case

Intuition: $\vdash_{\mathcal{T}_{\mathbb{E}}} A : \square \quad \rightsquigarrow \quad \vdash_{\mathrm{CIC}} [A] : \Sigma A : \square.\, \mathbb{E} \to A.$

Every exceptional type comes with its own implementation of failure!

$$[\![A]\!] : \square := \pi_1\ [A] \qquad \text{and} \qquad [A]_\varnothing : \mathbb{E} \to [\![A]\!] := \pi_2\ [A]$$

$$\begin{array}{lcl}
[\![\Pi x : A.\, B]\!] & \equiv & \Pi x : [\![A]\!].\, [\![B]\!] \\
[\Pi x : A.\, B]_\varnothing\ e & \equiv & \lambda x : [\![A]\!].\, [B]_\varnothing\ e \\
[x] & \equiv & x \\
[M\ N] & \equiv & [M]\ [N] \\
[\lambda x : A.\, M] & \equiv & \lambda x : [\![A]\!].\, [M]
\end{array}$$

The Exceptional Implementation, Negative case

Intuition: $\vdash_{\mathcal{T}_{\mathbb{E}}} A : \square \quad \rightsquigarrow \quad \vdash_{\mathrm{CIC}} [A] : \Sigma A : \square.\, \mathbb{E} \to A.$

Every exceptional type comes with its own implementation of failure!

$$[\![A]\!] : \square := \pi_1\ [A] \qquad \text{and} \qquad [A]_\varnothing : \mathbb{E} \to [\![A]\!] := \pi_2\ [A]$$

$$\begin{array}{lcl}
[\![\Pi x : A.\, B]\!] & \equiv & \Pi x : [\![A]\!].\, [\![B]\!] \\
[\Pi x : A.\, B]_\varnothing\ e & \equiv & \lambda x : [\![A]\!].\, [B]_\varnothing\ e \\
[x] & \equiv & x \\
[M\ N] & \equiv & [M]\ [N] \\
[\lambda x : A.\, M] & \equiv & \lambda x : [\![A]\!].\, [M]
\end{array}$$

If $\Gamma \vdash_{\mathrm{CIC}} M : A$ then $[\![\Gamma]\!] \vdash_{\mathrm{CIC}} [M] : [\![A]\!]$.

The Exceptional Implementation, Failure

It is straightforward to implement the failure operation.

$$\begin{array}{lcl} \mathbf{E} & : & \square \\ \mathtt{raise} & : & \Pi A : \square.\, \mathbf{E} \rightarrow A \end{array}$$

The Exceptional Implementation, Failure

It is straightforward to implement the failure operation.

$$\begin{array}{rcl} \mathbf{E} & : & \square \\ \texttt{raise} & : & \Pi A : \square.\, \mathbf{E} \to A \end{array}$$

$$\begin{array}{lcl} [\mathbf{E}] & : & \Sigma A : \square.\, \mathbb{E} \to A \\ [\mathbf{E}] & := & (\mathbb{E}, \lambda e : \mathbb{E}.\, e) \end{array}$$

$$\begin{array}{lcl} [\texttt{raise}] & : & \Pi A_0 : (\Sigma A : \square.\, \mathbb{E} \to A).\, \mathbb{E} \to \pi_1\ A_0 \\ [\texttt{raise}] & := & \pi_2 \end{array}$$

The Exceptional Implementation, Failure

It is straightforward to implement the failure operation.

$$\begin{array}{rcl} \mathbf{E} & : & \square \\ \mathtt{raise} & : & \Pi A : \square.\, \mathbf{E} \to A \end{array}$$

$$\begin{array}{rcl} [\mathbf{E}] & : & \Sigma A : \square.\, \mathbb{E} \to A \\ [\mathbf{E}] & := & (\mathbb{E}, \lambda e : \mathbb{E}.\, e) \\ & & \\ [\mathtt{raise}] & : & \Pi A_0 : (\Sigma A : \square.\, \mathbb{E} \to A).\, \mathbb{E} \to \pi_1\ A_0 \\ [\mathtt{raise}] & := & \pi_2 \end{array}$$

Computational rules trivially hold!

$$\begin{array}{ccc} [\mathtt{raise}\ (\Pi x : A.\, B)\ e] & \equiv & [\lambda x : A.\, \mathtt{raise}\ B\ e] \\ ||| & & ||| \\ \pi_2\ ((\Pi x : [\![A]\!].\, [\![B]\!]), (\lambda (e : \mathbb{E})\, (x : [\![A]\!]).\, \pi_2\ [\![B]\!]\ e))\ [e] & \equiv & \lambda x : [\![A]\!].\, \pi_2\ [B]\ [e] \end{array}$$

The Exceptional Implementation, Positive case

The really interesting case is the inductive part of CIC.

How to implement $[\mathbb{B}]_{\varnothing} : \mathbb{E} \to [\![\mathbb{B}]\!]$?

The Exceptional Implementation, Positive case

The really interesting case is the inductive part of CIC.

How to implement $[\mathbb{B}]_\varnothing : \mathbb{E} \to [\![\mathbb{B}]\!]$?

Could pose $[\![\mathbb{B}]\!] := \mathbb{B}$ and take an arbitrary boolean for $[\mathbb{B}]_\varnothing$...

... but that would not play well with computation, e.g. `catch`.

The Exceptional Implementation, Positive case

The really interesting case is the inductive part of CIC.

How to implement $[\mathbb{B}]_\varnothing : \mathbb{E} \to [\![\mathbb{B}]\!]$?

Could pose $[\![\mathbb{B}]\!] := \mathbb{B}$ and take an arbitrary boolean for $[\mathbb{B}]_\varnothing$...

... but that would not play well with computation, e.g. `catch`.

Worse, what about $[\bot]_\varnothing : \mathbb{E} \to [\![\bot]\!]$?

The Exceptional Implementation, Positive case

Very elegant solution: add a default case to every inductive type!

$$\texttt{Inductive}\ [\![\mathbb{B}]\!] := [\texttt{true}] : [\![\mathbb{B}]\!] \mid [\texttt{false}] : [\![\mathbb{B}]\!] \mid \mathbb{B}_\varnothing : \mathbb{E} \to [\![\mathbb{B}]\!]$$

The Exceptional Implementation, Positive case

Very elegant solution: add a default case to every inductive type!

$$\texttt{Inductive}\ [\![\mathbb{B}]\!] := [\texttt{true}] : [\![\mathbb{B}]\!] \mid [\texttt{false}] : [\![\mathbb{B}]\!] \mid \mathbb{B}_\varnothing : \mathbb{E} \to [\![\mathbb{B}]\!]$$

Pattern-matching is translated pointwise, except for the new case.

$$[\![\Pi P : \mathbb{B} \to \square.\, P\ \texttt{true} \to P\ \texttt{false} \to \Pi b : \mathbb{B}.\, P\ b]\!]$$

$$\equiv \quad \Pi P : [\![\mathbb{B}]\!] \to [\![\square]\!].\, P\ [\texttt{true}] \to P\ [\texttt{false}] \to \Pi b : [\![\mathbb{B}]\!].\, P\ b$$

- If b is $[\texttt{true}]$, use first hypothesis
- If b is $[\texttt{false}]$, use second hypothesis
- If b is an error $\mathbb{B}_\varnothing\ e$, **reraise** e using $[P\ b]_\varnothing\ e$

Shadok Logic Strikes Back

Theorem

The exceptional translation interprets all of CIC.

Shadok Logic Strikes Back

Theorem

The exceptional translation interprets all of CIC.

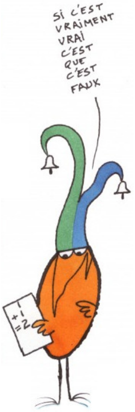
- ☺ A type theory with effects!
- ☺ Compiled away to CIC!
- ☺ Features full conversion
- ☺ Features full dependent elimination

Shadok Logic Strikes Back

Theorem

The exceptional translation interprets all of CIC.

- ☺ A type theory with effects!
- ☺ Compiled away to CIC!
- ☺ Features full conversion
- ☺ Features full dependent elimination



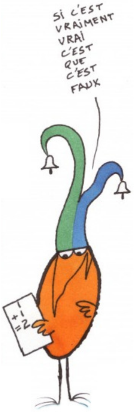
Shadok Logic Strikes Back

Theorem

The exceptional translation interprets all of CIC.

- ☺ A type theory with effects!
- ☺ Compiled away to CIC!
- ☺ Features full conversion
- ☺ Features full dependent elimination
- ☹ Ah, yeah, and also, the theory is inconsistent.

It suffices to raise an exception to inhabit any type.



Consistency: A Social Construct

An Impure Dependently-typed Programming Language

Do you whine about the fact that OCaml is logically inconsistent?

Consistency: A Social Construct

An Impure Dependently-typed Programming Language

Do you whine about the fact that OCaml is logically inconsistent?

Theorem (Exceptional Canonicity a.k.a. Progress a.k.a. Meaningless explanations)

If $\vdash_{\mathcal{T}_{\mathbb{E}}} M : \bot$, *then* $M \equiv \texttt{raise}\ \bot\ e$ *for some* $e : \mathbf{E}$.

Consistency: A Social Construct

An Impure Dependently-typed Programming Language

Do you whine about the fact that OCaml is logically inconsistent?

Theorem (Exceptional Canonicity a.k.a. Progress a.k.a. Meaningless explanations)

If $\vdash_{\mathcal{T}_{\mathbb{E}}} M : \bot$, *then* $M \equiv \texttt{raise} \perp e$ *for some* $e : \mathbf{E}$.

A Safe Target Framework

You can still use the CIC target to prove properties about $\mathcal{T}_{\mathbb{E}}$ programs!

Consistency: A Social Construct

An Impure Dependently-typed Programming Language

Do you whine about the fact that OCaml is logically inconsistent?

Theorem (Exceptional Canonicity a.k.a. Progress a.k.a. Meaningless explanations)

If $\vdash_{\mathcal{T}_{\mathbb{E}}} M : \bot$, *then* $M \equiv \texttt{raise}\ \bot\ e$ *for some* $e : \mathbf{E}$.

A Safe Target Framework

You can still use the CIC target to prove properties about $\mathcal{T}_{\mathbb{E}}$ programs!

Cliffhanger

You can prove that a program does not raise uncaught exceptions.

Consistency: A Social Construct

An Impure Dependently-typed Programming Language

Do you whine about the fact that OCaml is logically inconsistent?

Theorem (Exceptional Canonicity a.k.a. Progress a.k.a. Meaningless explanations)

If $\vdash_{\mathcal{T}_{\mathbb{E}}} M : \bot$, *then* $M \equiv \mathtt{raise}\ \bot\ e$ *for some* $e : \mathbf{E}$.

A Safe Target Framework

You can still use the CIC target to prove properties about $\mathcal{T}_{\mathbb{E}}$ programs!

Cliffhanger

You can prove that a program does not raise uncaught exceptions.

And now for a little ad before the second part of the show!

Informercial — Did You Know?

The exceptional translation is just a principled Friedman's A-translation!

Informercial — Did You Know?

The exceptional translation is just a principled Friedman's A-translation!

As such, it can be used for classical proof extraction.

Informative double-negation

$$[\![\neg\neg A]\!] \cong ([\![A]\!] \to \mathbb{E}) \to \mathbb{E}$$

Informercial — Did You Know?

The exceptional translation is just a principled Friedman's A-translation!

As such, it can be used for classical proof extraction.

Informative double-negation

$$[\![\neg\neg A]\!] \cong ([\![A]\!] \to \mathbb{E}) \to \mathbb{E}$$

First-order purification

If P is a Σ^0_1 type, then $\vdash_{\mathrm{CIC}} [\![P]\!] \leftrightarrow P + \mathbb{E}$.

Informercial — Did You Know?

The exceptional translation is just a principled Friedman's A-translation!

As such, it can be used for classical proof extraction.

Informative double-negation

$$[\![\neg\neg A]\!] \cong ([\![A]\!] \to \mathbb{E}) \to \mathbb{E}$$

First-order purification

If P is a Σ^0_1 type, then $\vdash_{\mathrm{CIC}} [\![P]\!] \leftrightarrow P + \mathbb{E}$.

Friedman's Trick in CIC

If P and Q are Σ^0_1 types, $\vdash_{\mathrm{CIC}} \Pi p : P.\ \neg\neg Q$ implies $\vdash_{\mathrm{CIC}} \Pi p : P.\ Q$.

Part II

If You Joined the Talk Recently

The exceptional type theory is logically inconsistent!

Cliffhanger (cont.)

You can prove that a program does not raise uncaught exceptions.

If You Joined the Talk Recently

The exceptional type theory is logically inconsistent!

Cliffhanger (cont.)

You can prove that a program does not raise uncaught exceptions.

Let's call **valid** a program in $\mathcal{T}_{\mathbb{E}}$ that "does not raise exceptions".

For instance,

- there is no valid proof of $\bot$
- the only valid booleans are `true` and `false`
- a function is valid if it produces a valid result out of a valid argument

If You Joined the Talk Recently

The exceptional type theory is logically inconsistent!

Cliffhanger (cont.)

You can prove that a program does not raise uncaught exceptions.

Let's call **valid** a program in $\mathcal{T}_{\mathbb{E}}$ that "does not raise exceptions".

For instance,

- there is no valid proof of $\bot$
- the only valid booleans are `true` and `false`
- a function is valid if it produces a valid result out of a valid argument

Validity is a type-directed notion!

The Curry-Howard-Shadok Correspondence

Let's locally write $M \Vdash A$ if M is valid at A.

The Curry-Howard-Shadok Correspondence

Let's locally write $M \Vdash A$ if M is valid at A.

$$f \Vdash A \rightarrow B \quad \equiv \quad \forall x : [\![A]\!]. \quad x \Vdash A \rightarrow f\,x \Vdash B$$

The Curry-Howard-Shadok Correspondence

Let's locally write $M \Vdash A$ if M is valid at A.

$$f \Vdash A \to B \quad \equiv \quad \forall x : [\![A]\!].\quad x \Vdash A \to f\,x \Vdash B$$

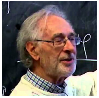
What? That's just **logical relations**.

The Curry-Howard-Shadok Correspondence

Let's locally write $M \Vdash A$ if M is valid at A.

$$f \Vdash A \to B \quad \equiv \quad \forall x : [\![A]\!]. \quad x \Vdash A \to f\,x \Vdash B$$

What? That's just **logical relations**.



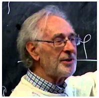
Come on. That's **intuitionistic realizability**.

The Curry-Howard-Shadok Correspondence

Let's locally write $M \Vdash A$ if M is valid at A.

$$f \Vdash A \to B \quad \equiv \quad \forall x : [\![A]\!]. \quad x \Vdash A \to f\,x \Vdash B$$

What? That's just **logical relations**.



Come on. That's **intuitionistic realizability**.

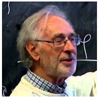
Fools ! That's **parametricity**.

The Curry-Howard-Shadok Correspondence

Let's locally write $M \Vdash A$ if M is valid at A.

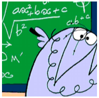
$$f \Vdash A \to B \quad \equiv \quad \forall x : [\![A]\!].\quad x \Vdash A \to f\,x \Vdash B$$

What? That's just **logical relations**.



Come on. That's **intuitionistic realizability**.

Fools ! That's **parametricity**.



Zo!

Making Everybody Agree

It's actually folklore that these techniques are essentially the same.

Making Everybody Agree

It's actually folklore that these techniques are essentially the same.

And there is already a parametricity translation for CIC! (Bernardy-Lasson)

We just have to adapt it to our exceptional translation.

Making Everybody Agree

It's actually folklore that these techniques are essentially the same.

And there is already a parametricity translation for CIC! (Bernardy-Lasson)

We just have to adapt it to our exceptional translation.

Idea:

From $\vdash M : A$ produce **two** sequents $\begin{cases} \vdash_{\mathrm{CIC}} [M] : [\![A]\!] \\ \quad + \\ \vdash_{\mathrm{CIC}} [M]_\varepsilon : [\![A]\!]_\varepsilon\ [M] \end{cases}$

where $[\![A]\!]_\varepsilon : [\![A]\!] \to \square$ is the validity predicate.

Parametric Exceptional Translation (Sketch)

Most notably,

$$\begin{array}{lcl}
[\![\Pi x : A.\, B]\!]_\varepsilon\ f & \equiv & \Pi(x : [\![A]\!])\,(x_\varepsilon : [\![A]\!]_\varepsilon\ x).\,[\![B]\!]_\varepsilon\ (f\,x) \\
[\![\mathbb{B}]\!]_\varepsilon\ b & \cong & b = [\mathtt{true}] + b = [\mathtt{false}] \\
[\![\bot]\!]_\varepsilon\ s & \cong & \bot
\end{array}$$

Parametric Exceptional Translation (Sketch)

Most notably,

$$\begin{array}{lcl}
[\![\Pi x : A.\, B]\!]_\varepsilon\ f & \equiv & \Pi (x : [\![A]\!])\,(x_\varepsilon : [\![A]\!]_\varepsilon\ x).\, [\![B]\!]_\varepsilon\ (f\,x) \\
[\![\mathbb{B}]\!]_\varepsilon\ b & \cong & b = [\mathtt{true}] + b = [\mathtt{false}] \\
[\![\bot]\!]_\varepsilon\ s & \cong & \bot
\end{array}$$

Every pure term is now automatically parametric.

If $\Gamma \vdash_{\mathrm{CIC}} M : A$ then $[\![\Gamma]\!]_\varepsilon \vdash_{\mathrm{CIC}} [M]_\varepsilon : [\![A]\!]_\varepsilon\ [M]$.

A Few Nice Results

Let's call $\mathcal{T}_{\mathbb{E}}^{p}$ the resulting theory. It inherits a lot from CIC!

Theorem (Consistency)

$\mathcal{T}_{\mathbb{E}}^{p}$ is consistent.

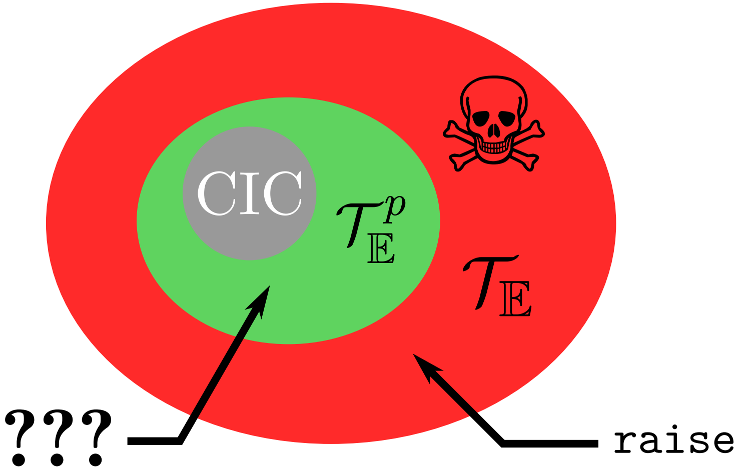
Theorem (Canonicity)

$\mathcal{T}_{\mathbb{E}}^{p}$ enjoys canonicity, i.e if $\vdash_{\mathcal{T}_{\mathbb{E}}^{p}} M : \mathbb{N}$ then $M \rightsquigarrow^{*} \bar{n} \in \bar{\mathbb{N}}$.

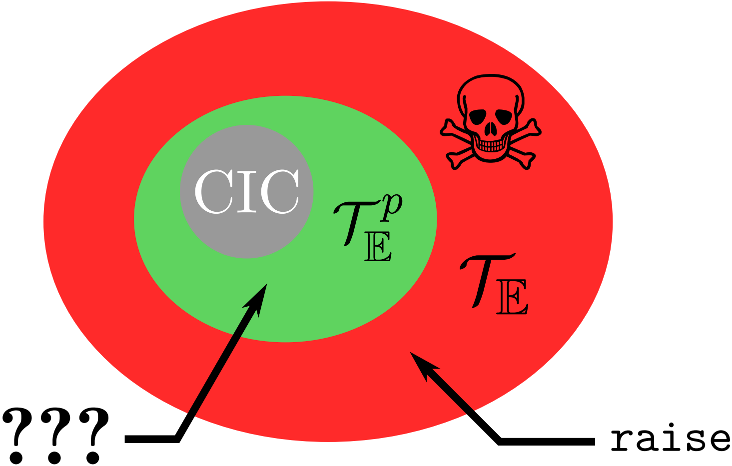
Theorem (Syntax)

$\mathcal{T}_{\mathbb{E}}^{p}$ has decidable type-checking, strong normalization and whatnot.

What If There Were No Cake?

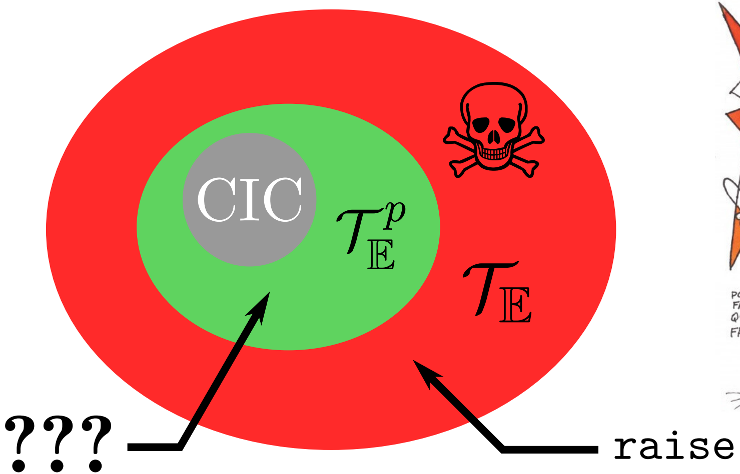


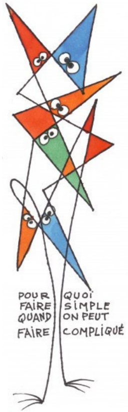
What If There Were No Cake?



Bernardy-Lasson parametricity is a conservative extension of CIC...

What If There Were No Cake?





Bernardy-Lasson parametricity is a conservative extension of CIC...

Less Is More

Spoiler

$\mathcal{T}_{\mathbb{E}}^{p}$ is **not** a conservative extension of CIC.

Less Is More

Spoiler

$\mathcal{T}^p_{\mathbb{E}}$ is **not** a conservative extension of CIC.

Intuitively,

- raising uncaught exceptions is forbidden in $\mathcal{T}^p_{\mathbb{E}}$

Less Is More

Spoiler

$\mathcal{T}_{\mathbb{E}}^{p}$ is **not** a conservative extension of CIC.

Intuitively,

- raising uncaught exceptions is forbidden in $\mathcal{T}_{\mathbb{E}}^{p}$
- … but you can still raise them locally
- … as long as you prove they don't escape!

Less Is More

Spoiler

$\mathcal{T}_{\mathbb{E}}^{p}$ is **not** a conservative extension of CIC.

Intuitively,

- raising uncaught exceptions is forbidden in $\mathcal{T}_{\mathbb{E}}^{p}$
- … but you can still raise them locally
- … as long as you prove they don't escape!

$\mathcal{T}_{\mathbb{E}}$ is the `unsafe` Coq fragment, and $\mathcal{T}_{\mathbb{E}}^{p}$ a semantical layer atop of it.

Less Is More

Spoiler

$\mathcal{T}_{\mathbb{E}}^{p}$ is **not** a conservative extension of CIC.

Intuitively,

- raising uncaught exceptions is forbidden in $\mathcal{T}_{\mathbb{E}}^{p}$
- ... but you can still raise them locally
- ... as long as you prove they don't escape!

$\mathcal{T}_{\mathbb{E}}$ is the `unsafe` Coq fragment, and $\mathcal{T}_{\mathbb{E}}^{p}$ a semantical layer atop of it.

Actually $\mathcal{T}_{\mathbb{E}}^{p}$ is the embodiement of Kreisel modified realizability in CIC.

Explaining the Analogy

	Kreisel realizability	$\mathcal{T}^p_{\mathbb{E}}$
Source theory	HA or HA^ω	CIC
Programming language	System T	$\mathcal{T}_{\mathbb{E}}$ ("unsafe Coq")
Logical meta-theory	HA^ω	CIC

Explaining the Analogy

	Kreisel realizability	$\mathcal{T}_{\mathbb{E}}^{p}$
Source theory	HA or HA^{ω}	CIC
Programming language	System T	$\mathcal{T}_{\mathbb{E}}$ (“unsafe Coq”)
Logical meta-theory	HA^{ω}	CIC

Kreisel realizability extends arithmetic with essentially two principles.

- $\mathrm{AC}_{\mathbb{N}} : (\forall n : \mathbb{N}.\, \exists m : \mathbb{N}.\, P\ (m, n)) \to \exists f : \mathbb{N} \to \mathbb{N}.\, \forall n : \mathbb{N}.\, P\ (n, f\ n)$
- $\mathrm{IP} : (\neg A \to \exists n : \mathbb{N}.\, P\ n) \to \exists n : \mathbb{N}.\, \neg A \to P\ n$

Choice

$$\mathrm{AC}_{\mathbb{N}} : (\forall n : \mathbb{N}.\, \exists m : \mathbb{N}.\, P\ (m, n)) \rightarrow \exists f : \mathbb{N} \rightarrow \mathbb{N}.\, \forall n : \mathbb{N}.\, P\ (n, f\ n)$$

Not much to say here.

In Kreisel realizability, $\mathrm{AC}_{\mathbb{N}}$ is a consequence of canonicity of System T.

Choice

$$\mathrm{AC}_{\mathbb{N}} : (\forall n : \mathbb{N}.\, \exists m : \mathbb{N}.\, P\ (m, n)) \rightarrow \exists f : \mathbb{N} \rightarrow \mathbb{N}.\, \forall n : \mathbb{N}.\, P\ (n, f\ n)$$

Not much to say here.

In Kreisel realizability, $\mathrm{AC}_{\mathbb{N}}$ is a consequence of canonicity of System T.

In $\mathcal{T}^{p}_{\mathbb{E}}$, $\mathrm{AC}_{\mathbb{N}}$ is a consequence of dependent elimination.

The latter is in turn meta-theoretically justified by canonicity.

Choice

$$\mathrm{AC}_{\mathbb{N}} : (\forall n : \mathbb{N}.\, \exists m : \mathbb{N}.\, P\ (m, n)) \rightarrow \exists f : \mathbb{N} \rightarrow \mathbb{N}.\, \forall n : \mathbb{N}.\, P\ (n, f\ n)$$

Not much to say here.

In Kreisel realizability, $\mathrm{AC}_{\mathbb{N}}$ is a consequence of canonicity of System T.

In $\mathcal{T}^{p}_{\mathbb{E}}$, $\mathrm{AC}_{\mathbb{N}}$ is a consequence of dependent elimination.

The latter is in turn meta-theoretically justified by canonicity.

In both cases, choice is built-in and a consequence of canonicity.

Independence of Premises

$$\mathrm{IP} : (\neg A \to \exists n : \mathbb{N}.\, P\ n) \to \exists n : \mathbb{N}.\, \neg A \to P\ n$$

That one is interesting! A unforeseen consequence of a subtle **bug**.

Kreisel's bug

Every type of realizers is inhabited. In particular, $[\![\bot]\!]_{\mathrm{KR}} \equiv \mathbb{N}$.

Independence of Premises

$$\mathrm{IP} : (\neg A \to \exists n : \mathbb{N}.\, P\ n) \to \exists n : \mathbb{N}.\, \neg A \to P\ n$$

That one is interesting! A unforeseen consequence of a subtle **bug**.

Kreisel's bug

Every type of realizers is inhabited. In particular, $[\![\bot]\!]_{\mathrm{KR}} \equiv \mathbb{N}$.

The realizer of IP critically relies on that!

Assuming System T had an empty type $\mathbb{0}$, and setting $[\![\bot]\!]_{\mathrm{KR}} \equiv \mathbb{0}$

- KR is still a model of HA
- KR still validates $\mathrm{AC}_{\mathbb{N}}$
- KR **doesn't** validate IP anymore

Volem Independència

$$\mathrm{IP} : (\neg A \to \exists n : \mathbb{N}.\, P\ n) \to \exists n : \mathbb{N}.\, \neg A \to P\ n$$

Theorem $(\mathrm{CIC} + \mathrm{IP})$

$\mathcal{T}_{\mathbb{E}}^{p}$ *validates* IP*, owing to the fact that in* $\mathcal{T}_{\mathbb{E}}$*, every type is inhabited.*

Volem Independència

$$\mathrm{IP} : (\neg A \to \exists n : \mathbb{N}.\, P\ n) \to \exists n : \mathbb{N}.\, \neg A \to P\ n$$

Theorem $(\mathrm{CIC} + \mathrm{IP})$

$\mathcal{T}_{\mathbb{E}}^{p}$ *validates* IP, *owing to the fact that in* $\mathcal{T}_{\mathbb{E}}$, *every type is inhabited.*

Proof (sketch).

In $\mathcal{T}_{\mathbb{E}}$, build a term `ip` : IP

- Given $f : \neg A \to \Sigma n : \mathbb{N}.\, P\ n$, apply it to `raise` $(\neg A)\ e$.
- If the returned integer is pure, return it with the associated proof.
- Otherwise, return a dummy integer and failing proof.

Easy to show that `ip` is actually valid in $\mathcal{T}_{\mathbb{E}}^{p}$. $\square$

Another Result for Free

Recall Markov's principle:

$$\Pi P : \mathbb{N} \to \mathbb{B}.\, \neg\neg(\Sigma n : \mathbb{N}.\, P\ n = \texttt{true}) \to \Sigma n : \mathbb{N}.\, P\ n = \texttt{true} \qquad \text{(MP)}$$

Another Result for Free

Recall Markov's principle:

$$\Pi P : \mathbb{N} \to \mathbb{B}.\, \neg\neg(\Sigma n : \mathbb{N}.\, P\ n = \mathtt{true}) \to \Sigma n : \mathbb{N}.\, P\ n = \mathtt{true} \qquad (\mathrm{MP})$$

Kreisel's Razor

Pick two out of three: $\{$canonicity, IP, $\mathrm{MP}\}$.

Another Result for Free

Recall Markov's principle:

$$\Pi P : \mathbb{N} \to \mathbb{B}.\, \neg\neg(\Sigma n : \mathbb{N}.\, P\ n = \mathtt{true}) \to \Sigma n : \mathbb{N}.\, P\ n = \mathtt{true} \qquad (\mathrm{MP})$$

Kreisel's Razor

Pick two out of three: $\{$canonicity, IP, $\mathrm{MP}\}$.

$\mathrm{IP} + \mathrm{MP} \Rightarrow \Pi P : \mathbb{N} \to \mathbb{B}.\, \Sigma n : \mathbb{N}.\, \Pi m : \mathbb{N}.\, P\ m = \mathtt{true} \to P\ n = \mathtt{true}$

Together with canonicity, this solves the halting problem.

Another Result for Free

Recall Markov's principle:

$$\Pi P : \mathbb{N} \to \mathbb{B}.\, \neg\neg(\Sigma n : \mathbb{N}.\, P\ n = \mathtt{true}) \to \Sigma n : \mathbb{N}.\, P\ n = \mathtt{true} \qquad (\mathrm{MP})$$

Kreisel's Razor

Pick two out of three: $\{$canonicity, IP, $\mathrm{MP}\}$.

$$\mathrm{IP} + \mathrm{MP} \Rightarrow \Pi P : \mathbb{N} \to \mathbb{B}.\, \Sigma n : \mathbb{N}.\, \Pi m : \mathbb{N}.\, P\ m = \mathtt{true} \to P\ n = \mathtt{true}$$

Together with canonicity, this solves the halting problem.

Corollary

$\nvdash_{\mathcal{T}^p_{\mathbb{E}}} \mathrm{MP}$ and thus $\nvdash_{\mathrm{CIC}} \mathrm{MP}$.

(This was proved recently by Coquand-Mannaa, although in a completely different way.)

Function Intensionality

Another interesting consequence that is similar to what happens in KR.

- $\mathcal{T}_{\mathbb{E}}^{p}$ satisfies definitional η-expansion: $\lambda x : A.\, M\, x \equiv M$.
- But it violates function extensionality!

$$\vdash_{\mathcal{T}_{\mathbb{E}}^{p}} \Pi i : \mathbb{1}.\, i = \mathtt{tt} \qquad \text{and} \qquad \vdash_{\mathcal{T}_{\mathbb{E}}^{p}} (\lambda i : \mathbb{1}.\, i) \neq (\lambda i : \mathbb{1}.\, \mathtt{tt})$$

Function Intensionality

Another interesting consequence that is similar to what happens in KR.

- $\mathcal{T}_{\mathbb{E}}^{p}$ satisfies definitional η-expansion: $\lambda x : A.\, M\, x \equiv M$.
- But it violates function extensionality!

$$\vdash_{\mathcal{T}_{\mathbb{E}}^{p}} \Pi i : \mathbb{1}.\, i = \mathtt{tt} \qquad \text{and} \qquad \vdash_{\mathcal{T}_{\mathbb{E}}^{p}} (\lambda i : \mathbb{1}.\, i) \neq (\lambda i : \mathbb{1}.\, \mathtt{tt})$$

The reason is that there are invalid proofs of $\mathbb{1}$.

You cannot build them, but they exists as phantom arguments.

What Else?

What kind of similar horrors can we do in $\mathcal{T}_{\mathbb{E}}^{p}$?

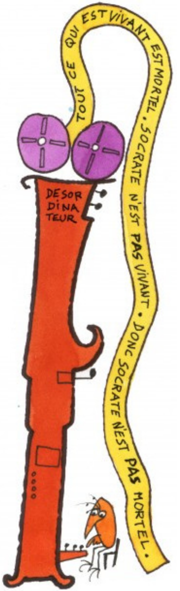
- I don't know!
- But there are probably lessons to be taken from realizability
- I'm probably pissing off both HoTT and PRL zealots by now

Get You A Larger Coq, Today!

We implemented $\mathcal{T}_{\mathbb{E}}$ and $\mathcal{T}_{\mathbb{E}}^{p}$ in Coq in a plugin.

`https://github.com/CoqHott/exceptional-tt`

- Allows to add exceptions to Coq just today.
- Compile effectful terms on the fly.
- Allows to reason about them in Coq.
- Write mind-blowing low-level code!



If You Were Sleeping During The Talk

$\mathcal{T}_{\mathbb{E}}$, a type theory that allows failure!

- Inconsistent as a logical theory
- A dependently-typed effectful programming language
- Can still be used for proof extraction like Friedman's A-translation

If You Were Sleeping During The Talk

$\mathcal{T}_{\mathbb{E}}$, a type theory that allows failure!

- Inconsistent as a logical theory
- A dependently-typed effectful programming language
- Can still be used for proof extraction like Friedman's A-translation

$\mathcal{T}_{\mathbb{E}}^{p}$, a type theory that allows local failure!

- A safe layer atop $\mathcal{T}_{\mathbb{E}}$ that enforces consistency
- Strict superset of CIC: proves IP, $\neg\texttt{funext}$, disproves MP

If You Were Sleeping During The Talk

$\mathcal{T}_{\mathbb{E}}$, a type theory that allows failure!

- Inconsistent as a logical theory
- A dependently-typed effectful programming language
- Can still be used for proof extraction like Friedman's A-translation

$\mathcal{T}_{\mathbb{E}}^{p}$, a type theory that allows **local** failure!

- A safe layer atop $\mathcal{T}_{\mathbb{E}}$ that enforces consistency
- Strict superset of CIC: proves IP, $\neg\texttt{funext}$, disproves MP

Both of them justified by purely syntactical means!

If You Were Sleeping During The Talk

$\mathcal{T}_{\mathbb{E}}$, a type theory that allows failure!

- Inconsistent as a logical theory
- A dependently-typed effectful programming language
- Can still be used for proof extraction like Friedman's A-translation

$\mathcal{T}^{p}_{\mathbb{E}}$, a type theory that allows **local** failure!

- A safe layer atop $\mathcal{T}_{\mathbb{E}}$ that enforces consistency
- Strict superset of CIC: proves IP, $\neg\texttt{funext}$, disproves MP

Both of them justified by purely syntactical means!

"The more it fails, the more likely it will eventually succeed."

TODO When I Have a Permanent Position

- $\mathcal{T}_{\mathbb{E}}$ looks like a good intermediate language for model building
- The Calculus of Shadok Constructions
- Potential applications to Gradual Typing?
- Syntactic models are super cool! Let's write more!


C'EST TOUT POUR AUJOURD'HUI

Food For Thought

It seems you need to have a name starting with K to name a realizability.

Kleene
Kreisel
Krivine