Effectful Type Theories

Pierre-Marie Pédrot

Gallinette (Inria)

CASS 2020

It is time to CIC ass and chew bubble-gum

CIC, the Calculus of Inductive Constructions.

It is 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, 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 is 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

Bird's Eye View of CIC

Recall from what you have seen

- A purely functional language
- ... plus Π generalizing arrows
- ... plus ADTs with generalized pattern-matching
- ... where types are terms
- Proof and computation living in harmony

Bird's Eye View of CIC

Recall from what you have seen

- A purely functional language
- ... plus Π generalizing arrows
- ... plus ADTs with generalized pattern-matching
- ... where types are terms
- Proof and computation living in harmony

CIC is quite a complex beast!

“The earthly paradise of the proof-program correspondence”

¿ "The earthly paradise of the proof-program correspondence" ?

The CIC brothers

Actually not quite one single theory.

Several flags tweaking the Coq kernel:

- Impredicative Set
- Type-in-type
- Indices Matter
- Cumulative inductive types
- ...

The CIC brothers

Actually not quite one single theory.

Several flags tweaking the Coq kernel:

- Impredicative Set
- Type-in-type
- Indices Matter
- Cumulative inductive types
- ...

The Many Calculi of Inductive Constructions.

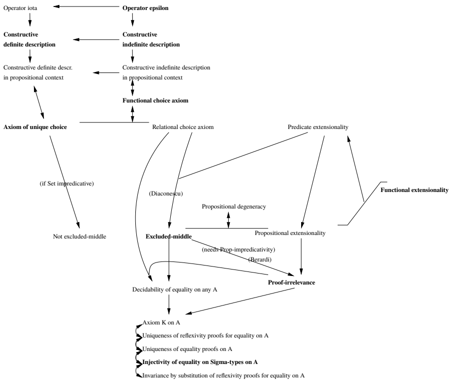
In the Axiom Jungle

A crazy amount of axioms used in the wild!

A crazy amount of axioms used in the wild!

The *claſſical ſet-theory* pole:

- Excluded middle, UIP, choice



In the Axiom Jungle

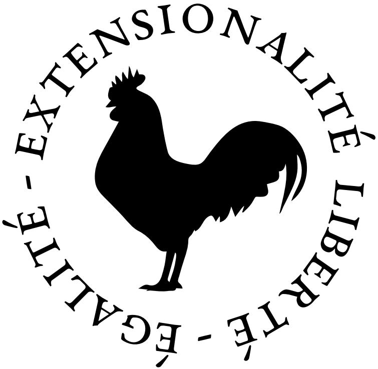
A crazy amount of axioms used in the wild!

The classical set-theory pole:

- Excluded middle, UIP, choice

The EXTENSIONAL pole:

- Funext, Propext, Bisim-is-eq



In the Axiom Jungle

A crazy amount of axioms used in the wild!

The claſſical ſet-theory pole:

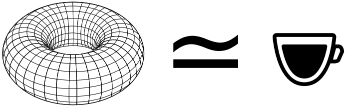
- Excluded middle, UIP, choice

The Extensional pole:

- Funext, Propext, Bisim-is-eq

The univalent pole:

- Univalence, what else?



« A mathematician is a device for turning toruses into equalities (up to homotopy). »

A crazy amount of axioms used in the wild!

The classical set-theory pole:

- Excluded middle, UIP, choice

The Extensional pole:

- Funext, Propext, Bisim-is-eq

The univalent pole:

- Univalence, what else?

The $\varepsilon\chi o\tau\iota c$ pole:

- Anti-classical axioms (???)

In the Axiom Jungle

A crazy amount of axioms used in the wild!

The classical set-theory pole:

- Excluded middle, UIP, choice

The EXTENSIONAL pole:

- Funext, Propext, Bisim-is-eq

The univalent pole:

- Univalence, what else?

The $\varepsilon\chi o\tau\iota c$ pole:

- Anti-classical axioms (???)

Varying degrees of compatibility.

Reality Check

Theorem 0

Axioms Suck.

Reality Check

Theorem 0

Axioms Suck.

Proof.

- They break computation.
- They are hard to justify.
- They might be incompatible with one another.

□

Reality Check

Theorem 0

Axioms Suck.

Proof.

- They break computation.
- They are hard to justify.
- They might be incompatible with one another.

$\square$

Mathematicians may not care too much, but...

CIC suffers from an even more **fundamental** flaw.

The Most Important Issue of Them All

CIC suffers from an even more **fundamental** flaw.

- You want to show the wonders of Coq to a fellow programmer
- You fire your favourite IDE
- ... and you're asked the DREADFUL question.

The Most Important Issue of Them All

CIC suffers from an even more **fundamental** flaw.

- You want to show the wonders of Coq to a fellow programmer
- You fire your favourite IDE
- … and you're asked the DREADFUL question.

COULD YOU WRITE A HELLO WORLD PROGRAM PLEASE?

Sad reality (a.k.a. Curry-Howard)

Intuitionistic Logic $\Leftrightarrow$ **Functional** Programming

Sad reality (a.k.a. Curry-Howard)

Intuitionistic Logic $\Leftrightarrow$ **Functional** Programming

Coq is even purer than Haskell:

- No mutable state (obviously)
- No exceptions (Haskell has them somehow)
- No arbitrary recursion
- and also no **HELLO WORLD** !

Sad reality (a.k.a. Curry-Howard)

Intuitionistic Logic $\Leftrightarrow$ **Functional** Programming

Coq is even purer than Haskell:

- No mutable state (obviously)
- No exceptions (Haskell has them somehow)
- No arbitrary recursion
- and also no **HELLO WORLD** !

We want a type theory with effects !

The Good News

Intuitionistic Logic $\Leftrightarrow$ **Functional** Programming

The Good News

Intuitionistic Logic $\Leftrightarrow$ **Functional** Programming

is by folklore the same as

Non-Intuitionistic Logic $\Leftrightarrow$ **Impure** Programming

The Good News

Intuitionistic Logic ⇔ **Functional** Programming

is by folklore the same as

Non-Intuitionistic Logic ⇔ **Impure** Programming

- `callcc` gives classical logic
- Delimited continuations prove Markov's principle
- Exceptions implement Markov's rule
- Well-behaved global cells provide univalence
- ...

We want a type theory with **effects** !

Thesis

We want a type theory with effects!

Thesis

We want a type theory with effects!

1. To program more (exceptions, non-termination...)
2. To prove more (classical logic, univalence...)
3. To write Hello World.

Thesis

We want a type theory with effects!

1. To program more (exceptions, non-termination...)
2. To prove more (classical logic, univalence...)
3. To write Hello World.

It's not just randomly coming up with typing rules though.

There are some good properties we want to ensure...

Thesis

We want a type theory with effects!

1. To program more (exceptions, non-termination...)
2. To prove more (classical logic, univalence...)
3. To write Hello World.

It's not just randomly coming up with typing rules though.

There are some good properties we want to ensure...

We want a model of type theory with effects.

Good Properties

Consistency There is no proof of `False`.

Implementability Type-checking is decidable.

Canonicity Closed integers are indeed integers, i.e

$$\vdash M : \mathbb{N} \quad \text{implies} \quad M \equiv \mathtt{S} \ldots \mathtt{S}\,\mathtt{0}$$

Assuming we have a notion of reduction compatible with conversion:

Normalization Reduction is normalizing

Subject reduction Reduction is compatible with typing

Good Properties

Consistency There is no proof of `False`.

Implementability Type-checking is decidable.

Canonicity Closed integers are indeed integers, i.e

$$\vdash M : \mathbb{N} \quad \text{implies} \quad M \equiv \mathtt{S} \ldots \mathtt{S}\,\mathtt{0}$$

Assuming we have a notion of reduction compatible with conversion:

Normalization Reduction is normalizing

Subject reduction Reduction is compatible with typing

Some of these properties are interdependent

From Hell

Semantics of type theory have a fame of being horribly complex.

From Hell

Semantics of type theory have a fame of being horribly complex.

I won't lie: **they are**. But part of this fame is due to its usual models.

From Hell

Semantics of type theory have a fame of being horribly complex.

I won't lie: **they are**. But part of this fame is due to its usual models.

Roughly three standard families of models:

- **Set-theoretical** models
- **Realizability** models
- **Categorical** models

Let's review them quickly!

The Set-Theoretical Model

Because Sets are a (crappy) type theory.

The Set-Theoretical Model

Because Sets are a (crappy) type theory.

Interpret everything as sets and expect $\vdash_{\mathsf{CIC}} M : A \Rightarrow \vdash_{\mathsf{ZFC}} [\![M]\!] \in [\![A]\!]$.

Pro

- Well-known and trusted target
- Imports ZFC properties.

Con

- Forego syntax, computation and decidability
- No effects in sight.
- Imports ZFC properties.

The Realizability Models

Construct programs that respect properties.

The Realizability Models

Construct programs that respect properties.

- Terms $M \rightsquigarrow$ programs $[M]$ (variable languages as a target)
- Types $A \rightsquigarrow$ meta-theoretical predicates $[\![A]\!]$
- $\vdash_{\mathsf{CIC}} M : A \quad \Rightarrow \quad [M] \in [\![A]\!]$

Pro

- Some preservation of syntax and computability

Con

- Usually crazily undecidable
- Meta-theory can be arbitrary crap, including ZFC

The Categorical Models

Abstract / obfuscated description of type theory.

The Categorical Models

Abstract / obfuscated description of type theory.

Rephrase the rules of CIC in a categorical way.

Pro

- Very abstract and subsumes both previous examples
- Somewhat "easier" to show some structure is a model of TT

Con

- Same limitations as the previous examples
- Mostly useless to actually construct a model
- Yet another syntax, usually arcane and ill-fitted

Down With Semantics

In this talk, I'd like to advocate for a fourth approach.

In this talk, I'd like to advocate for a fourth approach.

Syntactic Models

What is a model?

- Takes syntax as input.
- Interprets it into some low-level language.
- Must preserve the meaning of the source.
- Refines the behaviour of under-specified structures.

What is a model?

- Takes syntax as input.
- Interprets it into some low-level language.
- Must preserve the meaning of the source.
- Refines the behaviour of under-specified structures.

Five seconds of thorough thinking for the sleepy ones.

What is a model?

- Takes syntax as input.
- Interprets it into some low-level language.
- Must preserve the meaning of the source.
- Refines the behaviour of under-specified structures.

Five seconds of thorough thinking for the sleepy ones.

« This is a *compiler*... »

... and you know it.

Curry-Howard Orthodoxy

Let's look at what Curry-Howard provides in simpler settings.

Program Translations $\Leftrightarrow$ Logical Interpretations

Curry-Howard Orthodoxy

Let's look at what Curry-Howard provides in simpler settings.

Program Translations $\Leftrightarrow$ Logical Interpretations

On the **programming** side, enrich the language by program translation.

- Monadic style à la Haskell
- Compilation of higher-level constructs down to assembly

Curry-Howard Orthodoxy

Let's look at what Curry-Howard provides in simpler settings.

Program Translations $\Leftrightarrow$ Logical Interpretations

On the **programming** side, enrich the language by program translation.

- Monadic style à la Haskell
- Compilation of higher-level constructs down to assembly

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

- Double-negation $\Rightarrow$ classical logic (`callcc`)
- Friedman's trick $\Rightarrow$ Markov's rule (exceptions)
- Forcing $\Rightarrow$ $\neg$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**.

We take the following act of faith for granted.

CIC is.

Syntactic Models

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

We take the following act of faith for granted.

CIC is.

Not caring for its soundness, implementation, whatever. It just is.

Do everything by interpreting the new theories relatively to this foundation!

Suppress technical and cognitive burden by lowering impedance mismatch.

Syntactic Models II

Step 0: Fix a theory $\mathcal{T}$ as close as possible* to CIC, ideally CIC $\subseteq \mathcal{T}$.

Syntactic Models II

Step 0: Fix a theory $\mathcal{T}$ as close as possible* to CIC, ideally CIC $\subseteq \mathcal{T}$.

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

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

Syntactic Models II

Step 0: Fix a theory $\mathcal{T}$ as close as possible* to CIC, ideally CIC $\subseteq \mathcal{T}$.

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

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

Step 2: Flip views and actually pose

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

Syntactic Models II

Step 0: Fix a theory $\mathcal{T}$ as close as possible* to CIC, ideally CIC $\subseteq \mathcal{T}$.

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

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

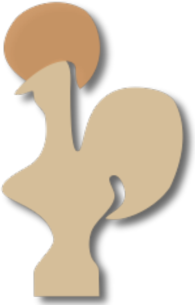
Step 2: Flip views and actually pose

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

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

Anatomy of a syntactic model

COMPILATION



$\vdash_{\mathsf{CIC}^{++}} M : A \qquad \rightsquigarrow \qquad \vdash_{\mathsf{CIC}} [M] : [\![A]\!]$

« CIC, the LLVM of type theory »

Syntactic Models III

Obviously, that's subtle.

- The translation $[\cdot]$ must preserve typing (not easy)
- In particular, it must preserve conversion (even worse)

Syntactic Models III

Obviously, that's subtle.

- The translation $[\cdot]$ must preserve typing (not easy)
- In particular, it must preserve conversion (even worse)

Yet, a lot of nice consequences.

- Does not require non-type-theoretical foundations (*monism*)
- **Can be implemented in Coq** (*software monism*)
- Easy to show (relative) consistency, look at $[\![\texttt{False}]\!]$
- Inherit properties from CIC: computationality, decidability...

In This Talk

We will show three simple models

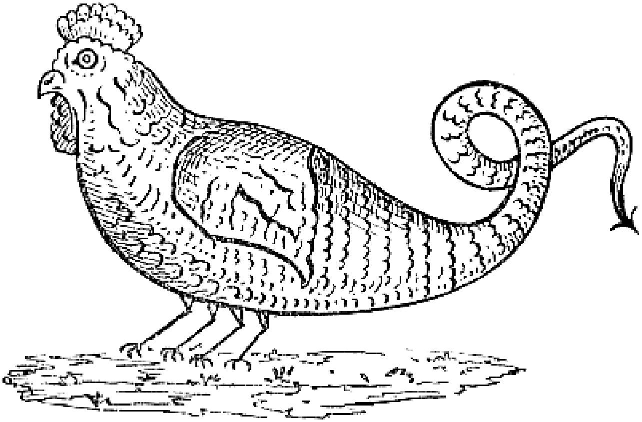
- Warmup: Intentional types (not an effect)
- Part I: Read-only cell
- Part II: Exceptions (two variants)

In This Talk

We will show three simple models

- Warmup: Intentional types (not an effect)
- Part I: Read-only cell
- Part II: Exceptions (two variants)

I and II feature fundamental interactions between effects and dependency.



Intensional Types, a.k.a. **Dynamically Typed CIC**

(effect-free introductory example)

Intensional Types

The intensional types translation extends type theory with

$$\begin{array}{lcl}
\texttt{flip} & : & \square \to \square \\
\texttt{flip_equiv} & : & \Pi(A : \square).\, \texttt{flip}\ A \cong A \\
\texttt{flip_neq} & : & \Pi(A : \square).\, \texttt{flip}\ A \neq A
\end{array}$$

Intensional Types

The intensional types translation extends type theory with

$$
\begin{array}{lcl}
\texttt{flip} & : & \square \to \square \\
\texttt{flip\_equiv} & : & \Pi(A : \square).\, \texttt{flip}\ A \cong A \\
\texttt{flip\_neq} & : & \Pi(A : \square).\, \texttt{flip}\ A \neq A
\end{array}
$$

This breaks amongst other things univalence...

The Intensional Types Implementation

Intuitively:

- Translate $A : \square$ into $[A] : \square \times \mathbb{B}$
- Translate $M : A$ into $[M] : [A].\pi_1$

The Intensional Types Implementation

Intuitively:

- Translate $A : \square$ into $[A] : \square \times \mathbb{B}$
- Translate $M : A$ into $[M] : [A].\pi_1$

$$\begin{array}{lcl}
[\![A]\!] & \equiv & [A].\pi_1 \\
[\square] & \equiv & (\square \times \mathbb{B}, \mathtt{true}) \\
[\Pi x : A.\, B] & \equiv & (\Pi x : [\![A]\!].\, [\![B]\!], \mathtt{true}) \\
[x] & \equiv & x \\
[M\ N] & \equiv & [M]\ [N] \\
[\lambda x : A.\, M] & \equiv & \lambda x : [\![A]\!].\, [M]
\end{array}$$

Types contain a boolean not used for their inhabitants!

The Intensional Types Implementation

Intuitively:

- Translate $A : \Box$ into $[A] : \Box \times \mathbb{B}$
- Translate $M : A$ into $[M] : [A].\pi_1$

$$\begin{array}{lcl}
[\![A]\!] & \equiv & [A].\pi_1 \\
[\Box] & \equiv & (\Box \times \mathbb{B}, \mathtt{true}) \\
[\Pi x : A.\, B] & \equiv & (\Pi x : [\![A]\!].\, [\![B]\!], \mathtt{true}) \\
[x] & \equiv & x \\
[M\ N] & \equiv & [M]\ [N] \\
[\lambda x : A.\, M] & \equiv & \lambda x : [\![A]\!].\, [M]
\end{array}$$

Types contain a boolean not used for their inhabitants!

Soundness

If $\vec{x} : \Gamma \vdash M : A$ then $\vec{x} : [\![\Gamma]\!] \vdash [M] : [\![A]\!]$.

Extending the Intensional Types

Let's define the new operations obtained through the translation.

$$
\begin{array}{lcl}
[\texttt{flip}] & : & [\![\square \to \square]\!] \\
[\texttt{flip}] & : & \square \times \mathbb{B} \to \square \times \mathbb{B} \\
[\texttt{flip}] & \equiv & \lambda(A, b).\,(A, \texttt{negb}\ b) \\
\\
[\texttt{flip\_equiv}] & : & [\![\Pi A : \square.\, \texttt{flip}\ A \cong A]\!] \\
[\texttt{flip\_equiv}] & \equiv & \ldots \\
\\
[\texttt{flip\_neq}] & : & [\![\Pi A : \square.\, \texttt{flip}\ A \neq A]\!] \\
[\texttt{flip\_neq}] & : & \Pi A : \square \times \mathbb{B}.\, [\texttt{flip}]\ A \neq A \\
[\texttt{flip\_equiv}] & \equiv & \ldots
\end{array}
$$

- $[\![\texttt{flip}\ A]\!] \equiv [\![A]\!]$
- And isomorphism only depends on $[\![A]\!]$
- But (intensional) equality observes the boolean...

Basilisk for Realz

This one example is not very interesting.

Basilisk for Realz

This one example is not very interesting.

You can do much better: a real mix of Python and Coq!

Basilisk for Realz

This one example is not very interesting.

You can do much better: a real mix of Python and Coq!

- Assuming the target theory features induction-recursion
- Represent (source) types by their code
- This gives a real type-quote function in the source theory

$$
\begin{array}{rl}
\texttt{type\_rect} : & \Pi(P : \square \to \square). \\
& \quad P\ \square \to \\
& \quad (\Pi(A : \square)\,(B : A \to \square).\ P\ A \to (\Pi x : A.\ P\ (B\ x)) \to \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad P\ (\Pi x : A.\ B)) \to \\
& \quad P\ \mathbb{N} \to \\
& \quad \ldots \to \\
& \Pi(A : \square).\ P\ A
\end{array}
$$

Basilisk for Realz

This one example is not very interesting.

You can do much better: a real mix of Python and Coq!

- Assuming the target theory features induction-recursion
- Represent (source) types by their code
- This gives a real type-quote function in the source theory

$$
\begin{aligned}
\texttt{type\_rect} : \quad & \Pi(P : \square \to \square).\\
& \quad P\ \square \to\\
& \quad (\Pi(A : \square)\,(B : A \to \square).\, P\ A \to (\Pi x : A.\, P\ (B\ x)) \to\\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad P\ (\Pi x : A.\, B)) \to\\
& \quad P\ \mathbb{N} \to\\
& \quad \ldots \to\\
& \Pi(A : \square).\, P\ A
\end{aligned}
$$

Coq is compatible with dynamic types!!!

The reader translation, a.k.a. **Baby Forcing**

Overview

Essentially the same as Haskell's reader effect*.

- There is a global unnamed cell
- That can be read
- That can be updated **in a well-scoped way**

Not quite a state!

* main difference is that we're in call-by-name.

The Reader Translation

Assume some fixed cell type $\mathbb{R} : \square$.

The Reader Translation

Assume some fixed cell type $\mathbb{R} : \square$.

The reader translation extends CIC into $\mathsf{CIC}_{\mathbb{R}}$, with

$$\begin{array}{rrcl} & \texttt{read} & : & \mathbb{R} \\ & \texttt{into} & : & \square \to \mathbb{R} \to \square \\ & \texttt{enter} & : & \Pi(A : \square).\, A \to \Pi r : \mathbb{R}.\, \texttt{into}\ A\ r \\ (\text{morally} & \texttt{enter} & : & \Pi(A : \square).\, A \to \mathbb{R} \to A) \end{array}$$

The Reader Translation

Assume some fixed cell type $\mathbb{R} : \square$.

The reader translation extends CIC into $\text{CIC}_{\mathbb{R}}$, with

$$\begin{array}{rrcl} & \mathtt{read} & : & \mathbb{R} \\ & \mathtt{into} & : & \square \to \mathbb{R} \to \square \\ & \mathtt{enter} & : & \Pi(A : \square).\, A \to \Pi r : \mathbb{R}.\, \mathtt{into}\ A\ r \\ (\text{morally} & \mathtt{enter} & : & \Pi(A : \square).\, A \to \mathbb{R} \to A) \end{array}$$

satisfying the expected definitional equations, e.g.

$$\begin{array}{rcl} \mathtt{enter}\ \mathbb{R}\ \mathtt{read}\ M & \equiv & M \\ \mathtt{enter}\ \mathbb{R}\ M\ \mathtt{read} & \equiv & M \end{array}$$

The Reader Implementation

Assuming a variable $r : \mathbb{R}$, intuitively:

- Translate $A : \square$ into $[A]_r : \square$
- Translate $M : A$ into $[M]_r : [A]_r$

The Reader Implementation

Assuming a variable $r : \mathbb{R}$, intuitively:

- Translate $A : \square$ into $[A]_r : \square$
- Translate $M : A$ into $[M]_r : [A]_r$

$$
\begin{array}{lcl}
[\![A]\!] & \equiv & \Pi r : \mathbb{R}.\, [A]_r \\
[\square]_r & \equiv & \square \\
[\Pi x : A.\, B]_r & \equiv & \Pi x : [\![A]\!].\, [B]_r \\
[x]_r & \equiv & x\ r \\
[M\ N]_r & \equiv & [M]_r\ (\lambda s : \mathbb{R}.\, [N]_s) \\
[\lambda x : A.\, M]_r & \equiv & \lambda x : [\![A]\!].\, [M]_r
\end{array}
$$

All variables are thunked w.r.t. $\mathbb{R}$!

The Reader Implementation

Assuming a variable $r : \mathbb{R}$, intuitively:

- Translate $A : \square$ into $[A]_r : \square$
- Translate $M : A$ into $[M]_r : [A]_r$

$$
\begin{array}{lcl}
[\![A]\!] & \equiv & \Pi r : \mathbb{R}.\, [A]_r \\
[\square]_r & \equiv & \square \\
[\Pi x : A.\, B]_r & \equiv & \Pi x : [\![A]\!].\, [B]_r \\
[x]_r & \equiv & x\ r \\
[M\ N]_r & \equiv & [M]_r\ (\lambda s : \mathbb{R}.\, [N]_s) \\
[\lambda x : A.\, M]_r & \equiv & \lambda x : [\![A]\!].\, [M]_r
\end{array}
$$

All variables are thunked w.r.t. $\mathbb{R}$!

Soundness

We have $\Gamma \vdash M : A$ implies $[\![\Gamma]\!], r : \mathbb{R} \vdash [M]_r : [A]_r$.

The Reader Implementation: Inductive Types

PLT tells us we have to take $[A + B]_r \equiv [\![A]\!] + [\![B]\!]$.

$$\begin{array}{rcl} [A + B]_r & \equiv & [\![A]\!] + [\![B]\!] \\ [\mathtt{inl}\ M]_r & \equiv & \mathtt{inl}\ (\Pi s : \mathbb{R}.\, [M]_s) \\ [\mathtt{inr}\ M]_r & \equiv & \mathtt{inr}\ (\Pi s : \mathbb{R}.\, [M]_s) \end{array}$$

The Reader Implementation: Inductive Types

PLT tells us we have to take $[A + B]_r \equiv [\![A]\!] + [\![B]\!]$.

$$
\begin{array}{rcl}
[A + B]_r & \equiv & [\![A]\!] + [\![B]\!] \\
[\mathtt{inl}\ M]_r & \equiv & \mathtt{inl}\ (\Pi s : \mathbb{R}.\, [M]_s) \\
[\mathtt{inr}\ M]_r & \equiv & \mathtt{inr}\ (\Pi s : \mathbb{R}.\, [M]_s)
\end{array}
$$

It's possible to implement **non-dependent** pattern-matching as usual.

PLT tells us we have to take $[A + B]_r \equiv [\![A]\!] + [\![B]\!]$.

$$\begin{array}{lcl} [A + B]_r & \equiv & [\![A]\!] + [\![B]\!] \\ [\mathtt{inl}\ M]_r & \equiv & \mathtt{inl}\ (\Pi s : \mathbb{R}.\, [M]_s) \\ [\mathtt{inr}\ M]_r & \equiv & \mathtt{inr}\ (\Pi s : \mathbb{R}.\, [M]_s) \end{array}$$

It's possible to implement **non-dependent** pattern-matching as usual.

$$[\mathtt{elim}_+]_r : [\![\Pi P : \square.\, (A \to P) \to (B \to P) \to A + B \to P]\!] \equiv \Pi(P : \mathbb{R} \to \square).$$
$$(\Pi s : \mathbb{R}.\, [\![A]\!] \to P\ s) \to (\Pi s : \mathbb{R}.\, [\![B]\!] \to P\ s) \to (\mathbb{R} \to [\![A]\!] + [\![B]\!]) \to P\ r$$

$$\begin{array}{lcl} \mathtt{elim}_+\ P\ N_l\ N_r\ (\mathtt{inl}\ M) & \equiv & N_l\ M \\ \mathtt{elim}_+\ P\ N_l\ N_r\ (\mathtt{inr}\ M) & \equiv & N_r\ M \end{array}$$

Uh-oh

Unfortunately, It's **not possible** to implement **dependent** elimination!

$$[\![\Pi P.\,(\Pi(x:A).\,P\;(\mathtt{inl}\;x)) \to (\Pi(y:B).\,P\;(\mathtt{inr}\;y)) \to \Pi b:A+B.\,P\;b]\!]$$

$$\equiv$$

$$\begin{aligned}
&\Pi P : \mathbb{R} \to (\mathbb{R} \to [\![A]\!] + [\![B]\!]) \to \square.\\
&\quad (\Pi(s:\mathbb{R})\,(x:[\![A]\!]).\,\boxed{P}\;s\;\boxed{(\lambda_:\mathbb{R}.\,\mathtt{inl}\;x)}) \to\\
&\quad (\Pi(s:\mathbb{R})\,(y:[\![B]\!]).\,\boxed{P}\;s\;\boxed{(\lambda_:\mathbb{R}.\,\mathtt{inr}\;y)}) \to\\
&\qquad \Pi(b:\mathbb{R} \to [\![A]\!] + [\![B]\!]).\,\boxed{P}\;r\;\boxed{b}
\end{aligned}$$

Uh-oh

Unfortunately, It's **not possible** to implement **dependent** elimination!

$$[\![\Pi P.\, (\Pi(x : A).\, P\ (\texttt{inl}\ x)) \to (\Pi(y : B).\, P\ (\texttt{inr}\ y)) \to \Pi b : A + B.\, P\ b]\!]$$

$$\equiv$$

$$\begin{aligned}
&\Pi P : \mathbb{R} \to (\mathbb{R} \to [\![A]\!] + [\![B]\!]) \to \square.\\
&\quad (\Pi(s : \mathbb{R})\,(x : [\![A]\!]).\ \boxed{P}\ s\ \boxed{(\lambda\, _ : \mathbb{R}.\, \texttt{inl}\ x)}\,) \to\\
&\quad (\Pi(s : \mathbb{R})\,(y : [\![B]\!]).\ \boxed{P}\ s\ \boxed{(\lambda\, _ : \mathbb{R}.\, \texttt{inr}\ y)}\,) \to\\
&\qquad \Pi(b : \mathbb{R} \to [\![A]\!] + [\![B]\!]).\ \boxed{P}\ r\ \boxed{b}
\end{aligned}$$

P only holds for two constant values but b can be anything!

Uh-oh

Unfortunately, It's **not possible** to implement **dependent** elimination!

$$[\![\Pi P.\,(\Pi(x : A).\,P\ (\mathtt{inl}\ x)) \to (\Pi(y : B).\,P\ (\mathtt{inr}\ y)) \to \Pi b : A + B.\,P\ b]\!]$$

$$\equiv$$

$$\begin{array}{l}\Pi P : \mathbb{R} \to (\mathbb{R} \to [\![A]\!] + [\![B]\!]) \to \square. \\ \quad (\Pi(s : \mathbb{R})\,(x : [\![A]\!]).\ \boxed{P}\ s\ \boxed{(\lambda\,\_ : \mathbb{R}.\,\mathtt{inl}\ x)}\,) \to \\ \quad (\Pi(s : \mathbb{R})\,(y : [\![B]\!]).\ \boxed{P}\ s\ \boxed{(\lambda\,\_ : \mathbb{R}.\,\mathtt{inr}\ y)}\,) \to \\ \qquad \Pi(b : \mathbb{R} \to [\![A]\!] + [\![B]\!]).\ \boxed{P}\ r\ \boxed{b}\end{array}$$

P only holds for two constant values but b can be anything!

Horrible Realization

Dependent elimination is incompatible with effects.

Not All Predicates are Equal

In general through $[\cdot]_r$ predicates have the following type:

$$P : \mathbb{R} \to (\mathbb{R} \to [\![A]\!] + [\![B]\!]) \to \square$$

Not All Predicates are Equal

In general through $[\cdot]_r$ predicates have the following type:

$$P : \mathbb{R} \to (\mathbb{R} \to [\![A]\!] + [\![B]\!]) \to \square$$

Assume there is $\Phi : \mathbb{R} \to [\![A]\!] + [\![B]\!] \to \square$ s.t.

$$P\ r\ b := \Phi\ r\ (b\ r)$$

Not All Predicates are Equal

In general through $[\cdot]_r$ predicates have the following type:

$$P : \mathbb{R} \to (\mathbb{R} \to [\![A]\!] + [\![B]\!]) \to \square$$

Assume there is $\Phi : \mathbb{R} \to [\![A]\!] + [\![B]\!] \to \square$ s.t.

$$P\ r\ b := \Phi\ \ r\ (b\ r)$$

In this case, induction principle becomes

$$\begin{array}{r}(\Pi(s : \mathbb{R})\,(x : [\![A]\!]).\,\Phi\ \ s\ (\mathtt{inl}\ \ x)) \to \\ (\Pi(s : \mathbb{R})\,(y : [\![B]\!]).\,\Phi\ \ s\ (\mathtt{inr}\ \ y)) \to \\ \Pi(b : \mathbb{R} \to [\![A]\!] + [\![B]\!]).\,\Phi\ \ r\ (b\ r)\end{array}$$

This **is** provable!

Not All Predicates are Equal

In general through $[\cdot]_r$ predicates have the following type:

$$P : \mathbb{R} \to (\mathbb{R} \to [\![A]\!] + [\![B]\!]) \to \square$$

Assume there is $\Phi : \mathbb{R} \to [\![A]\!] + [\![B]\!] \to \square$ s.t.

$$P\ r\ b := \Phi\ r\ (b\ r)$$

In this case, induction principle becomes

$$\begin{array}{r}(\Pi(s : \mathbb{R})\,(x : [\![A]\!]).\, \Phi\ s\ (\mathtt{inl}\ x)) \to \\ (\Pi(s : \mathbb{R})\,(y : [\![B]\!]).\, \Phi\ s\ (\mathtt{inr}\ y)) \to \\ \Pi(b : \mathbb{R} \to [\![A]\!] + [\![B]\!]).\, \Phi\ r\ (b\ r)\end{array}$$

This **is** provable!

Induction is still valid for predicates that evaluate **eagerly** their argument.

Linear Dependence is All You Need

We restrict dependent elimination in the following way:

$$\frac{\Gamma \vdash M : \mathbb{B} \qquad \ldots \qquad P \text{ eager in } b}{\Gamma \vdash \texttt{if } M \texttt{ then } N_1 \texttt{ else } N_2 : P\{b := M\}}$$

- Can be underapproximated by a generic, syntactic **guard condition**
- This can be made formal by the notion of linearity.
- The CBN doppelgänger of the dreaded **value restriction** in CBV!
- There is a canonical way to make any predicate linear

Linear Dependence is All You Need

We restrict dependent elimination in the following way:

$$\frac{\Gamma \vdash M : \mathbb{B} \qquad \ldots \qquad P \text{ eager in } b}{\Gamma \vdash \texttt{if } M \texttt{ then } N_1 \texttt{ else } N_2 : P\{b := M\}}$$

- Can be underapproximated by a generic, syntactic **guard condition**
- This can be made formal by the notion of linearity.
- The CBN doppelgänger of the dreaded **value restriction** in CBV!
- There is a canonical way to make any predicate linear

This restriction forms **Baclofen Type Theory**.

Outrageous claim

BTT is the generic theory to deal with dependent effects

The Exceptional Type Theory

(a.k.a. the Curry-Howard-Shadok correspondence)

An extension of CIC rooted in Shadok wisdom.

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

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 dependent elimination

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 dependent elimination
- 😕 Didn't I say this was not possible???

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 dependent elimination
- 😕 Didn't I say this was not possible???

The Exceptional Type Theory: Overview

The exceptional type theory $\mathcal{T}_{\mathbb{E}}$ extends CIC with

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

The Exceptional Type Theory: Overview

The exceptional type theory $\mathcal{T}_{\mathbb{E}}$ extends CIC with

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

The equations are those of call-by-name exceptions.

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

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

The Exceptional Type Theory: Overview

The exceptional type theory $\mathcal{T}_{\mathbb{E}}$ extends CIC with

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

The equations are those of call-by-name exceptions.

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

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

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

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

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

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

The Exceptional Implementation, Negative case

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

Every exceptional type comes with its own implementation of failure!

The Exceptional Implementation, Negative case

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

Every exceptional type comes with its own implementation of failure!

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

$$\begin{array}{lcl}
[\![\square]\!] & \equiv & \Sigma A : \square.\, \mathbb{E} \to A \\
[\square]_\varnothing\ e & \equiv & \ldots \\
[\![\Pi x : A.\, B]\!] & \equiv & \Pi x : [\![A]\!].\, [\![B]\!] \\
[\Pi x : A.\, B]_\varnothing\ e & \equiv & \lambda x : [\![A]\!].\, [B]_\varnothing\ e
\end{array}$$

The Exceptional Implementation, Negative case

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

Every exceptional type comes with its own implementation of failure!

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

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

The Exceptional Implementation, Negative case

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

Every exceptional type comes with its own implementation of failure!

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

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

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

The Exceptional Implementation, Failure

It is straightforward to implement the failure operation.

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

The Exceptional Implementation, Failure

It is straightforward to implement the failure operation.

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

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

The Exceptional Implementation, Failure

It is straightforward to implement the failure operation.

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

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

Computational rules trivially hold!

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

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.

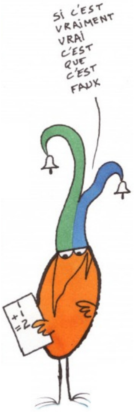
- ☺ A type theory with effects!
- ☺ Compiled away to CIC!
- ☺ Features full conversion
- ☺ Features full dependent elimination

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



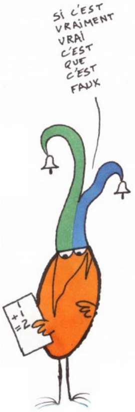
Shadok Logic Strikes Back

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* $M \equiv \mathtt{raise}\ \bot\ e$ *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* $M \equiv \mathtt{raise}\ \bot\ e$ *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!

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!

Cliffhanger

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

Exception

Gotta catch 'em all!

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 $\perp$
- 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.

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

The Curry-Howard-Shadok Correspondence

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

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

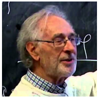
What? That's just **logical relations**.

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**.



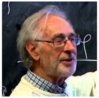
Come on. That's **intuitionistic realizability**.

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**.

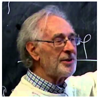
Fools ! That's **parametricity**.

The Curry-Howard-Shadok Correspondence

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

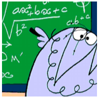
$$f \Vdash A \rightarrow B \quad \equiv \quad \forall x : [\![A]\!]. \quad x \Vdash A \rightarrow f\,x \Vdash B$$

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 $\begin{cases} \vdash_{\text{CIC}} [M] : [\![A]\!] \\ \quad + \\ \vdash_{\text{CIC}} [M]_\varepsilon : [\![A]\!]_\varepsilon\ [M] \end{cases}$

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

A Few Nice Results

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

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.*

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 $M \rightsquigarrow^{*} \bar{n} \in \bar{\mathbb{N}}$.

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.*

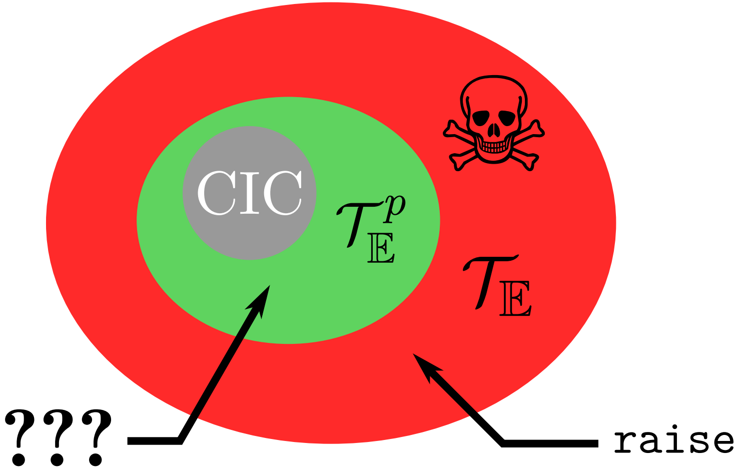
Theorem (Canonicity)

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

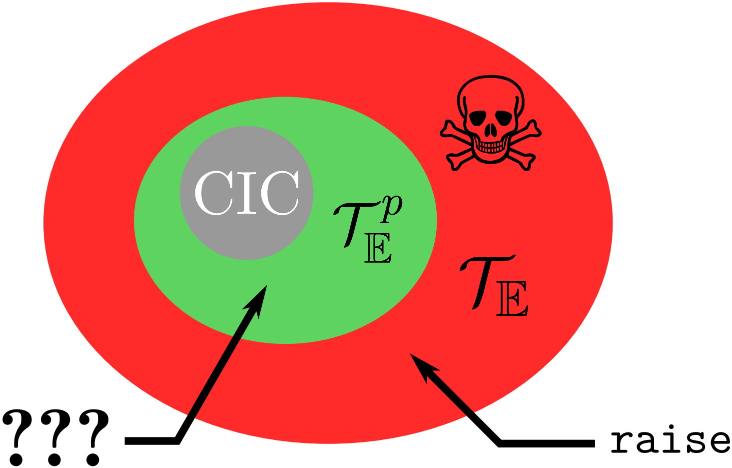
Theorem (Syntax)

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

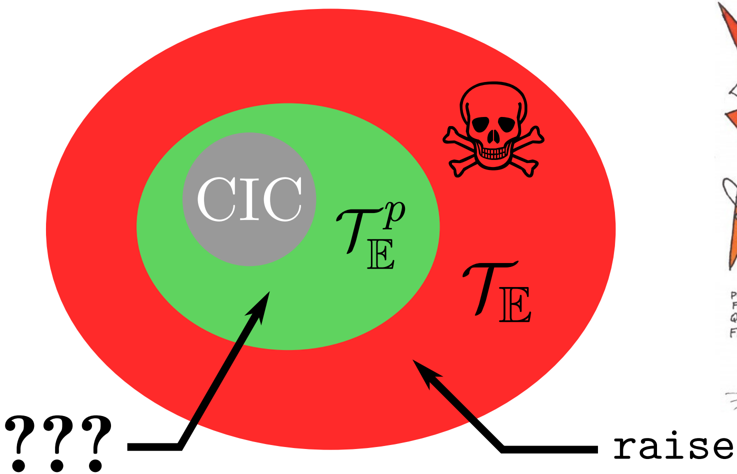
What If There Were No Cake?

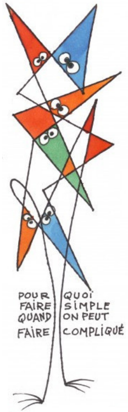


What If There Were No Cake?



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





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

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

Intuitively,

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

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.

Independence of Premises

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

Theorem (CIC + IP)

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

Independence of Premises

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

Theorem (CIC + IP)

$\mathcal{T}^p_{\mathbb{E}}$ *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 $\mathcal{T}^p_{\mathbb{E}}$. □

Stepping Back

Enter Effects

The end of the talk will focus on the big picture.

Enter Effects

The end of the talk will focus on the big picture.

Why did people have so much trouble mixing effects and dependency?

Enter Effects

The end of the talk will focus on the big picture.

Why did people have so much trouble mixing effects and dependency?

Because it's hard.

- Usual models are hard to grasp $\rightsquigarrow$ use syntactic models (done)
- Stuff breaks $\rightsquigarrow$ let's concentrate on that

Conversion

Dependency entails one major difference with simpler types.

Conversion

Dependency entails one major difference with simpler types.

Recall conversion:

$$\frac{A \equiv_\beta B \qquad \Gamma \vdash M : B}{\Gamma \vdash M : A}$$

Conversion

Dependency entails one major difference with simpler types.

Recall conversion:

$$\frac{A \equiv_\beta B \qquad \Gamma \vdash M : B}{\Gamma \vdash M : A}$$

Bad news 1

Typing rules embed the dynamics of programs!

Conversion

Dependency entails one major difference with simpler types.

Recall conversion:

$$\frac{A \equiv_\beta B \qquad \Gamma \vdash M : B}{\Gamma \vdash M : A}$$

Bad news 1

Typing rules embed the dynamics of programs!

Combine that with this other observation and we're in trouble.

Bad news 2

Effects make reduction strategies relevant.

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

What can go wronger?

- Call-by-name: **functions** well-behaved vs. **inductives** ill-behaved
- Call-by-value: **inductives** well-behaved vs. **functions** ill-behaved

What can go wronger?

- Call-by-name: **functions** well-behaved vs. **inductives** ill-behaved
- Call-by-value: **inductives** well-behaved vs. **functions** ill-behaved

In **call-by-name** + effects:

$$(\lambda x.\, M)\ N \equiv M\{x := N\} \quad \rightsquigarrow \quad \textbf{arbitrary substitution}$$

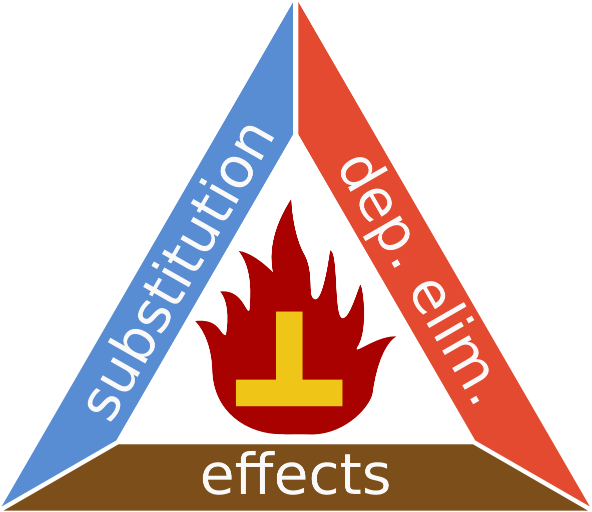
$$(\lambda b : \texttt{bool}.\, M)\ \textbf{fail} \quad \rightsquigarrow \quad \textbf{non-standard booleans}$$

In **call-by-value** + effects:

$$(\lambda x.\, M)\ V \equiv M\{x := V\} \quad \rightsquigarrow \quad \textbf{substitute only values}$$

$$(\lambda b : \texttt{unit}.\, \textbf{fail}\ b) \quad \rightsquigarrow \quad \textbf{invalid } \eta\textbf{-rule}$$

An Incompatibility



Conclusion

- Syntactic models bring semantics to the masses
- You can use them to add effects
- But you have to pick your poison
- Effects can be removed a posteriori for great profit

Scribitur ad narrandum, non ad probandum.

Thanks for your attention.