Taming EFFECTS in a *Dependent* World

Pierre-Marie Pédrot

Max Planck Institute for Software Systems

CSEC Kick-off

8th March 2018

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

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

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

The Most Important Issue of Them All

Yet CIC suffers from a **fundamental** flaw.

Yet CIC suffers from a 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

Yet CIC suffers from a **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?

A Well-known Limitation

This is pretty much standard. By the Curry-Howard correspondence

Intuitionistic Logic ⇔ **Functional** Programming

A Well-known Limitation

This is pretty much standard. By the Curry-Howard correspondence

Intuitionistic Logic ⇔ **Functional** Programming

That means NO EFFECTS in CIC, amongst which:

- no exceptions, state, non-termination, printing...
- ... and thus no Hello World

Dually, for the same reasons, NO CLASSICAL REASONING.

- Curry-Howard principle: effects extend your logic.

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.

Thesis

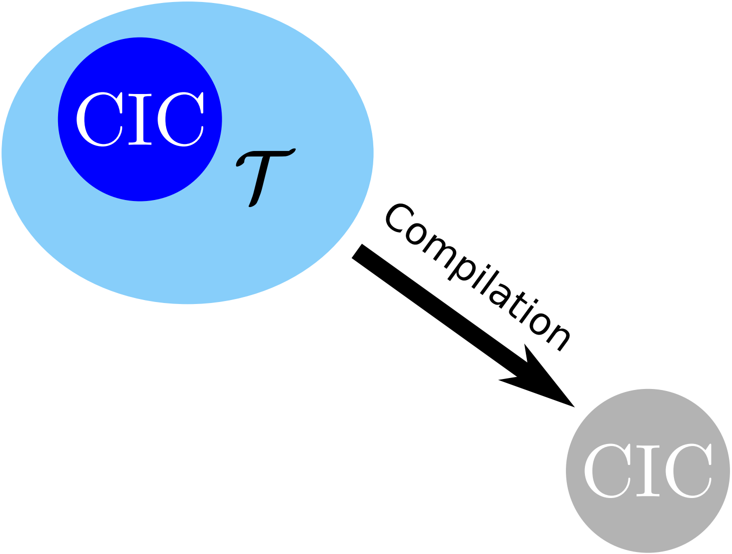
We want a type theory with effects!

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.

We want a model of type theory with effects.

1. The theory ought to be logically consistent
2. It should be implementable (e.g. decidable type-checking)
3. Other nice properties like canonicity ($\vdash n : \mathbb{N}$ implies $n \rightsquigarrow \mathtt{S} \ldots \mathtt{S}\,\mathtt{0}$)



« CIC, the LLVM of Type Theory »

Conversion

Dependency entails one major difference with usual program translations.

Conversion

Dependency entails one major difference with usual program translations.

Meet conversion:

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

Conversion

Dependency entails one major difference with usual program translations.

Meet 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 usual program translations.

Meet 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.

A Though Choice

We have two canonical possibilities in presence of effects.

A Though Choice

We have two canonical possibilities in presence of effects.

Call-by-value	**Call-by-name**
	

Call-by-value

- Usual monadic decomposition
- Understandable semantics
- Values still enjoy canonicity
- Good old ML

Call-by-name

- More complex model (CBPV)
- Counter-intuitive behaviours
- Jeopardizes canonicity
- WTF PLT?

Problem I

Recall conversion:

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

Problem I

Recall conversion:

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

In case you forgot your glasses:

CIC has an CBN equational theory.

Problem I

Recall conversion:

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

In case you forgot your glasses:

CIC has an CBN equational theory.

It's unclear what you can do with CBV dependency...

... and probably type terrorists will start crying foul and calling it heresy.

So we have to stick to CBN to please the conservative reviewers.

Problem II

Assuming rightly I don't care about peer pressure, we have another issue.

Problem II

Assuming rightly I don't care about peer pressure, we have another issue.

Monadic encodings don't scale to dependent types.

Problem II

Assuming rightly I don't care about peer pressure, we have another issue.

Monadic encodings don't scale to dependent types.

The reason lies in the typing of bind:

$$\texttt{bind} : T\ A \rightarrow (A \rightarrow T\ B) \rightarrow T\ B.$$

It's seemingly not possible to adapt it to the dependent case!

$$\texttt{dbind} : \Pi(\hat{x} : T\ A).\,(\Pi(x : A).\,T\ (B\ x)) \rightarrow T\ (B\ ?).$$

Meanwhile, CBPV naturally extends to dependent types.

We also have to stick to CBN for technical reasons.

Life is Life

Like Homer, we're dragged to the horrible CBN side against our will.

Come on, what could possibly go wronger?

Life is Life

Like Homer, we're dragged to the horrible CBN side against our will.

Come on, what could possibly go wronger?

Dependent elimination + CBN effects $\Rightarrow$ inconsistency.

This is the internal counterpart of the lack of canonicity.

Reduction vs. Effects

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

Reduction vs. Effects

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

Why is that?

In call-by-name + effects:

$$(\lambda x.\, M)\ N \equiv M\{x := N\} \quad \rightsquigarrow \quad \text{arbitrary substitution}$$
$$(\lambda b : \texttt{bool}.\, M)\ \mathbf{fail} \quad \rightsquigarrow \quad \text{non-standard booleans}$$

In call-by-value + effects:

$$(\lambda x.\, M)\ V \equiv M\{x := V\} \quad \rightsquigarrow \quad \text{substitute only values}$$
$$(\lambda b : \texttt{unit}.\, \mathbf{fail}\ b) \quad \rightsquigarrow \quad \text{invalid } \eta\text{-rule}$$

Eliminating Addiction to Dependence

Recall that dependent elimination is just the induction principle.

For instance, on the boolean type:

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

This is a statement reflecting canonicity as an internal property in CIC.

Eliminating Addiction to Dependence

Recall that dependent elimination is just the induction principle.

For instance, on the boolean type:

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

This is a statement reflecting canonicity as an internal property in CIC.

But there are effectful closed booleans which are neither `true` nor `false`...

Dependent elimination is *hardcore intuitionistic*.

It makes a very strong assumption about the universe of discourse.

Note also that dependent elimination on Σ-types implies AC...

If there is no solution, there is no problem

Dependent elimination + CBN effects $\Rightarrow$ inconsistency.

Two Easy Ways Out!

If there is no solution, there is no problem

Dependent elimination + CBN effects $\Rightarrow$ inconsistency.

Two Easy Ways Out!

1. Get into rehab: weaken dependent elimination for a linear fix.
2. Embrace inconsistency: truth is a totally overrated social construct.

In the remaining of this talk, we will have a look at one instance of each case, namely **read-only cells** and **exceptions**.

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

The Reader Translation

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

The reader translation extends type theory with

$$\begin{array}{rcl} \mathtt{read} & : & \mathbb{R} \\ \mathtt{into} & : & \square \to \mathbb{R} \to \square \\ \mathtt{enter}_A & : & A \to \Pi r : \mathbb{R}.\, \mathtt{into}\ A\ r \end{array}$$

satisfying a few expected definitional equations.

The Reader Translation

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

The reader translation extends type theory with

$$\begin{array}{rcl} \mathtt{read} & : & \mathbb{R} \\ \mathtt{into} & : & \square \to \mathbb{R} \to \square \\ \mathtt{enter}_A & : & A \to \Pi r : \mathbb{R}.\, \mathtt{into}\ A\ r \end{array}$$

satisfying a few expected definitional equations.

The `into` function has unfoldings on type formers:

$$\begin{array}{lcll} \mathtt{into}\ (\Pi x : A.\, B)\ r & \equiv & \Pi x : A.\, \mathtt{into}\ B\ r & \\ \mathtt{into}\ A\ r & \equiv & A & \text{for positive } A \end{array}$$

and it is somewhat redundant:

$$\mathtt{enter}_\square\ A\ r \quad \equiv \quad \mathtt{into}\ A\ r$$

The Reader Implementation

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

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

The Reader Implementation

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

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

On the other side of the CBPV adjunction:

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

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

The Reader Implementation: Inductive Types

PLT tells us we have to take $[\mathbb{B}]_r \equiv \mathbb{B}$.

- It's possible to implement **non-dependent** pattern matching as usual.
- Preserves definitional computation rules

The Reader Implementation: Inductive Types

PLT tells us we have to take $[\mathbb{B}]_r \equiv \mathbb{B}$.

- It's possible to implement **non-dependent** pattern matching as usual.
- Preserves definitional computation rules

But it's **not possible** to implement **dependent** pattern matching!

$$
\begin{aligned}
& [\![\Pi P : \mathbb{B} \to \square.\, P\,\texttt{true} \to P\,\texttt{false} \to \Pi b : \mathbb{B}.\, P\; b]\!]_r \\
\equiv\;\; & \Pi P : \mathbb{R} \to (\mathbb{R} \to \mathbb{B}) \to \square. \\
& \quad (\Pi s : \mathbb{R}.\, P\; s\; (\lambda \_ : \mathbb{R}.\, \texttt{true})) \to (\Pi s : \mathbb{R}.\, P\; s\; (\lambda \_ : \mathbb{R}.\, \texttt{false})) \to \\
& \quad\quad \Pi b : \mathbb{R} \to \mathbb{B}.\, P\; r\; b
\end{aligned}
$$

P only holds for two specific values but $b : \mathbb{R} \to \mathbb{B}$ can be anything!

We cannot even test in general that b is extensionally one of those values.

Not All Predicates are Equal

For certain predicates $P : \mathbb{R} \rightarrow (\mathbb{R} \rightarrow \mathbb{B}) \rightarrow \square$, induction still valid though.

Not All Predicates are Equal

For certain predicates $P : \mathbb{R} \to (\mathbb{R} \to \mathbb{B}) \to \square$, induction still valid though.

Indeed, if $P\ r\ b \equiv \Phi\ r\ (b\ r)$ for some Φ, the induction principle becomes

$$(\Pi s : \mathbb{R}.\, \Phi\ s\ \mathtt{true}) \to (\Pi s : \mathbb{R}.\, \Phi\ s\ \mathtt{false}) \to \Pi b : \mathbb{R} \to \mathbb{B}.\, \Phi\ r\ (b\ r)$$

which is provable by case-analysis on $b\ r$.

Not All Predicates are Equal

For certain predicates $P : \mathbb{R} \rightarrow (\mathbb{R} \rightarrow \mathbb{B}) \rightarrow \square$, induction still valid though.

Indeed, if $P\ r\ b \equiv \Phi\ r\ (b\ r)$ for some Φ, the induction principle becomes

$$(\Pi s : \mathbb{R}.\, \Phi\ s\ \texttt{true}) \rightarrow (\Pi s : \mathbb{R}.\, \Phi\ s\ \texttt{false}) \rightarrow \Pi b : \mathbb{R} \rightarrow \mathbb{B}.\, \Phi\ r\ (b\ r)$$

which is provable by case-analysis on $b\ r$.

Such predicates evaluate « immediately » their argument b.

They only rely on the resulting **value**!

This property is completely **independent** from the reader effect.

Moi, j'ai dit linéaire, linéaire ? Comme c'est étrange...

Actually we have a generic **semantic** criterion for valid predicates.

Actually we have a generic **semantic** criterion for valid predicates.

LINEARITY.

- Courtesy of G. Munch, rephrased recently by P. Levy.
- Little to do with « linear use of variables »
- Although tightly linked to linear logic

Linearity in a Nutshell

Defined as an (undecidable) equational property of CBN functions.

A function $f : A \to B$ is linear in A if for all $\hat{x} : \texttt{box}\ A$,

$$f\,(\texttt{match}\ \hat{x}\ \texttt{with}\ \texttt{Box}\ x \Rightarrow x) \equiv \texttt{match}\ \hat{x}\ \texttt{with}\ \texttt{Box}\ x \Rightarrow f\,x$$

where

$$\texttt{Inductive box}\ A := \texttt{Box} : A \to \texttt{box}\ A.$$

Linearity in a Nutshell

Defined as an (undecidable) equational property of CBN functions.

A function $f : A \rightarrow B$ is linear in A if for all $\hat{x} : \texttt{box}\ A$,

$$f\,(\texttt{match}\ \hat{x}\ \texttt{with Box}\ x \Rightarrow x) \equiv \texttt{match}\ \hat{x}\ \texttt{with Box}\ x \Rightarrow f\,x$$

where

$$\texttt{Inductive box}\ A := \texttt{Box} : A \rightarrow \texttt{box}\ A.$$

- A CBN $f : A \rightarrow B$ is linear in A if semantically CBV in A.
- Categorically, f linear iff it is an algebra morphism.
- In a pure language, all functions are linear!

Linear Dependence is All You Need

We restrict dependent elimination in the following way:

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

- Can be underapproximated by a **syntactic** criterion
- A new kind of guard condition in CIC
- The CBN doppelgänger of the dreaded **value restriction** in CBV!
- Every predicate can be freely made linear thanks to storage operators

We can generalize this restriction to form **Baclofen Type Theory**.

- Strict subset of CIC
- Works with our **forcing translation** (LICS 2016)
- Works with our **weaning translation** (LICS 2017)
- Prevents Herbelin's paradox: CIC + callcc inconsistent

A Bishop-style Type Theory

We can generalize this restriction to form **Baclofen Type Theory**.

- Strict subset of CIC
- Works with our **forcing translation** (LICS 2016)
- Works with our **weaning translation** (LICS 2017)
- Prevents Herbelin's paradox: CIC + callcc inconsistent

BTT is the generic theory to deal with dependent effects
« Bishop-style, effect-agnostic type theory »

(Take that, Brouwerian HoTT!)

Exception

Gotta catch 'em all!

That's *literally* what we are going to do.

The Exceptional Type Theory: Overview

The exceptional type theory extends vanilla CIC with

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

The Exceptional Type Theory: Overview

The exceptional type theory extends vanilla CIC with

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

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

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

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

The Exceptional Type Theory: Overview

The exceptional type theory extends vanilla CIC with

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

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

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

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

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

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

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

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

The Exceptional Implementation, Negative case

Intuitive idea: translate every $A : \square$ into $[A] : \Sigma A : \square.\, \mathbb{E} \to A$.

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

The Exceptional Implementation, Negative case

Intuitive idea: translate every $A : \square$ into $[A] : \Sigma A : \square.\, \mathbb{E} \rightarrow A$.

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

Because CBN, trivial on the negative fragment:

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

The Exceptional Implementation, Positive case

The really interesting case is the inductive part of CIC.

How to implement e.g. $[\mathbb{B}]_\varnothing : \mathbb{E} \to [\![\mathbb{B}]\!]$? Or worse $[\bot]_\varnothing : \mathbb{E} \to [\![\bot]\!]$?

The Exceptional Implementation, Positive case

The really interesting case is the inductive part of CIC.

How to implement e.g. $[\mathbb{B}]_\varnothing : \mathbb{E} \to [\![\mathbb{B}]\!]$? Or worse $[\bot]_\varnothing : \mathbb{E} \to [\![\bot]\!]$?

Very simple: 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

The really interesting case is the inductive part of CIC.

How to implement e.g. $[\mathbb{B}]_\varnothing : \mathbb{E} \to [\![\mathbb{B}]\!]$? Or worse $[\bot]_\varnothing : \mathbb{E} \to [\![\bot]\!]$?

Very simple: 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.

$$\begin{array}{ll} & [\![\Pi P : \mathbb{B} \to \square.\, P\ \texttt{true} \to P\ \texttt{false} \to \Pi b : \mathbb{B}.\, P\ b]\!] \\ \cong & \Pi P : [\![\mathbb{B}]\!] \to [\![\square]\!].\, P\ [\texttt{true}] \to P\ [\texttt{false}] \to \Pi b : [\![\mathbb{B}]\!].\, P\ b \end{array}$$

- 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 $[P\ b]_\varnothing\ e$

Logic Strikes Back

Theorem

The exceptional translation interprets all of CIC.

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

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 $\vdash M : \bot$, *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 $\vdash M : \bot$, *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 exceptional 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 $\vdash M : \bot$, *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 exceptional 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 $\vdash M : \bot$, *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 exceptional programs!

Cliffhanger

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

And now for a little ad before the continuing 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 $\vdash_{\mathrm{CIC}} \Pi p : P.\ Q$.

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 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 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-Failure Correspondence

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

The Curry-Howard-Failure 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-Failure Correspondence

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

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

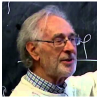
What? That's just **logical relations**.

The Curry-Howard-Failure 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**.

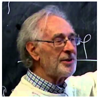


Come on. That's **intuitionistic realizability**.

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**.

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 $\left\{\begin{array}{c} \vdash_{\mathrm{CIC}} [M] : [\![A]\!] \\ + \\ \vdash_{\mathrm{CIC}} [M]_\varepsilon : [\![A]\!]_\varepsilon\ [M] \end{array}\right.$

where $[\![A]\!]_\varepsilon : [\![A]\!] \rightarrow \square$ 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.

Less Is More

Spoiler

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

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

Kreisel realizability extends arithmetic with Independence of Premises.

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

CIC^{++}

Using the same tricks as in Kreisel realizability:

Axiom of choice is provable in $\mathcal{T}_{\mathbb{E}}^{p}$. (It's already in CIC...)

Independence of premises is provable in $\mathcal{T}_{\mathbb{E}}^{p}$! (Using local exceptions.)

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

Function extensionality is disprovable in $\mathcal{T}_{\mathbb{E}}^{p}$!

$$\vdash_{\mathcal{T}_{\mathbb{E}}^{p}} (\lambda i : \mathtt{unit}.\, i) \neq (\lambda i : \mathtt{unit}.\, \mathtt{tt})$$

Implementations

Thanks to the fact we build syntactic models, we can implement them in Coq through a plugin.

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

- Allows to add effects to Coq just today.
- Implement your favourite effectful operators...
- Compile effectful terms on the fly.
- Allows to reason about them in Coq.

Conclusion

- Effects and dependency: not that complicated if sticking to CBN.
  - But a trade-off about dependent elimination
  - Inconsistency vs. linear dependent elimination
- Even inconsistent theories have practical interest.
  - Exceptions enlarge the dynamic behaviour of your proofs
  - Provide an unsafe hatch that can be used in a safe context
- An experimentally confirmed notion of effectful type theories, BTT
  - Works for forcing, weaning (and `callcc`?)
  - Restriction of dependent elimination on linearity guard condition
  - Conjecture: the correct way to add effects to TT
- Implementation of plugins in Coq: try it out.

Thanks for your attention.